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 non-Euclidean data (graphs)
- GNN: a type of DNN that 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 feature vectors from neighbors/edges
    - **Sparse** connections
    - **Irregular** memory access patterns
    - **Poor** locality
    - **Memory intensive**
    - **Variable** execution time for each vertex, correlated with the vertex’s degree
  - **Update**: each vertex computes its output features from the aggregation outputs with a deep learning operator (e.g. MLP)
    - **Dense** computation
    - **Regular** memory access patterns
    - **Good** locality
    - **Compute intensive**
    - **Fixed** execution time for each vertex

\[
\begin{align*}
  a_v^k &= \text{AGGREGATE}(h_u^{(k-1)} | \forall u \in \mathcal{N}(v) \cup \{v\}) \\
  h_v^k &= \text{UPDATE}(a_v^k)
\end{align*}
\]
GNN Characteristic: Activation (Feature) Sparsity

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

Example: feature sparsity during 3-layer GraphSAGE 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
Contribution: Graphite

- Graphite: cooperative SW-HW techniques that optimize GNNs on CPUs
- Software techniques:
  - Overlapping the memory access in aggregation with the compute in update with layer fusion
  - Compressing the sparse activation to reduce memory traffic
  - Increasing the locality of the irregular memory accesses in aggregation
  - 1.60x-2.64x speedup
- HW-SW co-design techniques:
  - Enhancing the CPU DMA engine to offload GNN aggregation
  - 1.63x-3.14x speedup
Graphite Software Techniques
Basic Optimized Implementation

- **Aggregation**
  - JIT-generated assembly kernel
  - Output parallelized
  - Hand vectorized
  - Uses OpenMP dynamic scheduling
  - Software prefetch

- **Update**
  - Stock library GEMM

Diagram:
- Batch of vertices `dynamically` distributed to threads.
- All vertices `evenly` distributed to threads.
- Time and barrier.
- Batch of vertices, aggregation, update.
Layer Fusion

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

- Prefetches the features needed by the aggregation in the next batch
- The ongoing prefetch overlaps with the update
Overlapping Compute-Memory: Among Cores

- **Aggregation**: variable time
- **Update**: fixed time
- Executions on different processors naturally go out-of-phase
Feature Compression

- Goal: avoid loading/storing zeros by employing fast online (de)compression
- Uses vector comparison and (de)compression instructions

**Compression**

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

- **Step 2: bubble collapse**
  - Compressed vector: 10 7 0 43 0 0 0 22
  - Decompressed vector: 1 1 0 1 0 0 0 1

**Decompression**

- **Step 2: bubble expand**
  - Compressed vector: 10 7 0 43 0 0 0 22
  - Decompressed vector: 10 7 0 43 0 0 0 22
Increasing Locality in Aggregation

- Aggregation: each vertex gathers features from its neighbors
  - Features span multiple cache lines
  - Temporal locality of a feature is important

- Goal: increase the temporal reuse of vertex features by preprocessing the inputs
Increasing Locality in Aggregation: Algorithm

- Computes a new processing order of vertices
- Grouping: assigns each vertex to the group of its highest-degree neighbor
- All vertices in the same 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: Algorithm

- Computes a new processing order of vertices
- Grouping: assigns each *vertex* to the group of its *highest-degree neighbor*
- All vertices in the same 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
- All vertices in the same 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 brings input features into L1 and does simple reductions
  - Similar to gather and reduce
  - Input features have low reuse
- Scatter-gather is a common DMA operation
- Graphite augments the scatter-gather DMA to perform aggregation
Graphite DMA Structure

- Each processor is attached with a DMA engine
  - Connected the on-chip interconnect
  - Works in user-space
  - Works with virtual address: uses L2 STLB for address translation
- Descriptor-based programming model
  - Each 64B descriptor encodes an entire aggregation
  - Easily built from CSR encoded graph adjacency matrices
- Reuses function units in the existing DMA engine
- Adds a narrow vector unit to perform reductions
- Incompatible with feature compression for cost reason
DMA Aggregation
DMA Aggregation

core issues descriptor

Core + L1 -> DMA

L2

L3 + directory

Core + L1 -> DMA

L2

L3 + directory

Core + L1 -> DMA

L2

L3 + directory

NoC
DMA Aggregation

DMA gathers input features

Core + L1
L2
DMA
L2
DMA
Core + L1
L2
DMA
Core + L1
L2
DMA
L3 + directory
L3 + directory
L3 + directory

NoC
DMA Aggregation

DMA reduces and writes results to L2

Core + L1 → L2 DMA → L3 + directory

Core + L1 → L2 DMA → L3 + directory

Core + L1 → L2 DMA → L3 + directory

NoC
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

**Average inference speedup**
- Basic optimized implementation: 1.07
- Layer fusion: 1.35
- Feature compression: 1.45
- Layer fusion feature compression: 1.81

**Average training speedup**
- Basic optimized implementation: 1.07
- Layer fusion: 1.18
- Feature compression: 1.40
- Layer fusion feature compression: 1.55
- Layer fusion feature compression locality optimization: 1.88
Performance: HW+SW Techniques

- **Average inference speedup**
  - Fusion: 1.30
  - Fusion DMA: 1.79

- **Average training speedup**
  - Fusion: 1.24
  - Fusion DMA: 1.53
  - Fusion locality: 1.83
  - Fusion DMA locality: 2.43
Conclusion

- GNN workloads on CPUs are often 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 CPU DMA engine to offload GNN aggregation
- Evaluated with 28 cores
  - SW-only techniques: 1.60x-2.64x speedup (native)
  - HW+SW techniques: 1.63x-3.14x speedup (simulated)

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