BERKELEY LAB
Bringing Science Solutions to the World

U.S. DEPARTMENT OF ENERGY

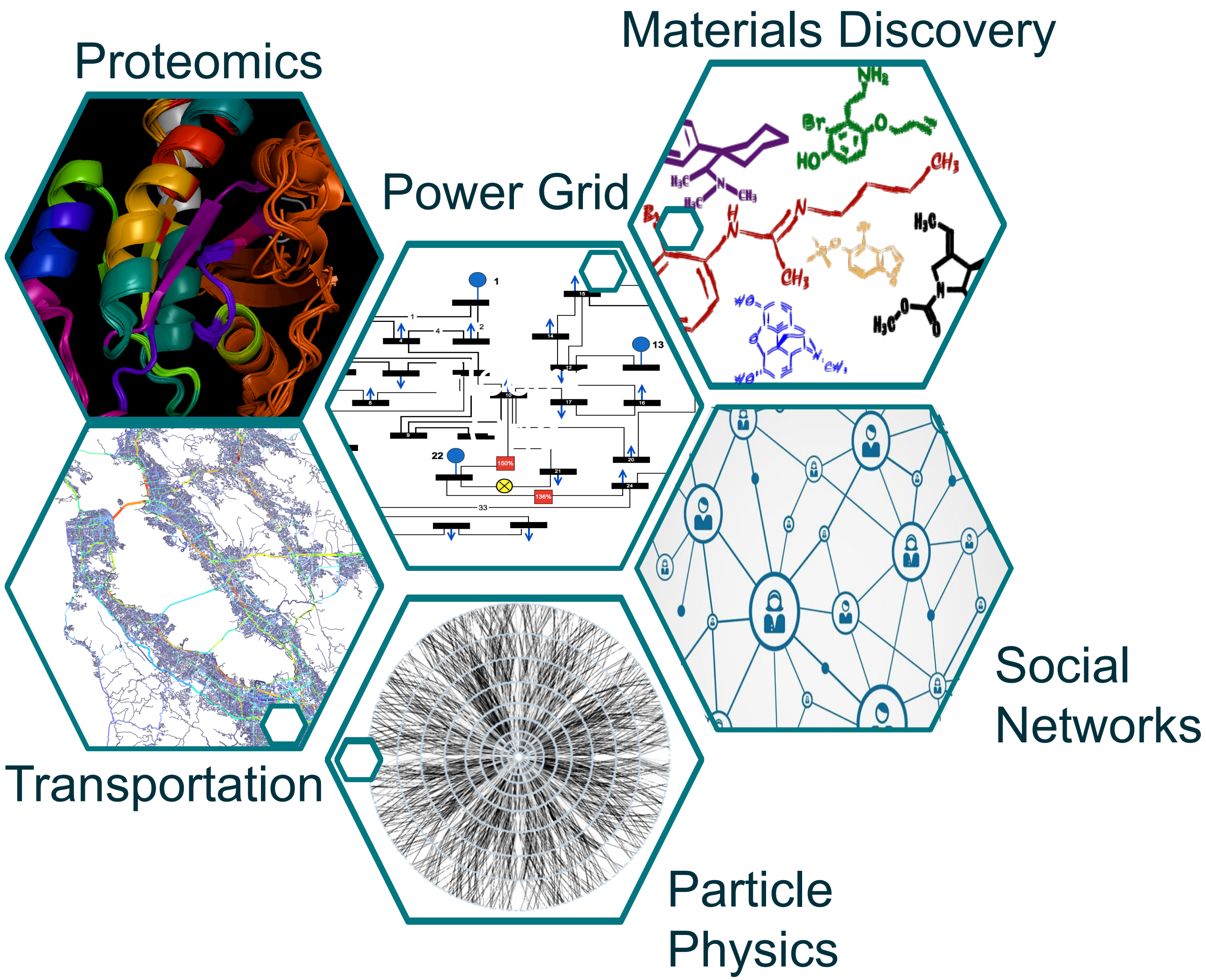
Reducing Communication in Graph Neural Network Training

Berkeley
UNIVERSITY OF CALIFORNIA

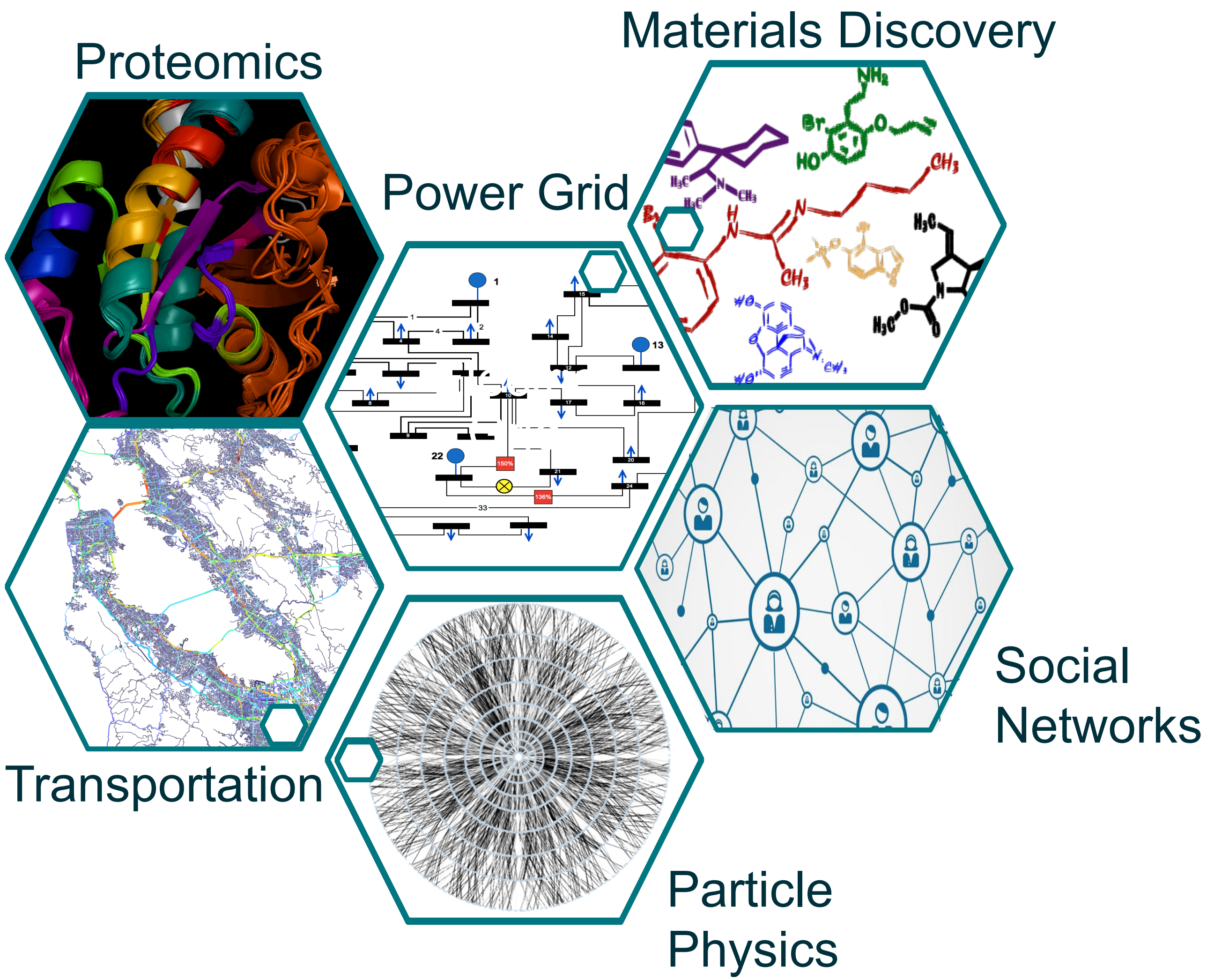
Alok Tripathy, Katherine Yelick, Aydın Buluç
University of California, Berkeley
Lawrence Berkeley National Laboratory

November 18th, 2020

Why focus on graphs?

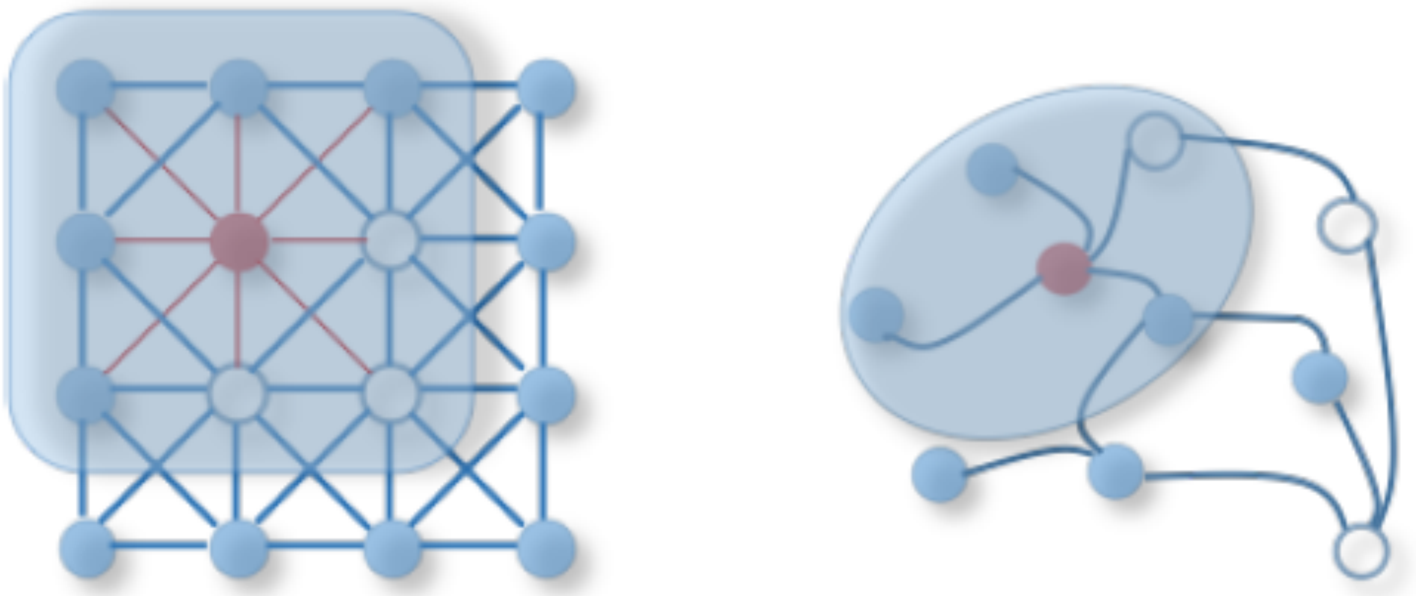


Learning problems on graphs



- Graph classification
- Edge classification
- **Node classification**

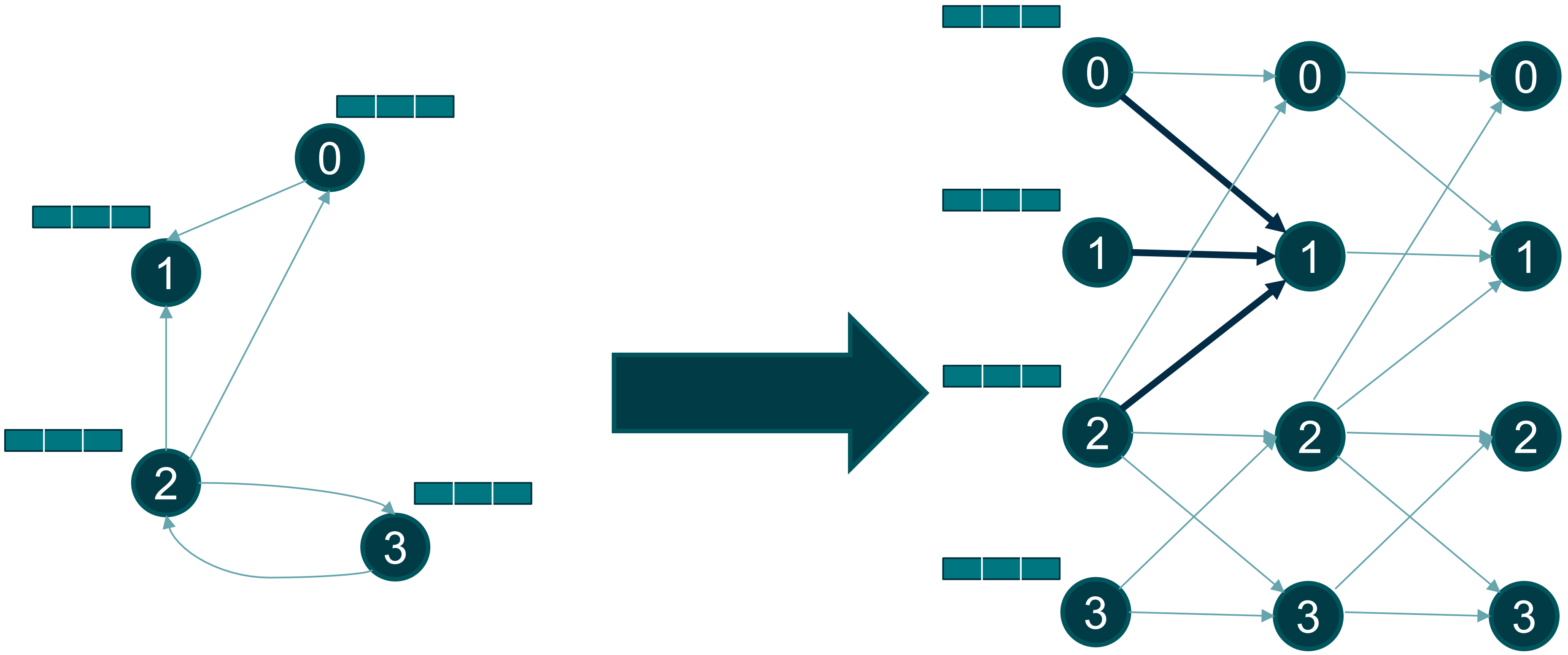
Why not use CNNs?



- Must generalize convolution

[1] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, “A comprehensive survey on graph neural networks,” arXiv preprint arXiv:1901.00596, 2019.

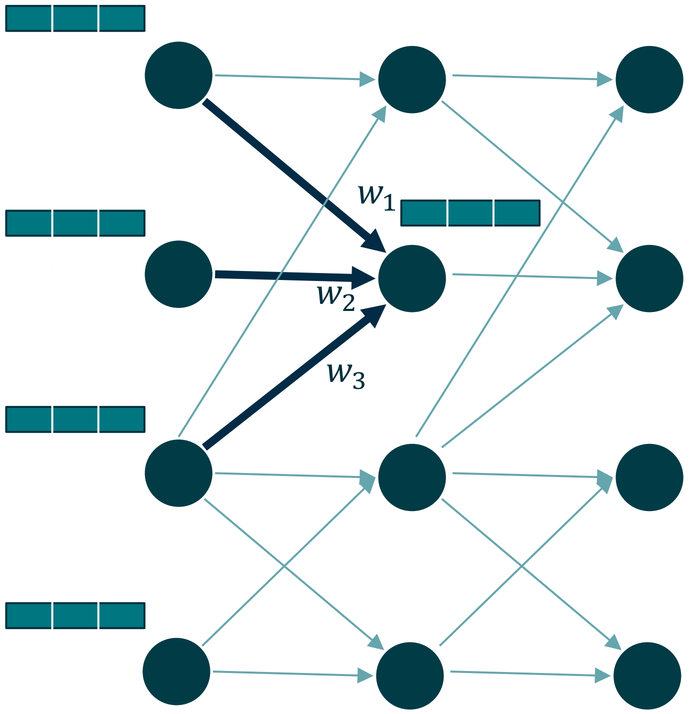
GNN basics



Input Graph

GNN of Input Graph

GNN training basics



1. Initialize feature vectors in layer 0
2. Sum neighbors' vectors for each node
3. Apply weight to vector sums

GNN issues

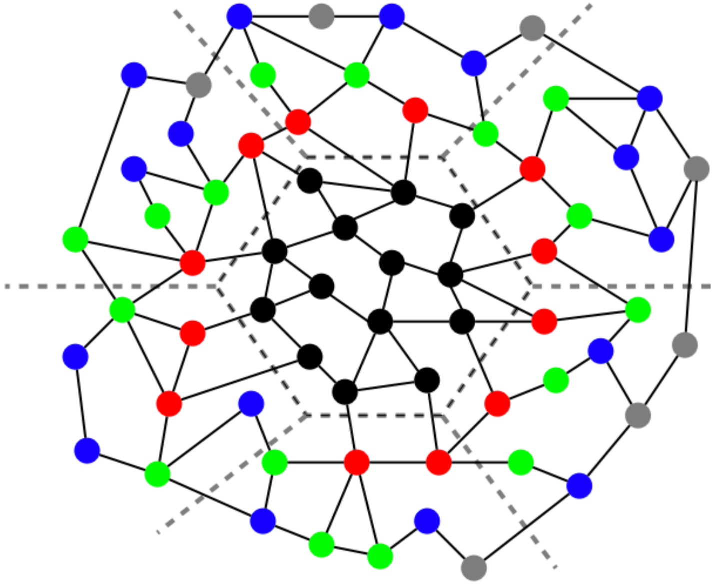
- GNN models are huge: $O(nfL)$
  - » n: number of vertices
  - » f: length of feature vector
  - » L: number of layers

- Need to distribute GNN training + inference

Why not use mini-batch SGD?

sample

No dependencies



Layered dependencies

- Layered dependencies → space issue persists
- Focus on full gradient descent

[2] Marghoob Mohiyuddin, Mark Hoemmen, James Demmel, and Katherine Yelick. Minimizing Communication in Sparse Matrix Solvers. In Proceedings of the 2009 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis(SC), 2009.

How do we distribute GNN training?

1. Formulate GNN training with sparse-dense matrix multiplication operations
   » Both forward and back propagation

2. Distribute with distributed sparse-dense matrix multiplication algorithms

– Focus on node classification, but methods are general

GNN Training with Sparse-Dense Matrix Multiplication

GNN training as sparse-dense matrix multiplication

Forward Propagation:

$$\mathbf{Z}^l \leftarrow \mathbf{A}^\mathsf{T}\mathbf{H}^{l-1}\mathbf{W}^l$$

$$\mathbf{H}^l \leftarrow \sigma(\mathbf{Z}^l)$$

Backward Propagation:

$$\mathbf{G}^{l-1} \leftarrow \mathbf{A}\mathbf{G}^l(\mathbf{W}^l)^\mathsf{T} \odot \sigma'(\mathbf{Z}^{l-1})$$

$$\mathbf{Y}^{l-1} \leftarrow (\mathbf{H}^{l-1})^\mathsf{T}\mathbf{A}\mathbf{G}^l$$

- $\mathbf{A}$ is stored in sparse format
- All other matrices dense

Symbols and Notations	
Symbol	Description
$\mathbf{A}$	Modified adjacency matrix of graph $(n \times n)$
$\mathbf{H}^l$	Embedding matrix in layer l $(n \times f)$
$\mathbf{W}^l$	Weight matrix in layer l $(f \times f)$
$\mathbf{Y}^l$	Matrix form of $\frac{\partial \mathcal{L}}{\partial W^l_{ij}}$ $(f \times f)$
$\mathbf{Z}^l$	Input matrix to activation function $(n \times f)$
$\mathbf{G}^l$	Matrix form of $\frac{\partial \mathcal{L}}{\partial Z^l_{ij}}$ $(n \times f)$
σ	Activation function
f	Length of feature vector per vertex
f_u	Feature vector for vertex u
L	Total layers in GNN
P	Total number of processes
α	Latency
β	Reciprocal bandwidth

GNN training as sparse-dense matrix multiplication

Forward Propagation:

$$\mathbf{Z}^l \leftarrow \mathbf{A}^{\mathsf{T}}\mathbf{H}^{l-1}\mathbf{W}^l$$ ← SpMM, DGEMM

$$\mathbf{H}^l \leftarrow \sigma(\mathbf{Z}^l)$$ ← In paper

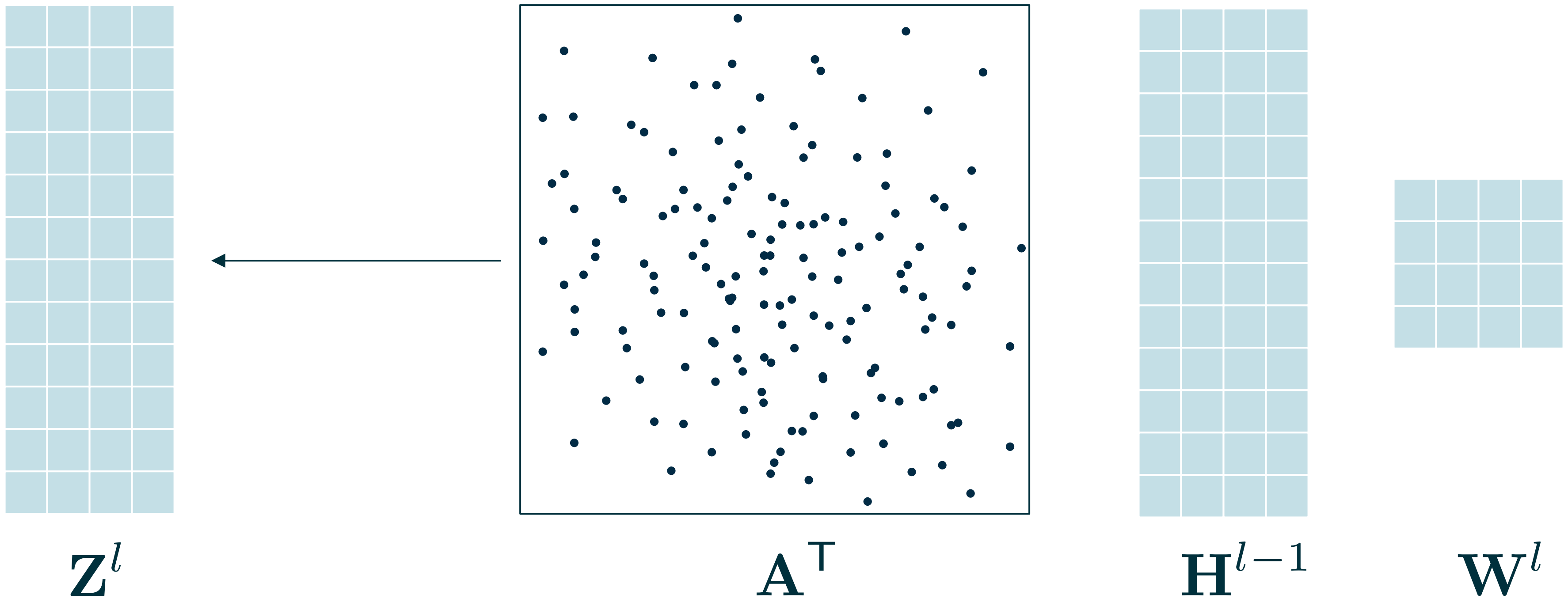
Backward Propagation:

$$\mathbf{G}^{l-1} \leftarrow \mathbf{A}\mathbf{G}^l(\mathbf{W}^l)^{\mathsf{T}} \odot \sigma'(\mathbf{Z}^{l-1})$$ ← SpMM, DGEMM

$$\mathbf{Y}^{l-1} \leftarrow (\mathbf{H}^{l-1})^{\mathsf{T}}\mathbf{A}\mathbf{G}^l$$ ← DGEMM

- **Entirely SpMM, DGEMM calls**

Bottleneck of GNN training

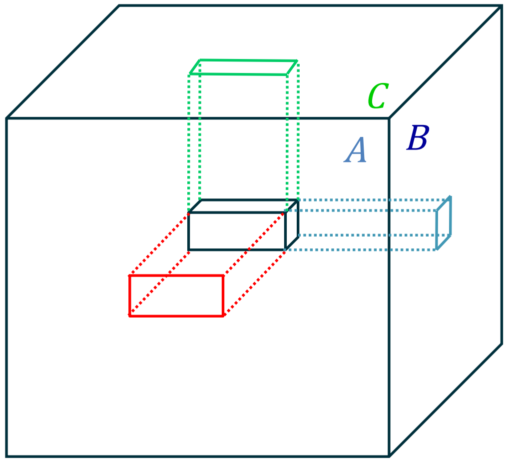


- SpMM >>> DGEMM

Distributed Matrix Multiplication Algorithms

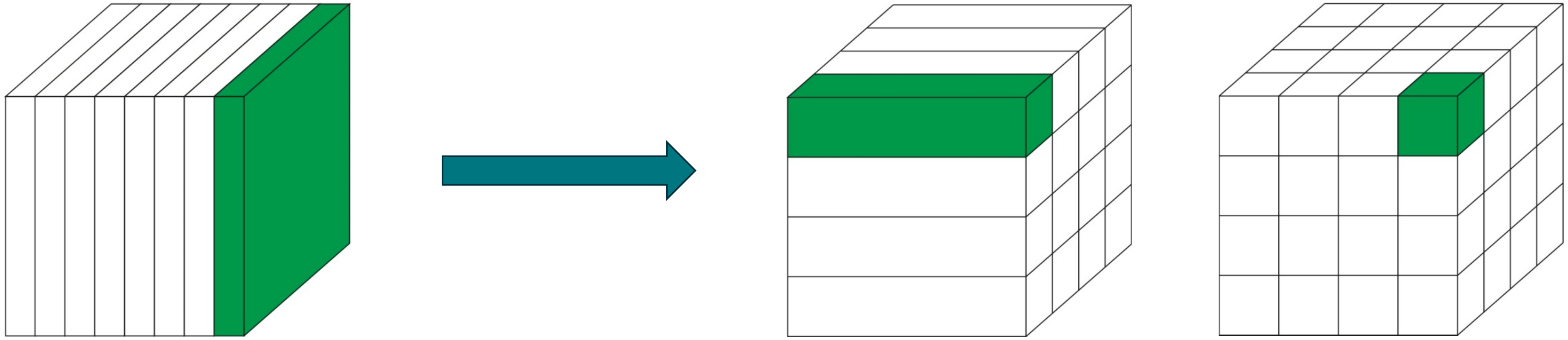
The computation cube of matrix-matrix multiplication

Matrix multiplication: $\forall (i,j) \in n \times n, \quad C(i,j) = \Sigma_k A(i,k)B(k,j),$



The *computation (discrete) cube*:

- A face for each (input/output) matrix
- A grid point for each multiplication



1D algorithms 1.5D algorithms 2D algorithms 3D algorithms

GNN training communication analysis

Communication Analyses			
Algorithm	Latency	Bandwidth	Memory
1D	$\lg P + 2P$	$2nf + f^2$	$\frac{nnz(\mathbf{A})}{P} + \frac{nf}{P}$
1.5D	$2\frac{P}{c^2}\lg\frac{P}{c^2}$	$\frac{2nf}{c} + \frac{2nfc}{P}$	$\frac{nnz(\mathbf{A})c}{P} + \frac{nfc}{P}$
2D	$5\sqrt{P} + 3\lg P$	$\frac{8nf}{\sqrt{P}} + \frac{2nnz(\mathbf{A})}{\sqrt{P}}$	$\frac{nnz(\mathbf{A})}{P} + \frac{nf}{P}$
3D	$4P^{1/3}$	$\frac{2nnz(\mathbf{A})}{P^{2/3}} + \frac{12nf}{P^{2/3}}$	$\frac{nnz(\mathbf{A})}{P} + \frac{nf}{P}$

Symbols and Notations	
Symbol	Description
$\mathbf{A}$	Modified adjacency matrix of graph $(n \times n)$
$\mathbf{H}^l$	Embedding matrix in layer l $(n \times f)$
$\mathbf{W}^l$	Weight matrix in layer l $(f \times f)$
$\mathbf{Y}^l$	Matrix form of $\frac{\partial \mathcal{L}}{\partial W^l_{ij}}$ $(f \times f)$
$\mathbf{Z}^l$	Input matrix to activation function $(n \times f)$
$\mathbf{G}^l$	Matrix form of $\frac{\partial \mathcal{L}}{\partial Z^l_{ij}}$ $(n \times f)$
σ	Activation function
f	Length of feature vector per vertex
f_u	Feature vector for vertex u
L	Total layers in GNN
P	Total number of processes
α	Latency
β	Reciprocal bandwidth

- $nnz(\mathbf{A})$ is the number of edges
- c is the replication factor for 1.5D (c =1 is 1D)

BERKELEY LAB
Bringing Science Solutions to the World

U.S. DEPARTMENT OF ENERGY

GNN Training with Sparse-Dense Matrix Multiplication Results

Implementation Details

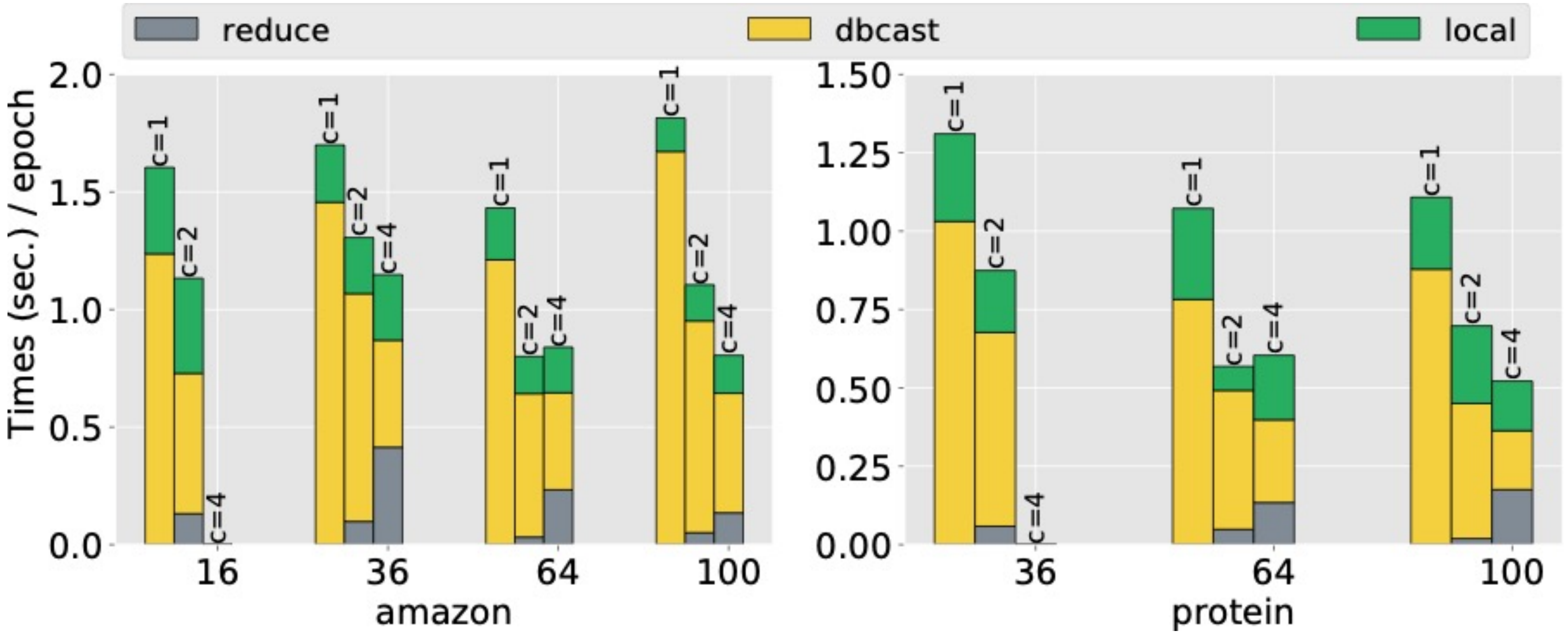
- PyTorch 1.3 with NCCL 2.0 backend
  - » Kipf-Welling model (3-layers, 16 hidden activations)

- System:
  - » Summit at OLCF
  - » 6 NVIDIA V100s per node
  - » NVLINK 2.0, EDR Infiniband

- Datasets:

Name	Vertices	Edges	Features	Labels
Amazon	14M	231M	300	24
Reddit	233K	114M	602	41
Protein	8M	2B	128	256

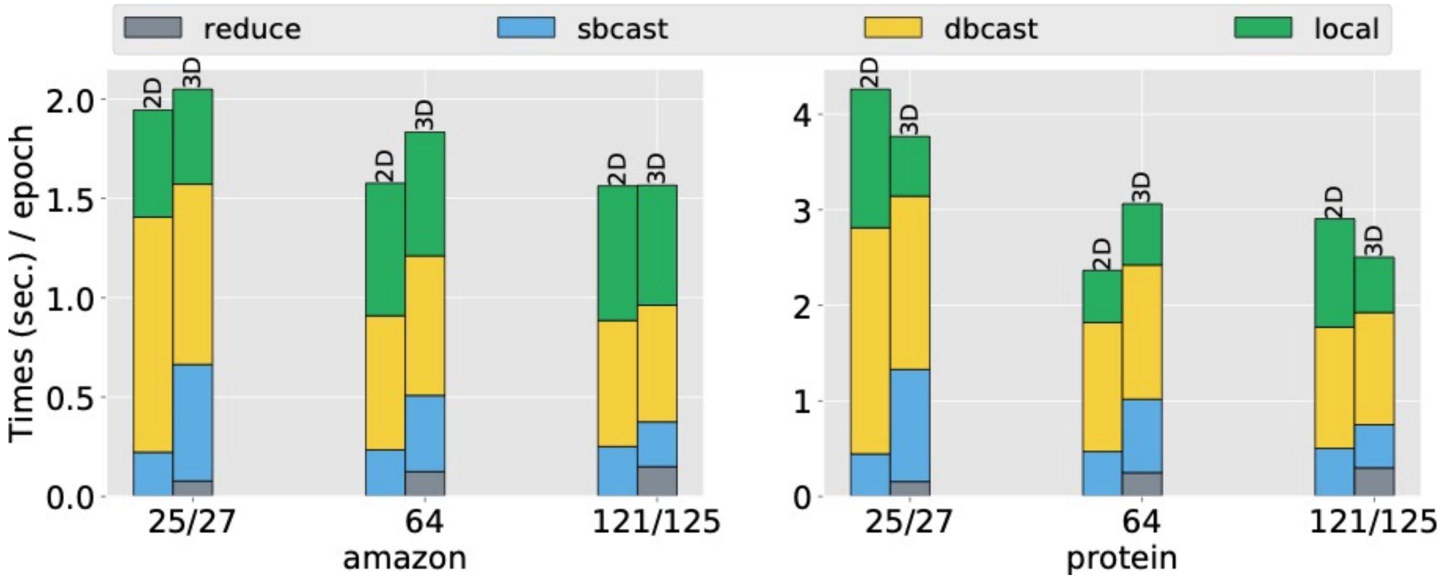
GNN Training with 1.5D Matrix Multiplication



- Scales with both P and c with 1 GPU/node
  - » Summit topology
  - » Full 6GPU/node results in paper
- Expect to scale with all GPUs / node with future architectures
  - » e.g. Perlmutter

BERKELEY LAB

GNN Training with 2D/3D Matrix Multiplication



- Other algorithms evaluated in practice (with 6GPUs/node)
- Communication scales with P, consistent with analysis
- Computation scales less well → explained in paper

BERKELEY LAB

Conclusions

- Graphs are everywhere
  - » Lots of deep learning problems on graphs

- Can solve DL on graphs with GNNs
  - » But must distribute training

- Our work
  - » Can formulate GNN training as sparse-dense matrix multiplications
  - » Distribute GNN training with distributed SpMM
  - » Code: https://github.com/PASSIONLab/CAGNET
  - » Paper: https://arxiv.org/pdf/2005.03300.pdf

BERKELEY LAB