Let's Start With Some Questions…

1. Can graphs be easily processed by DNNs (Deep Neural Networks)?

   No!

2. What is the solution to this problem?

   Graph Neural Networks (GNNs)

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

Slides adapted from [2] and presented by Fiona Fisher and Maxence Cumer

March 26, 2025

The Authors

Zhangxiaowen Gong, *UIUC & Intel*

Yao Yao, *UIUC*

Christopher J. Hughes, *Intel*

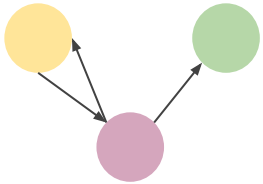
Houxiang Ji, *UIUC*

Christopher W. Fletcher, *UIUC*

Josep Torrellas, *UIUC*

The Big Idea: Processing *Graphs* on *Graph Neural Networks* with *CPUs*

Graphs



- Graphs are **data structures** notated by **edges** and **nodes/vertices**.
  - In practice, the nodes/vertices are **objects** (e.g., a online product) and their edges are **relationships** (e.g., what a customer is most likely to buy after that product).
- Graphs have many use-cases in modern computing [3].
  - e.g., friend networks on social media platforms, citation networks for online research paper repositories, flight routes and planning…
- Traditional Deep Neural Networks (DNNs) cannot compute with graphs, because they are **non-Euclidean**.
  - *Non-euclidean* means graphs have “curvature”, unlike other data sets, like a grid of pixels, that DNNs can handle [4].

GNN Characteristic: Alternating Phases

- Two alternating phases (per layer): **Aggregation** and **Update**.

- **Aggregation**: each vertex gathers and reduces features 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.

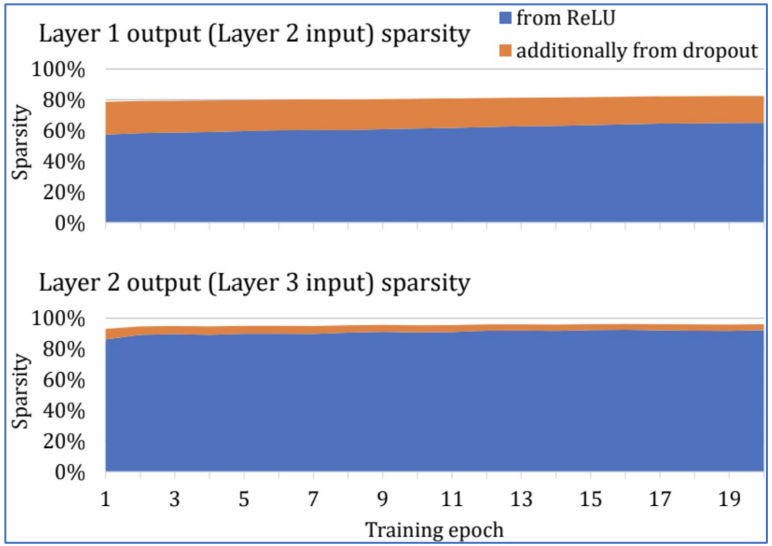
$$\mathbf{a}_v^k = \text{AGGREGATE}(\mathbf{h}_u^{(k-1)} \mid \forall u \in \mathcal{N}(v) \cup \{v\})$$

- **Update**: each vertex computes its output features from the aggregation outputs with a DL op (e.g. MLP).
  - **Dense** computation.
  - **Regular** memory access patterns.
  - **Good** locality.
  - **Compute intensive.**
  - **Similar** execution time for each vertex.

$$\mathbf{h}_v^k = \text{UPDATE}(\mathbf{a}_v^k)$$

GNN Characteristic: Activation (Feature) Sparsity

- **Sparsity**: zeros in the working sets.
  - **ReLU**: Layer activation function that sets any negative number to 0.
  - **Feature dropout**: ML technique to reduce overfitting. During training, some features (usually 50%) are set at zero.
  - Graphs are also often inherently sparse.
- Feature sparsity is dynamic & unstructured.
  - Costly to frequently compress.
- The problem: operating on zeros is ineffectual.



Example: feature sparsity during 3-layer GraphSAGE training on the ogbn-products dataset.

Other GNN Characteristics

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

Dataset	Vertex feature length
Cora	1,433
Citeseer	3,703
Reddit	602
Ogbn-products	100

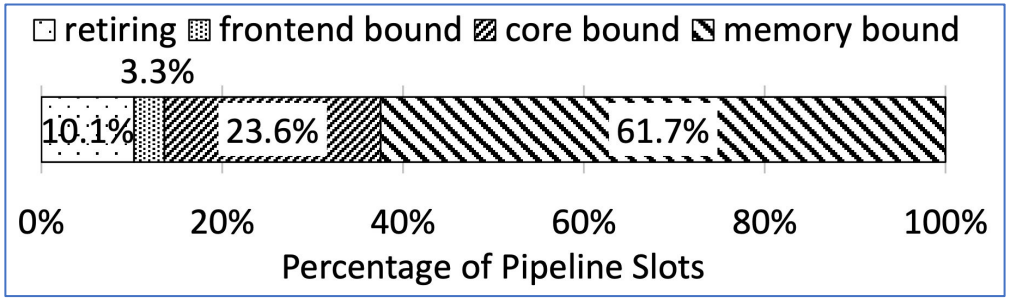
- Reuses input graphs in training.

Motivation: GNNs on CPUs

DNNs are typically run on GPUs or accelerators. Why switch to CPUs for GNNs?

1. CPUs are very common.
   a. GNN tasks could be performed on the same machine as other tasks.
2. CPUs have terabyte-level memory capacity.
   a. Real-world graphs are very large. Often millions to billions of vertices and edges.

- GNNs on CPUs are memory bandwidth bound.



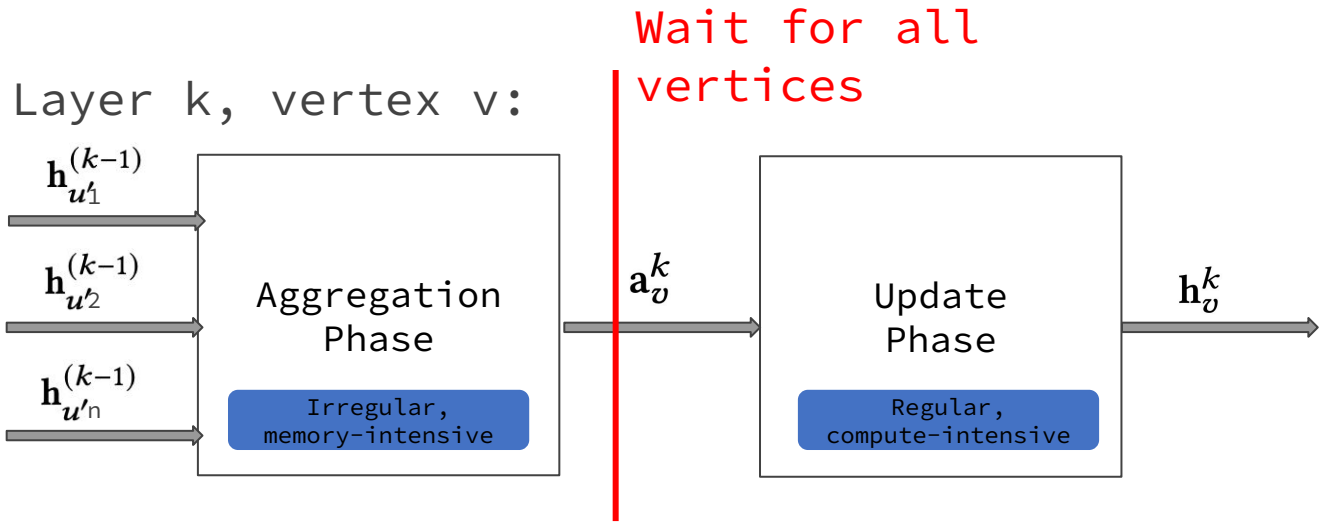
And Finally: Graphite!

Graphite is a group of cooperative hardware and software techniques designed to optimize running GNNs on CPUs.

- **Software techniques** that speedup inference by 1.8x & training by 1.9x:
  - Layer fusion to overlap compute and memory.
  - Feature compression to reduce memory traffic.
  - Input preprocessing to increase locality.
- **HW-SW codesign techniques** that speedup inference by 1.8x & training by 2.4x:
  - Enhanced DMA engine to offload aggregation.

Graphite Software Techniques

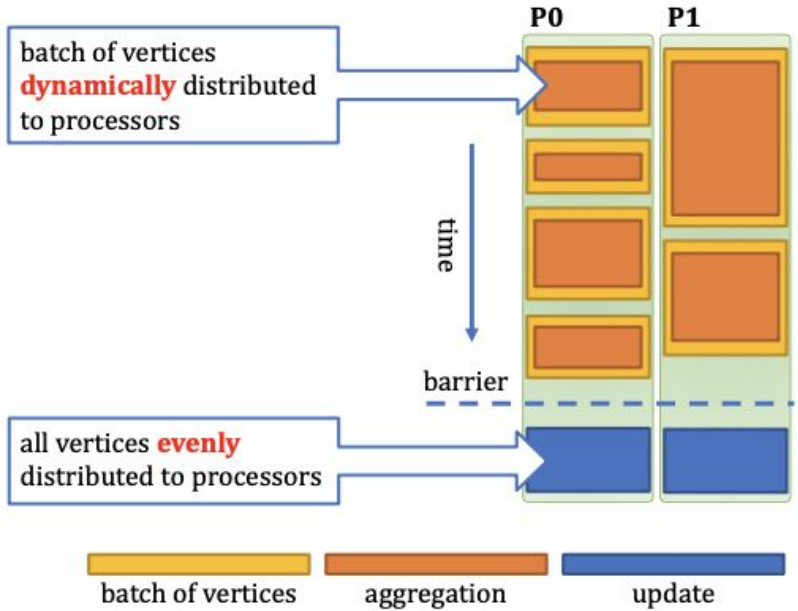
Basic Optimized Implementation



The aggregation vectors of all vertices are **independent**

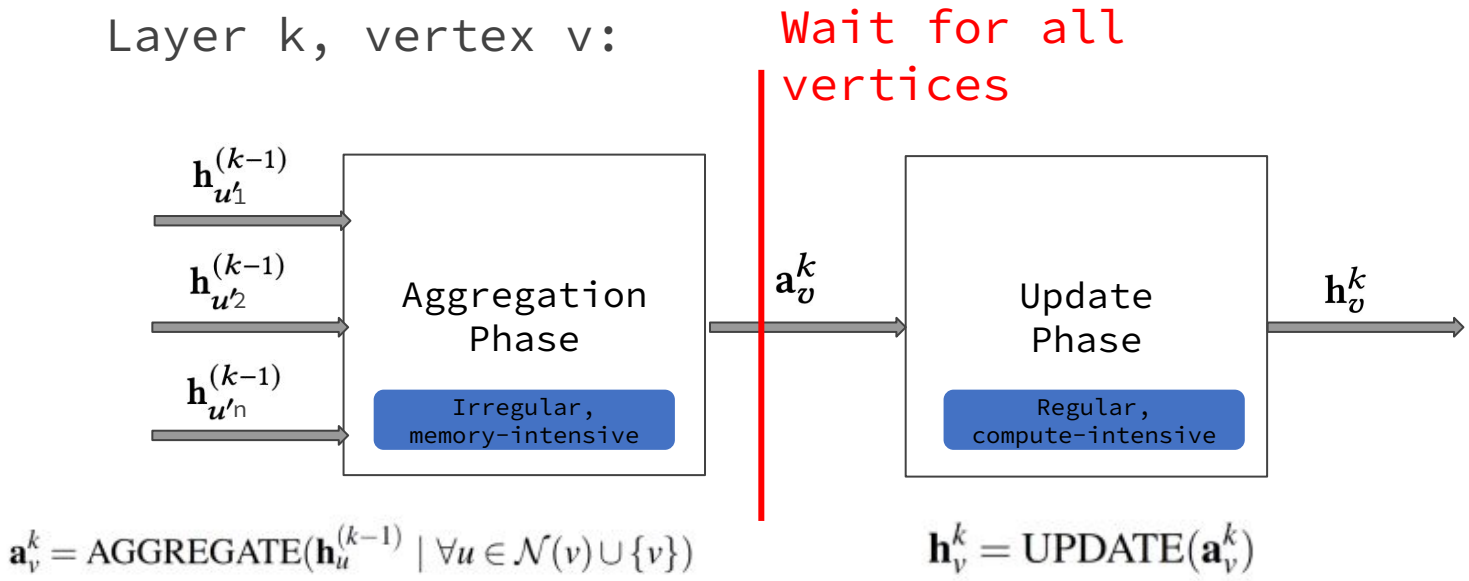
Basic Optimized Implementation

- Create different threads:
  - Each thread will process the aggregation phase for a subset of vertices
  - In the end of the aggregation phase, the feature vectors of each vertex will be updated



Why do we need a synchronization barrier before the update?

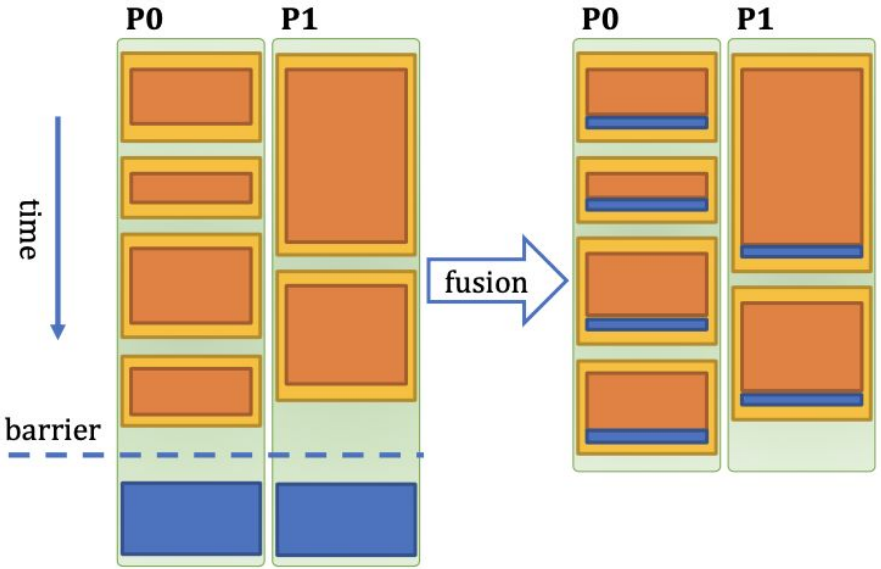
Basic Optimized Implementation



The feature vector of a vertex **only** depends on its aggregation vector

Layer Fusion

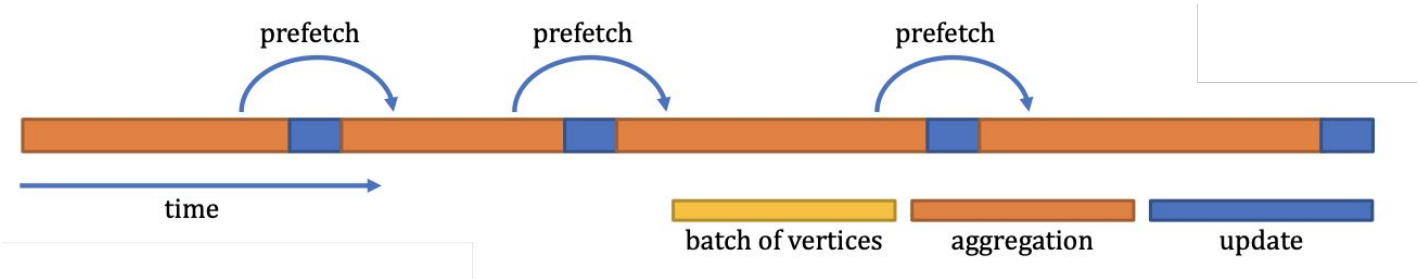
- Goal: overlap memory-bound and compute-bound operations
- Fusion: interleave aggregation and update of vertex batches

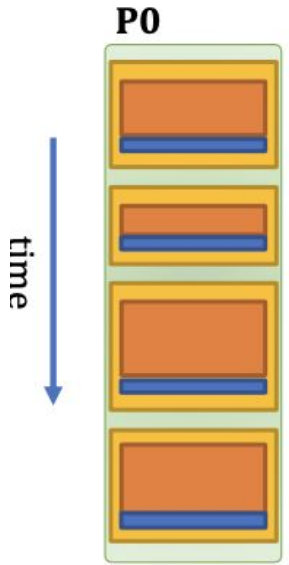


The feature vector of a vertex **only** depends on its aggregation vector

Overlapping Compute-Memory

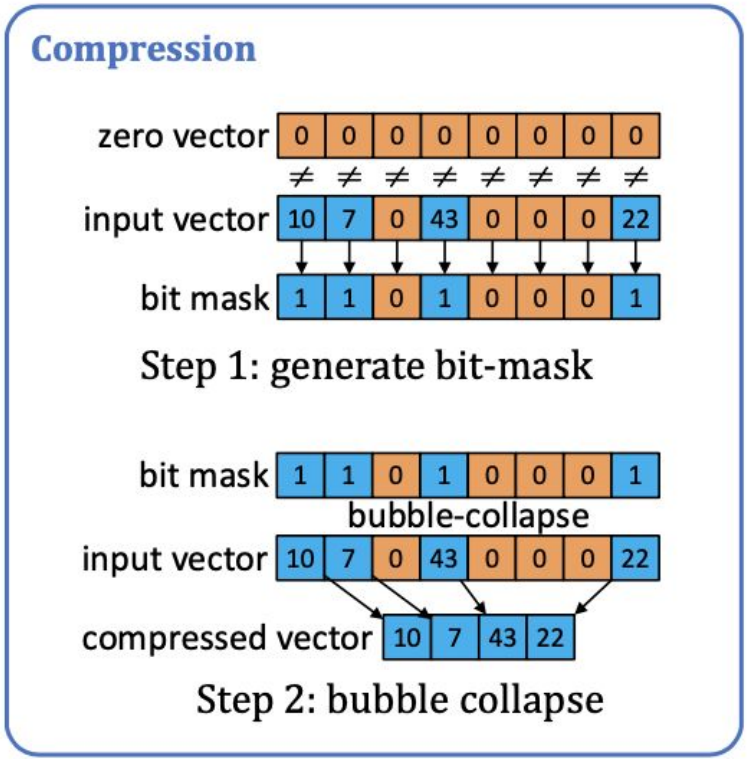
- Prefetches the features needed by the aggregation of the next batch
- Ongoing prefetch overlaps with the update



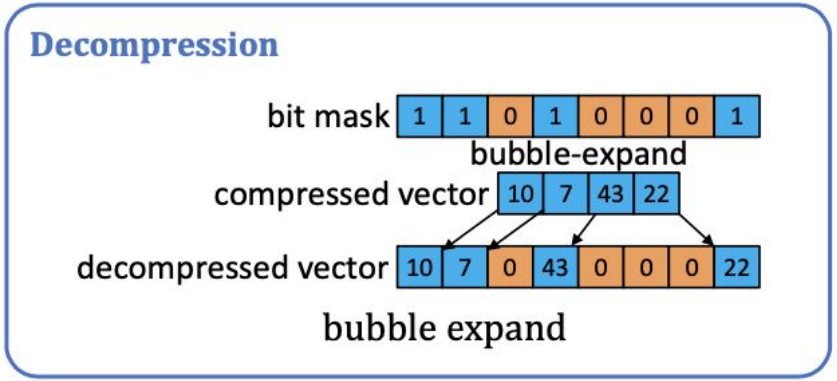


In multi-core systems, memory bandwidth is a shared resource

Feature Compression



- Goal: reduce memory traffic
- Avoid loading/storing zeros
- Fast vector compression and decompression instructions



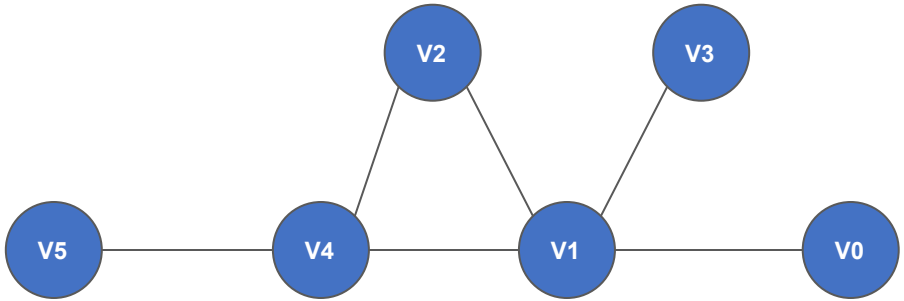
Increasing Locality in Aggregation

$$\mathbf{a}_v^k = \text{AGGREGATE}(\mathbf{h}_u^{(k-1)} \mid \forall u \in \mathcal{N}(v) \cup \{v\})$$

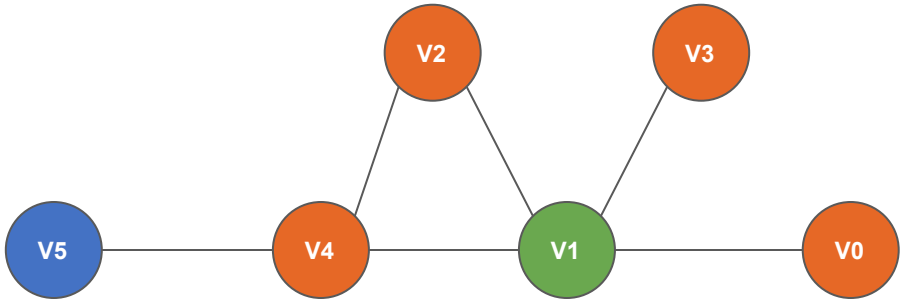
- Goal: increase temporal reuse of vertex features
- Idea:
  - Compute a new processing order of vertices
  - Assign each vertex to the group of its highest-degree neighbor
  - Vertices in a group are processed temporally closely and reuse at lease one feature vector

The cost of preprocessing the inputs is amortized in GNN training

Increasing Locality in Aggregation

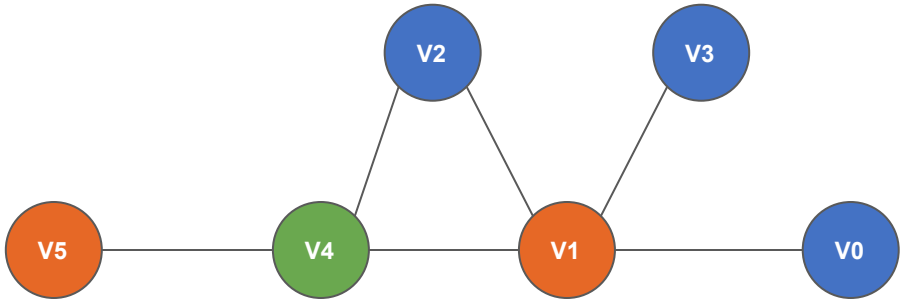


Increasing Locality in Aggregation



Group of V1:
V0, V2, V3, V4

Increasing Locality in Aggregation



Group of V1:
V0, V2, V3, V4

Group of V4:
V1, V5

Linear complexity:
O(V+E)
Good scalability

The cost of preprocessing the inputs is amortized in GNN training

- Original processing order: V0, V1, V2, V3, V4, V5
- New processing order: V0, V2, V3, V4, V1, V5

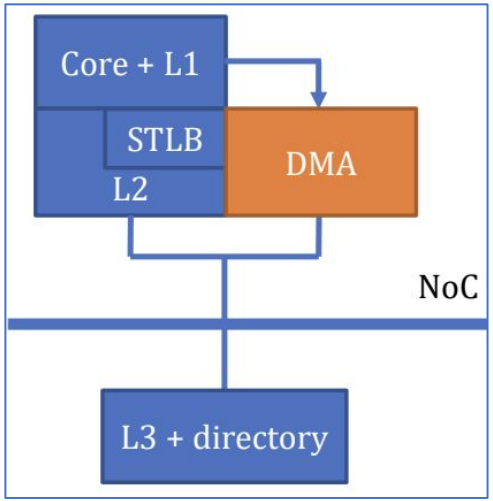
Graphite HW-SW Co-design Techniques

GNN Aggregation and DMA

- Scatter-gather is a common Direct Memory Access (DMA) operation.
  - Aggregation is a gather *and reduce* operation.
  - Graphite **enhances DMA** to perform aggregation.
- **Incompatible with feature compression.**
  - Compression hardware is costly.
  - Only useful for GNNs which implement ReLU and/or feature dropout.

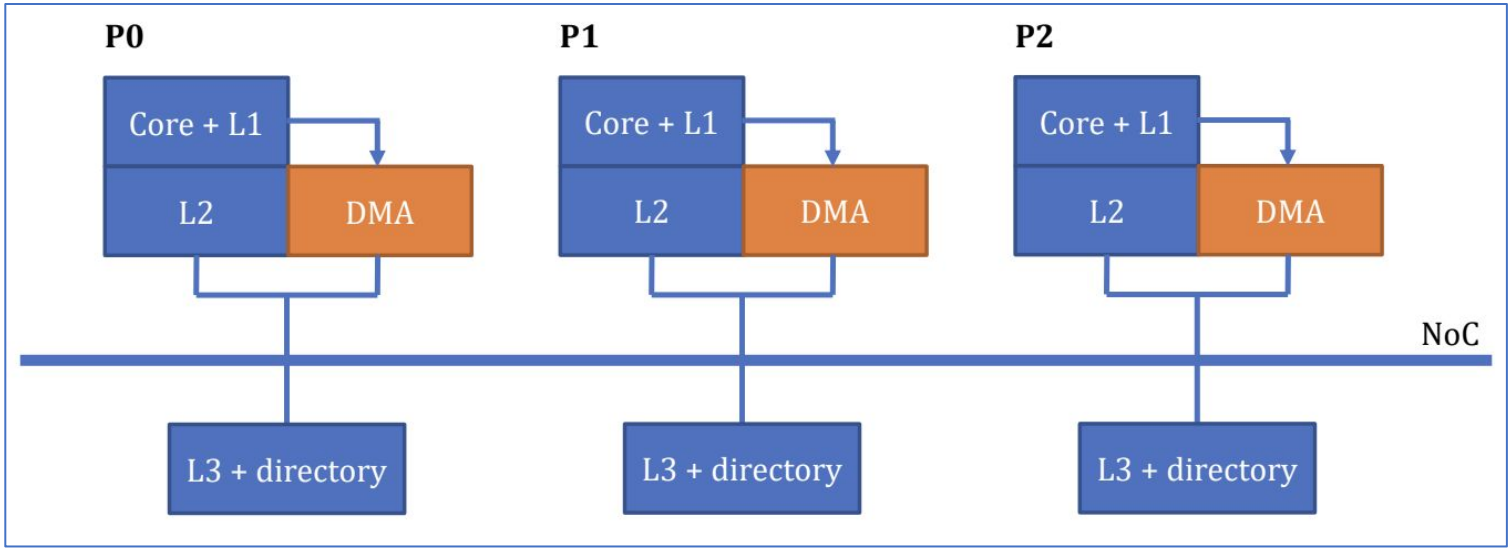
Graphite DMA Structure

- Each core has a DMA engine attached to the L2 cache.
  - Easy access to the STLB for virtual address translation.
  - Aggregation results can be quickly transferred.
  - Benefits from the locality of the shared L3.
- Graphite reuses function units in existing DMA.
  - Adds a narrow vector unit to perform reductions.
- Uses a descriptor-based programming model.
  - 64B descriptor encodes an aggregation.
  - Easily built from CSR encoded adjacency matrices.



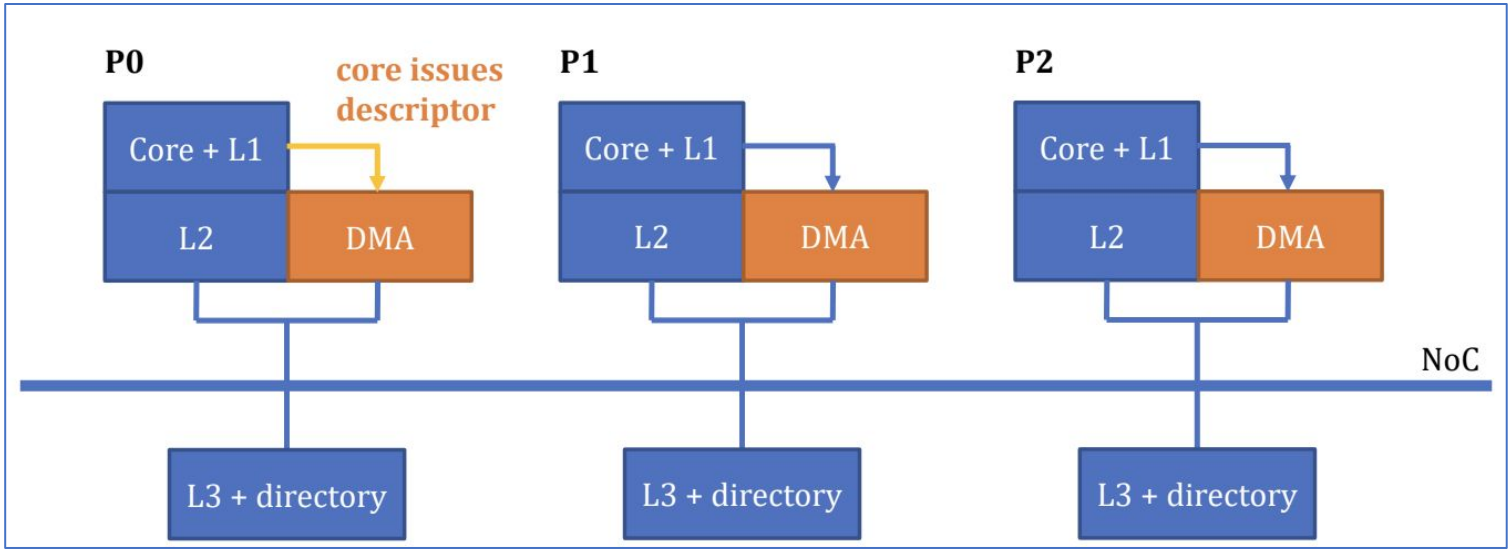
DMA Aggregation

One DMA engine per processor, connected via the NoC.



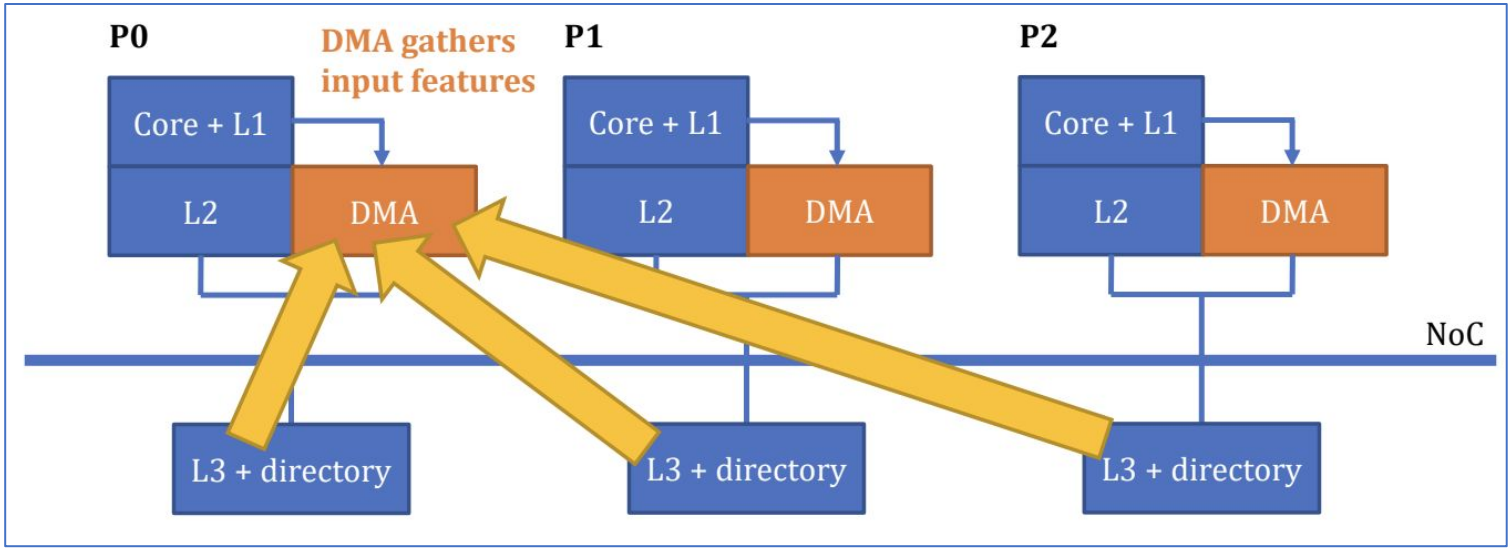
DMA Aggregation: The Descriptor

- In a traditional DMA, the descriptor identifies the gather operation.
  - In GNNs, the data blocks (feature vectors) are small.
- In Graphite, the descriptor identifies the entire aggregation.



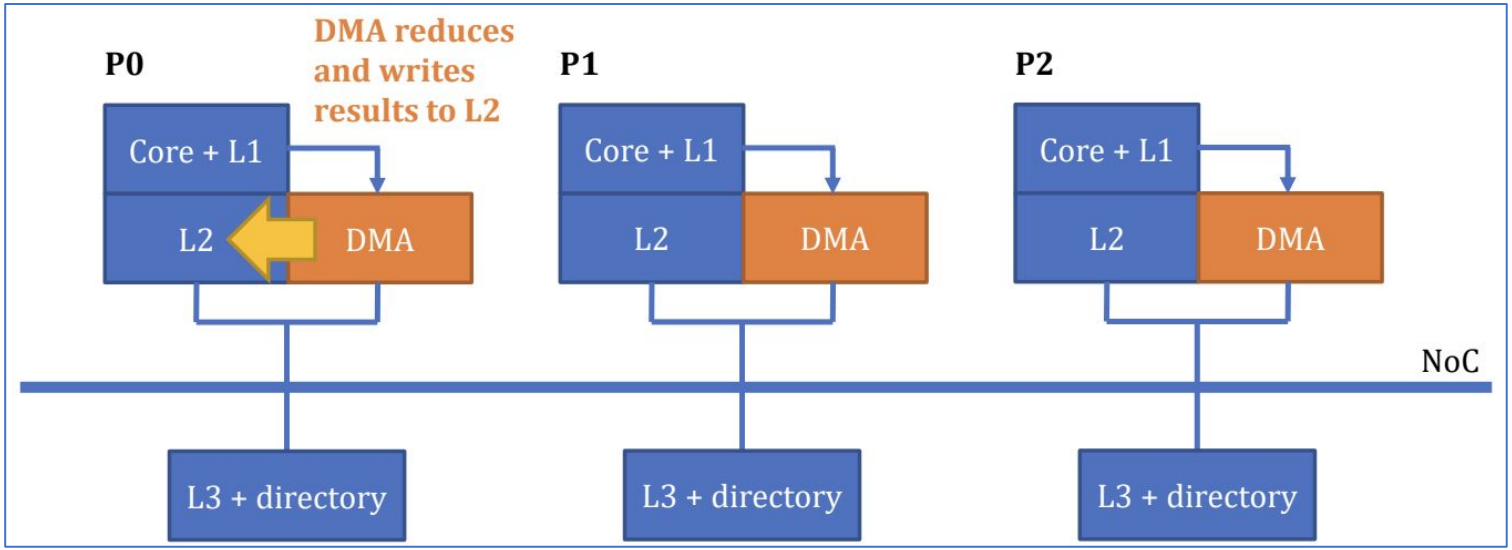
DMA Aggregation: The Operation

No cache coherency concerns with this design due to the read-only nature of the input features.



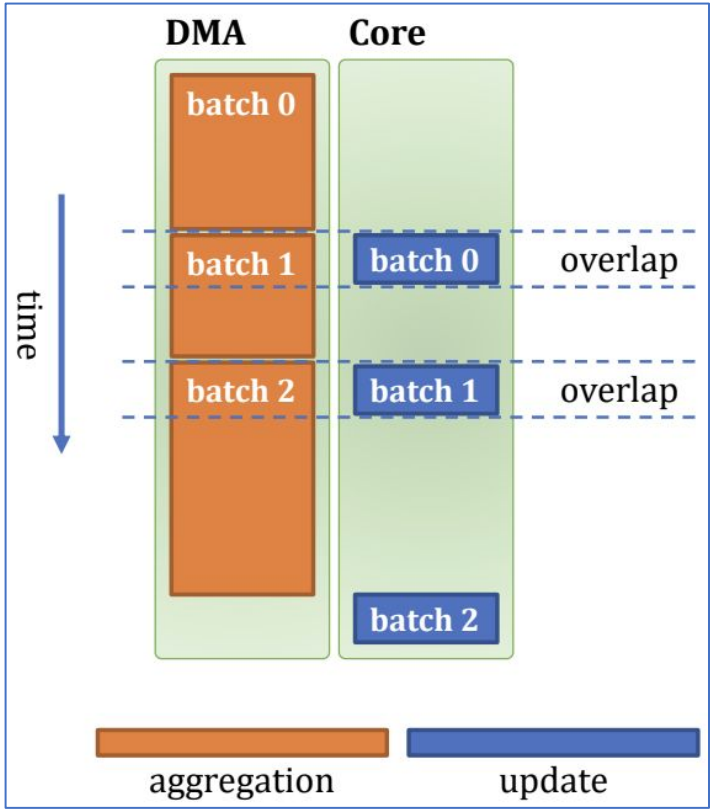
DMA Aggregation: The Operation

At beginning of the aggregation phase, the relevant L2 lines are pre-fetched to avoid miss latency when they are written back.



Putting It Together: DMA-Assisted Layer Fusion

- On each processor:
  - DMA handles aggregation stage.
  - Core handles update stage.
- The **update** of a vertex batch overlaps with the **aggregation** of the next vertex batch.

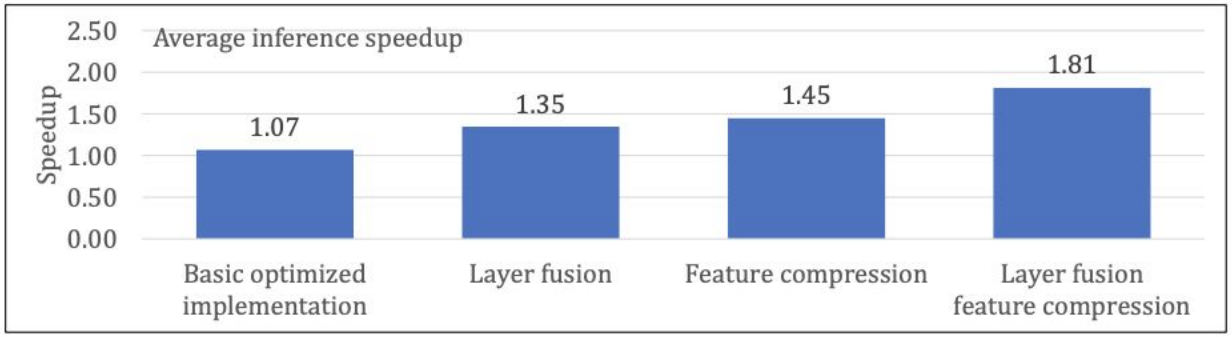


Evaluation

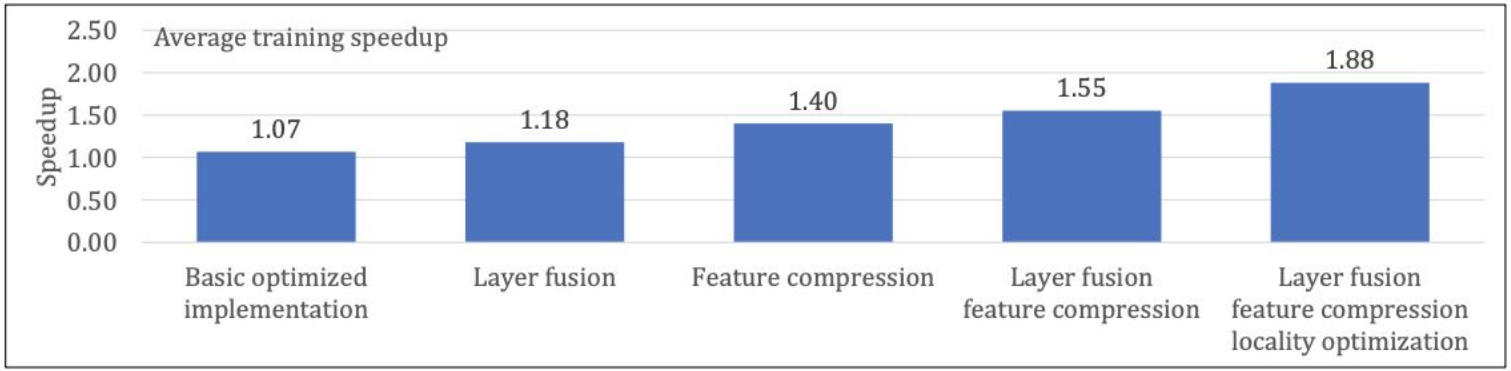
Evaluation Setup

- 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[6] + MKL GEMM
- Evaluation:
  - SW-only techniques: 28-core Cascade Lake server running 28 threads
  - HW+SW techniques: Sniper[7] multi-core simulator simulating the 28-core server

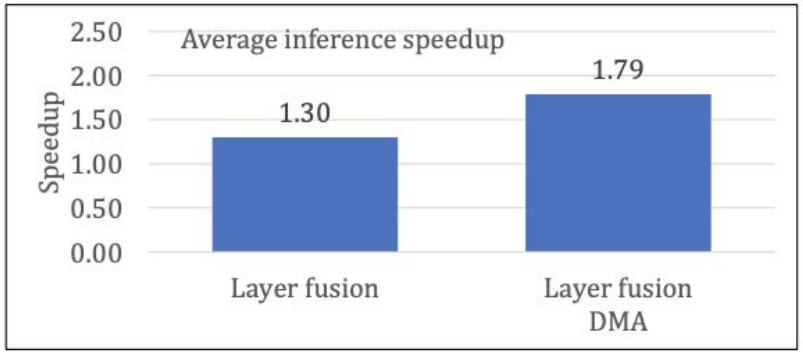
Performance: SW-only techniques



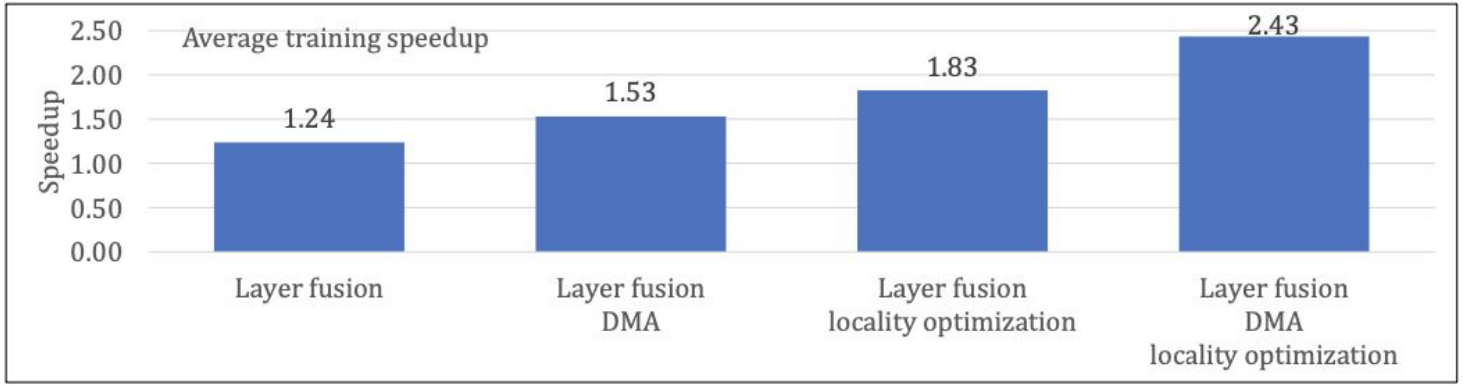
- Feature compression @ 50% sparsity
- Locality optimization only on training
- Techniques are synergetic



Performance: HW+SW techniques



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



Conclusion

Conclusion

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

- 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)

References

[1] Zhangxiaowen Gong, Houxiang Ji, Yao Yao, Christopher W. Fletcher, Christopher J. Hughes, and Josep Torrellas. 2022. Graphite: Optimizing Graph Neural Networks on CPUs Through Cooperative Software-Hardware Techniques. In *The 49th Annual International Symposium on Computer Architecture (ISCA '22)*, June 18–22, 2022, New York, NY, USA. ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3470496.3527403

[2] Gong, Z., Ji, H., Yao, Y., Fletcher, C., Hughes, C., & Torrellas, J. (2022). *Graphite: Optimizing Graph Neural Networks on CPUs Through Cooperative Software-Hardware Techniques*. https://www.iscaconf.org/isca2022/slides/isca22-gong-graphite.pdf

[3] *Real-Life Applications of Graphs*. (2024, April 9). GeeksforGeeks. https://www.geeksforgeeks.org/real-life-applications-of-graphs/

[4] Kurniadi, E. (2011, November 14). *The Difference Between Euclidean and Non Euclidean Geometry*. Elika Kurniadi. https://elikakurniadi.wordpress.com/2011/11/14/the-difference-between-euclidean-and-non-euclidean-geometry/

[6] Vasimuddin Md, et al. 2021. DistGNN: scalable distributed training for large-scale graph neural networks. SC'21

[7] Trevor E. Carlson, et al. 2011. Sniper: Exploring the Level of Abstraction for Scalable and Accurate Parallel Multi-Core Simulations. SC'11

Questions?