



# Challenges in Accelerating Graph Machine Learning

Viktor Prasanna

University of Southern California

[prasanna@usc.edu](mailto:prasanna@usc.edu)

[fpga.usc.edu](http://fpga.usc.edu)

[dslab.usc.edu](http://dslab.usc.edu)

[sites.usc.edu/prasanna](http://sites.usc.edu/prasanna)

March 8, 2024



# Graph Analytics

- Leverage graph structures to represent, understand, and analyze relationships that exist between entities (people, device)
- Example kernels
  - Classic: Centrality, Distances, Connectivity, ..
  - Recent: GNN, GCN, Temporal GNNs, ....
- Streaming Graph Analytics
  - A stream of (graph) data as input
  - Very large data and limited amount of (local) memory
  - Online and Approximate Solutions
- Graph databases
  - NoSQL
  - NewSQL
  - Neo4J, Tigergraph

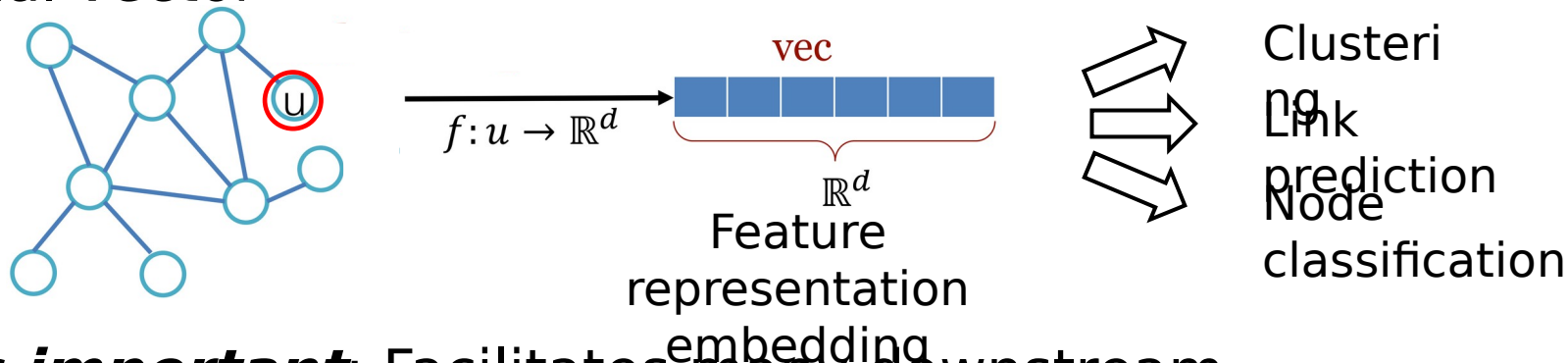


- [1] Graph exploitation symposium <https://events.ll.mit.edu/graphex/>  
[2] BigGraphs workshop <http://www.biggraphs.org/>



# Graph Neural Networks (GNNs)

**What it does:** Embed node features and graph structure into a low-dimensional vector



**Why it is important:** Facilitates many downstream applications

- *Tasks:* node classification, link prediction, clustering ...
- *Applications:* social recommendation, traffic forecast, protein analysis

## **Applications of GCNs in commercial systems:**

- Pinterest: web-scale recommender system
- Alibaba: GCN-based recommendation system for e-commerce



## **Why Acceleration:**

Very large graphs, training can take hours to days  
Facilitate many real-time downstream applications



# GNNs - High Level Abstraction

## GNN Parameters

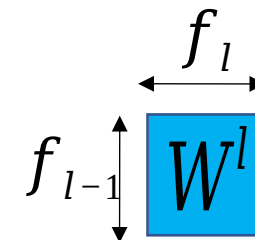
- : Number of Layers
- : feature size in layer
- : Adjacency matrix for feature aggregation
- : Feature Matrix for layer
- : input nodes in layer of GNN
- : number of nodes in layer
- : Target vertices for inference
- : Weight matrix for feature transformation
- element-wise activation function

## Given

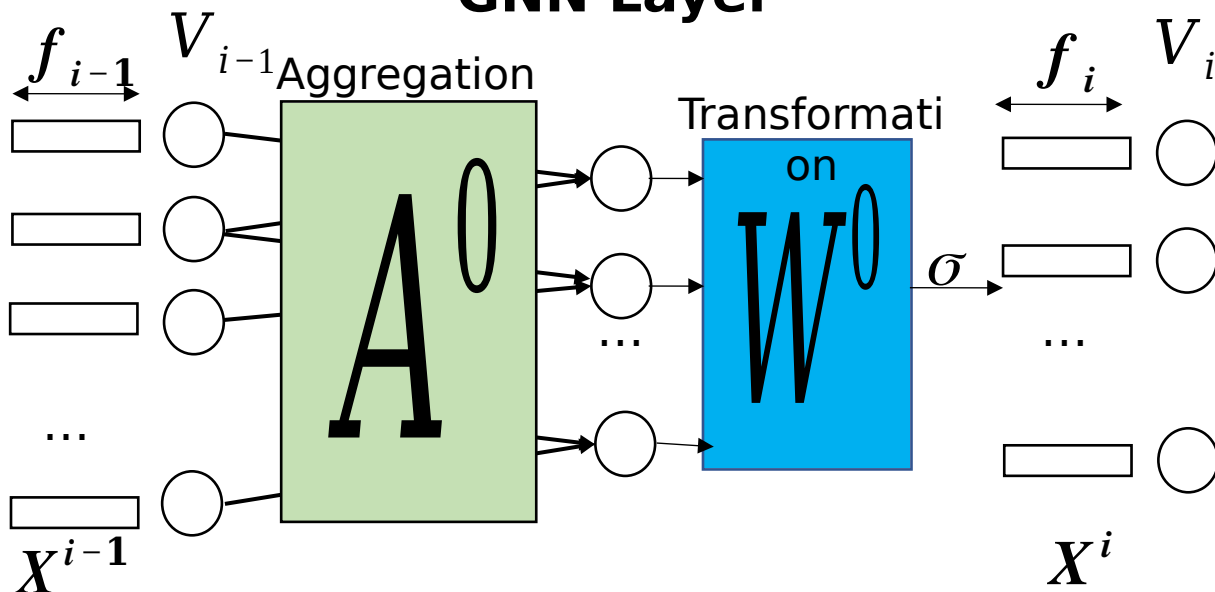
**Input:** Graph: (data)  
Target Vertices:

## GNN Model size

For 2 layer GNN with, 4 bytes per  
**Model size: 256 KB**



## GNN Layer



## Mini-Batch GNN

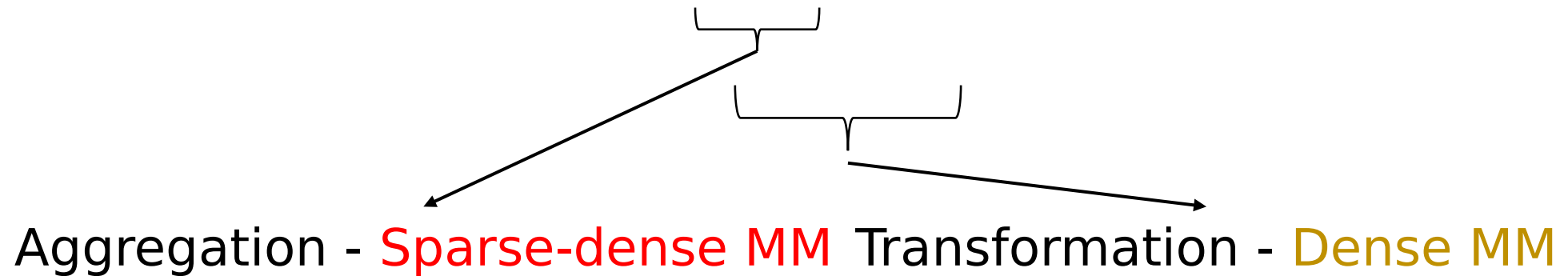
- Reduce memory usage on large graphs
- Layer sampling: sample supporting neighbors layer-by-layer
- Subgraph sampling: sample subgraphs and build complete GNN on the sampled subgraphs

Multiple layers are stacked to form a complete GNN model.



# GNN Computations

## Forward Pass Layer Propagation



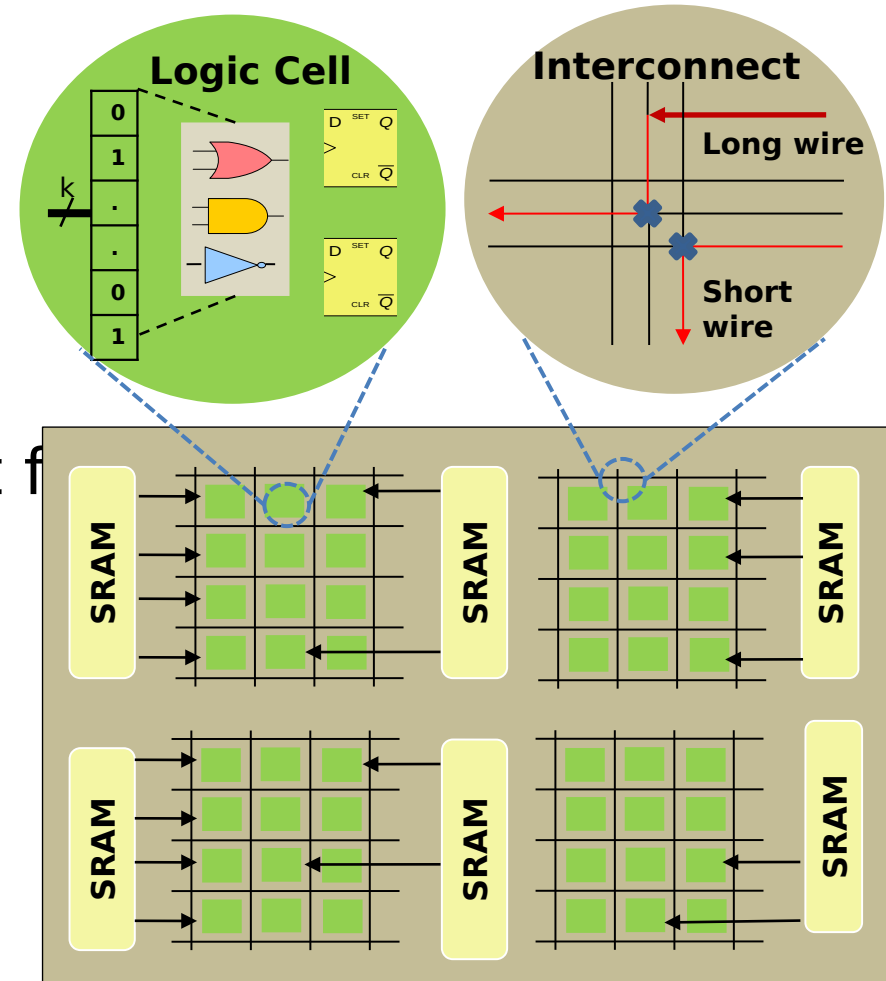
### Notes

- : sparse matrix. In sampling based approaches, is obtained by sampling which further reduces dimension or increases sparsity
- : dense matrix, typical size: (64 - 512) X (64 - 512) entries
- : typical size: 10s K X (64 - 512)



# Acceleration Technology: FPGA

- Field Programmable Gate Arrays
  - Configurable logic blocks (LUT)
  - Programmable interconnect
  - Programmable on-chip memory
- Logic block functionality
  - Simple logic (AND, OR, etc.)—4-input f
  - Shift register memory
- Memory hierarchy
  - LUT-based distributed RAM
  - SRAM (BRAM, M20K,....)
  - External memory through I/O
- Huge on-chip bandwidth (Tbps)



# FPGA, Multicore, GPU and Accelerators



Feature	FPGA (Virtex UltraScale+)	FPGA (Stratix 10 NX)	Multicore ( <b>Xeon MAX 9480</b> )	GPU (NVIDIA H100)	AI Accelerator ( <b>Cerebras WSE</b> )
<b>Parallelism</b>	Fine grained, pipelined	Fine grained, pipelined	Thread/Core level (Task)	SIMT/SMP (Data Parallel)	Core level and Data Parallel
<b>Operating frequency</b>	Up to 825 MHz	Up to 800 MHz	2.6 GHz	1.7 GHz	-
<b>On-chip memory bandwidth</b>	Configurable (8 Tbps)	~8 Tbps	L1: 794 Gbps, L2: 400 Gbps	4~6 Tbps	9.6 Pbps
<b>On-chip memory access latency</b>	1.2 ns	~ 1 ns	L1: 1.2 ns, L2: 3 ns, L3: 15 ns	L1: 1~20 ns	-
<b>On chip Memory</b>	56 MB Block RAM	94.5 Mb eSRAM	112.5 MB Cache	50 MB L2 Cache	18 G SRAM
<b>DDR channels - bandwidth</b>	Up to 32 (HBM) 460 GB/s	(HBM2)-512 GB/s	Up to 8 (DDR5)- 307 GB/s	2-7.8 TB/s	N/A
<b>Power</b>	100 W		350 W	700 W	15 KW
<b>Parallelism</b>	2-12K DSPs	3960 AI Tensor blocks	56 cores 112 Threads	14592 CUDA Cores	400,000 SLAC Cores
<b>Technology</b>	14 nm	14 nm FinFET	10 nm	4 nm	16 nm



# FPGAs for Graph Analytics

- Challenges
  - Large data volume
  - Irregular memory accesses
  - Limited data reuse
  - High computation complexity (machine learning on graphs)
- Opportunities
  - Emerging memory technologies
    - Large capacity
    - Low access latency
    - High bandwidth
  - New architectures supporting coherent shared-memory
  - Fine-grained acceleration
  - Customized memory controller to handle irregular memory accesses
  - Algorithmic innovation to reduce computation complexity





# References

- GraphSAINT
  - Zeng; Zhou; Srivastava; Kannan; Prasanna; “GraphSAINT: Graph sampling based inductive learning method.” ICLR 2020.
- ShaDOW-GNN
  - Zeng; Zhang; Xia; Srivastava; Malevich; Kannan; Prasanna; Jin; Chen. “Deep Graph Neural Networks with Shallow Subgraph Samplers.” NeurIPS 2021.
- HP-GNN
  - Lin; Zhang; Prasanna. "HP-GNN: Generating High Throughput GNN Training Implementation on CPU-FPGA Heterogeneous Platform." ACM FPGA 2022.
- HyScale-GNN
  - Lin; Prasanna. “HyScale-GNN: A Scalable Hybrid GNN Training System on Single-Node Heterogeneous Architecture.” IEEE IPDPS 2023.
- GraphAGILE:
  - Zhang; Zeng; Prasanna. “GraphAGILE: An FPGA-based Overlay Accelerator for Low-latency GNN Inference.” IEEE Transactions on Parallel and Distributed Systems (TPDS).



# References (Continued)

- Dynaspase
  - Zhang; Prasanna. “Dynaspase: Accelerating GNN Inference through Dynamic Sparsity Exploitation.” IEEE IPDPS 2023.
- SeDyT
  - Zhou; James; Kannan; Prasanna; “SeDyT: A General Framework for Multi-Step Event Forecasting via Sequence Modeling on Dynamic Entity Embeddings.” CIKM 2021
- HTNet
  - Zhou; Kannan, Swami, Prasanna; “HTNet: Dynamic WLAN Performance Prediction using Heterogenous Temporal GNN” INFOCOM 2023
- DistTGL
  - Zhou; Zheng; Song; Karypis; Prasanna; “DistTGL: Distributed Memory-Based Temporal Graph Neural Network Training” SC 2023



# GNN Acceleration?

- GNN applications consist of GNN training and GNN inference
  - GNN training: train GNN model on large-scale graph dataset
  - GNN inference: use trained GNN model for downstream tasks
- **GNN training:**
  - Requirement: need fast (high throughput), accurate and scalable GNN training system that performs on large-scale graph datasets
  - Read-world graphs are large
    - OGB graph: 0.25 billion nodes and 1.7 billion edges (167 GB)
    - AliGraph (Taobao): 0.48 billion nodes and 0.9 billion edges
  - Can take hours to days to train a GNN model using OGB graph on a GPU platform
- **GNN inference:**
  - Requirement: need accurate, low-latency and high-throughput GNN inference system for GNN application, such as personalized recommendation
  - Accuracy and latency is important for quality of service (QoS):
    - Latency in Facebook recommendation system should be within 10-100 ms
  - Samples  $>$  nodes to inference a single target node

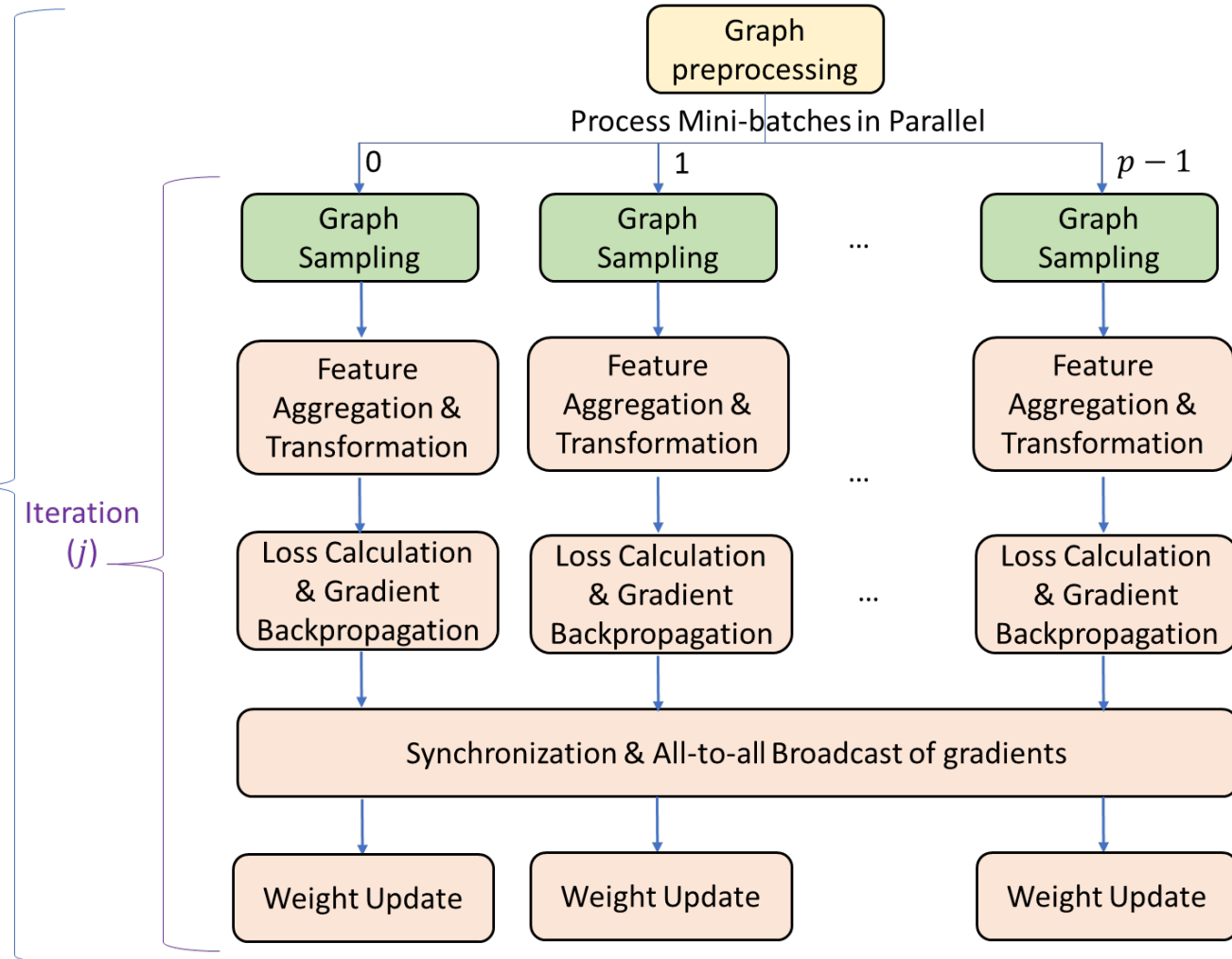
# Workflow Abstraction for GNN training (Parallel)



- Let  $p$  be the number of parallel processors
- Each process stores a local copy of weights
- number of epochs,  $n$ : size of each mini-batch
- Total number of minibatches:

```

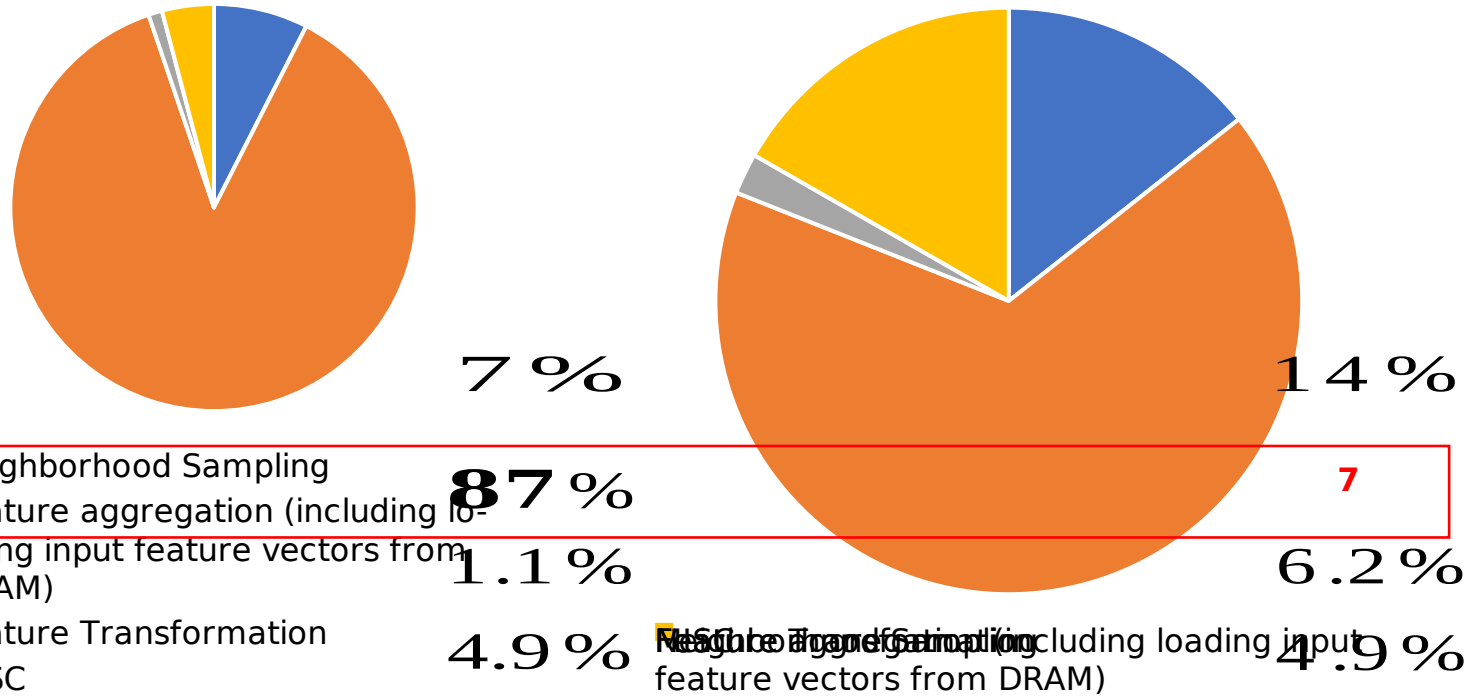
Repeat epochs until convergence
  Graph preprocessing ( ex. to build )
  All processes to in Parallel Do
    Process minibatches in the range
    For to do
      //Process  $t$ h minibatch
      Construct a GNN using graph
      sampling*
      Load Features*
      GNN Forward Propagation*
      Calculate Loss*
      GNN Backward Propagation*
      Synchronize_all_processes() □ ??
      All_to_all_Broadcast of Gradients
      □ ??
      Update Local Copy of Weights
    EndFor
  End Parallel Do
End repeat
  
```





# GNN Computation Profile

### Execution Time for Each Iteration



• **Software:** Pytorch Geometric

### Platform Specification

- Processor: Intel i9-9900K, 8 Cores with 16 Threads, 4.6 GHz
- DRAM: 80 GB DDR4 with 4 channels, 77 GB/s

### Model parameters

- Number of layers: 2
- Sample size: = 25 for two layers
- Minibatch size

**Reddit:** : 0.23 M, : 11 M

**Neighborhood size:** 115,000

**Per Iteration Time:** 2.37 s

**Total Time:** 1064 s

**Amazon:** : 1.6 M, : 0.13 B

**Neighborhood size:** 72,000

**Per Iteration Time:** 0.197 s

**Total Time:** 615 s

**Bottleneck: Gathering Feature Vectors (DRAM accesses) + Aggregation (sparse MM)**

# GNN vs DNN training and inference



	GNN	DNN (e.g. CNNs)
Model	<p><b>Small</b></p> <p><b>Number of parameters:</b> 10s of thousands</p> <p><b>Size:</b> 100s of KBs</p>	<p><b>Large</b></p> <p><b>Number of parameters:</b> 10s-100s of millions</p> <p><b>Size:</b> 10s-100s of MBs</p>
Data	<p>Size of sample for an inference <b>large</b> compared to model</p> <p>For example: In GraphSAGE, ~16 MB data sample needs to be fetched for one inference on a model of size ~64 KB</p>	<p>Size of sample for an inference <b>small</b> compared to model</p> <p>For example: In Inceptionv3, ~350 KB sized image needs to be fetched for one inference on a model of size 92 MB</p>
Computation / Communication	<ul style="list-style-type: none"><li>• Sparse computations on unstructured data</li><li>• Computations on small matrices (compared to CNNs)</li><li>• For parallel implementations, model synchronization time negligible</li></ul>	<ul style="list-style-type: none"><li>• Dense computations on structured data (unless explicit sparsification techniques used)</li><li>• Computations on large matrices (using im2col, kn2row methods)</li><li>• For parallel implementations, model synchronization time is significant</li></ul>



# Acceleration Challenges

- **Large data size:** Real-world graphs can be very large. Does not fit on the FPGA/Processor/GPU on-chip memory.
- **Poor data reuse and random memory access:** Real-world graphs are unstructured. Feature aggregation is a graph traversal process that leads to poor data reuse and random memory accesses.
- **Load imbalance:** Real-world graphs have unbalanced degree distribution. There is workload imbalance between the vertices in Feature aggregation, leading to imbalance across mini-batches.
- **Heterogenous kernels:** Feature aggregation is communication-intensive while the Feature transformation is computation-intensive. Pipelining these two kernels can lead to stalled pipeline execution.
- **Variability of input graphs and GNN models:** Graphs have various size and sparsity and various GNN models have various parameters.
- **Full graph, Sampling based, Training, Inference?**

# HP-GNN System Architecture



## Parallel Computation Kernels

- Multiple kernels distributed to multiple dies
- Multiple dies and multiple DDRs are connected through an all-to-all interconnection

## Update Kernel

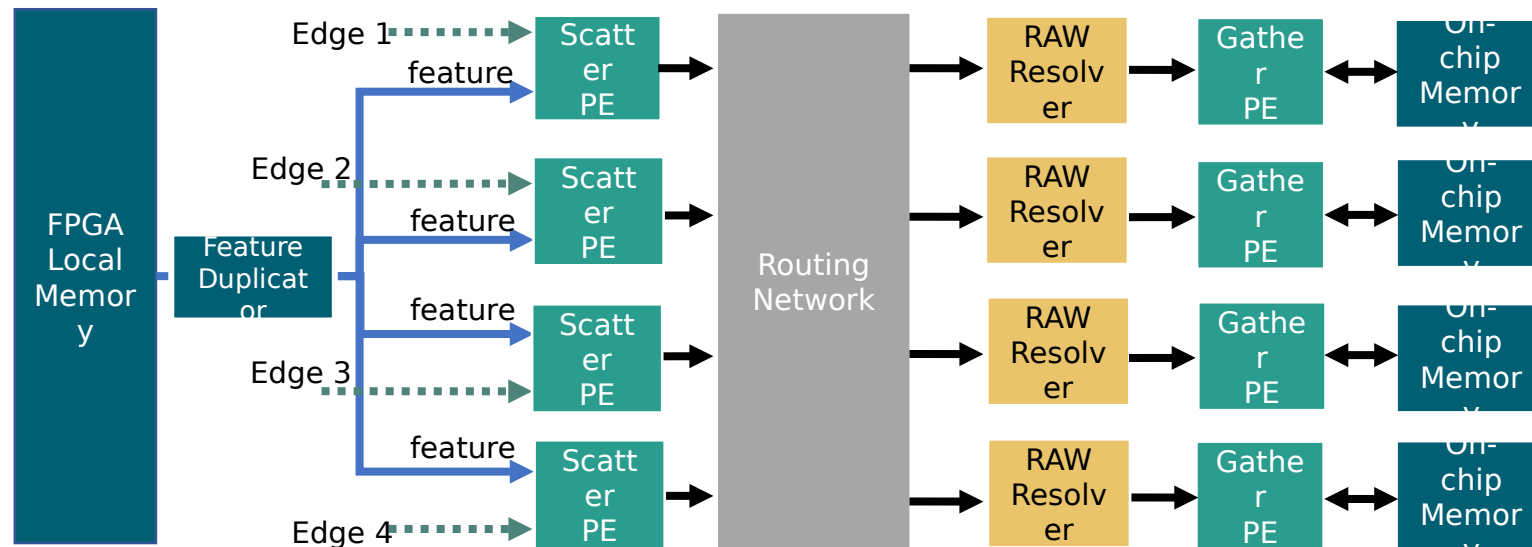
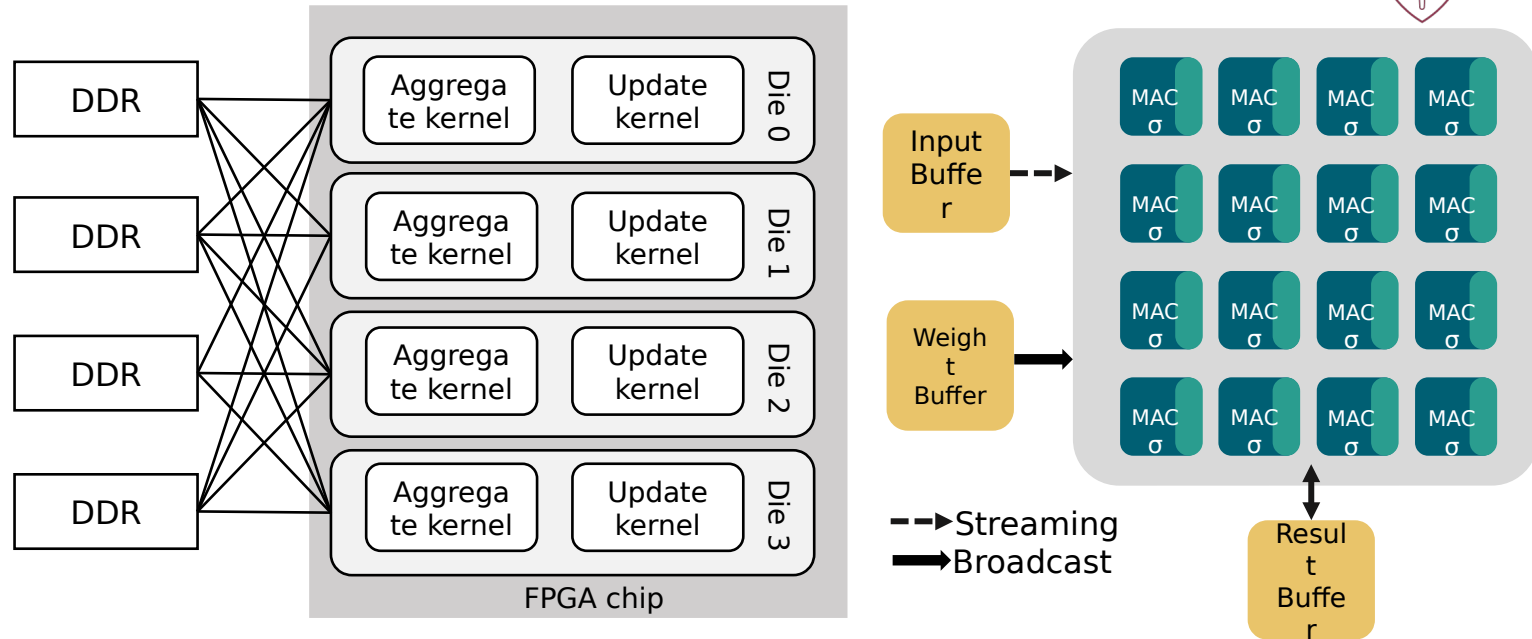
- systolic array
- performs multiplication-accumulation operations in each clock cycle.

## Aggregate Kernel

- Aggregation by scatter-gather paradigm
- Feature Duplicator
  - Duplicate feature to all Scatter PEs to exploit data reuse
- Scatter and Gather PEs
  - User-defined scatter/gather function
  - pipelines, each pipeline aggregates features to the destination in every clock cycle.

Yi-Chen Lin, Bingyi Zhang, and Viktor Prasanna

## HP-GNN: Generating High Throughput GNN Training



— data comes from FPGA local memory    - - - data comes from host via PCIe





# Experimental Results

## • Results

- Comparison of throughput (NVTPS)
- 55.67x speedup over CPU-only
- 2.17x speedup over CPU+GPU
- 3.39x - 4.45x speedup over state-of-the-art accelerators

		GraphACT [27]‡	Rubik [5]	Our work
Platform	Device	Alveo U250	ASIC	Alveo U250
	Peak Perf.	0.6 TFLOPS	1 TFLOPS	0.6 TFLOPS
	Bandwidth	77 GB/s	432 GB/s	77 GB/s
	On-chip Mem.	54 MB	2 MB	54 MB
SS-SAGE (Throughput)	RD	546.8K (1×)	717.0K (1.31×)	2.43M (4.45×)
	YP	769.8K (1×)	N/A	2.78M (3.61×)

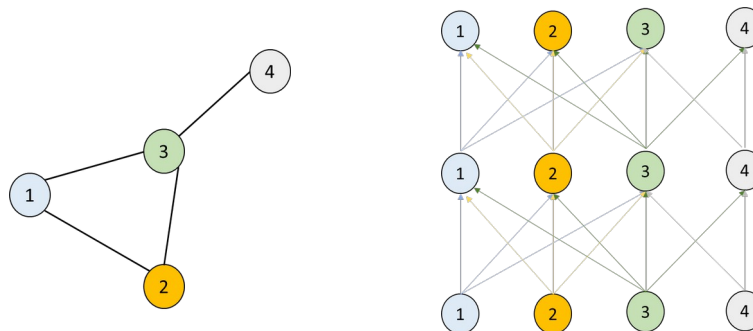
‡ Scaled from U200 to U250 using the number of DSPs.

	Data	CPU	CPU-GPU	CPU-FPGA
NS-GCN	FL	265.5K (1×)	2.69M (10.1×)	16.38M (61.7×)
	RD	85.65K (1×)	7.15M (83.5×)	18.50M (216×)
	YP	275.6K (1×)	9.36M (34.0×)	24.61M (89.2×)
	AM	480.6K (1×)	13.0M (29.0×)	29.26M (60.8×)
NS-SAGE	FL	225.2K (1×)	2.74M (12.2×)	11.84M (52.6×)
	RD	78.50K (1×)	6.90M (88.0×)	13.10M (166×)
	YP	266.0K (1×)	9.19M (34.5×)	18.12M (68.1×)
	AM	479.3K (1×)	13.57M (28.3×)	21.15M (44.1×)
SS-GCN	FL	215.2K (1×)	768.3K (3.59×)	2.81M (13.0×)
	RD	118.9K (1×)	536.4K (4.51×)	2.56M (21.5×)
	YP	159.1K (1×)	751.0K (4.71×)	3.08M (19.4×)
	AM	25.55K (1×)	OoM	1.47M (57.5×)
SS-SAGE	FL	179.9K (1×)	626.7K (3.48×)	2.71M (15.1×)
	RD	94.72K (1×)	505.2K (5.33×)	2.43M (25.6×)
	YP	126.7K (1×)	709.7K (5.60×)	2.78M (22.0×)
	AM	17.40K (1×)	OoM	1.45M (83.3×)
<b>Average</b>		<b>193.4K (1×)</b>	<b>4.96M (25.66×)</b>	<b>10.77M (55.67×)</b>



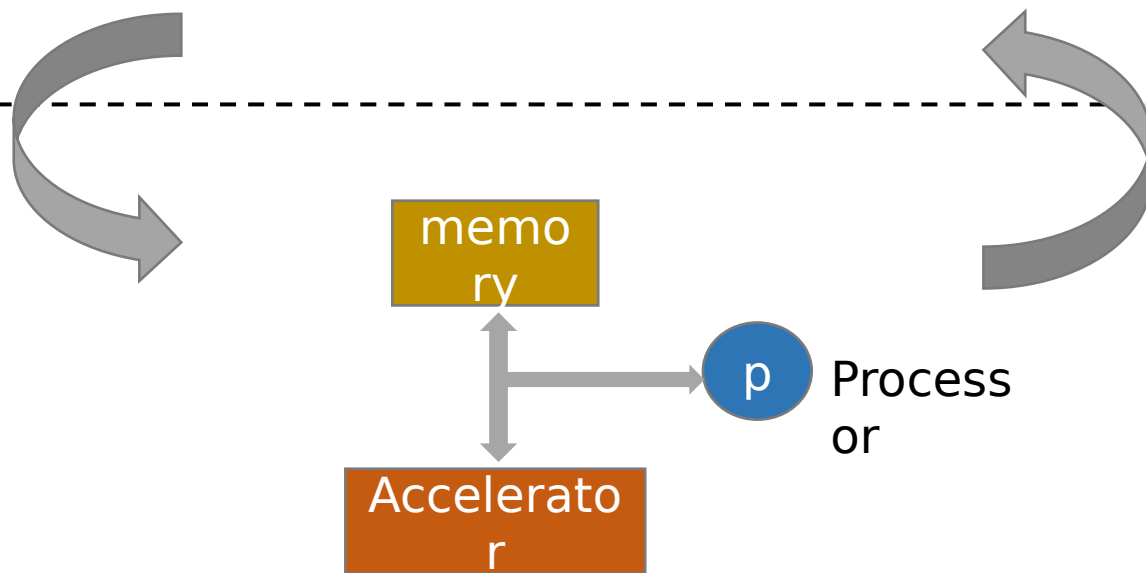
# GNN algorithm-architecture co-design

ML model architecture



- Accuracy
- Expressivity
- Graph sampling
- # of layers
- Layer Connectivity
- Aggregation function
- .....

Accelerator architecture



- On-chip memory
- Memory bandwidth
- Memory latency
- Data reuse
- Computation parallelism
- Scalability
- Task pipelining
- Computation complexity
- .....

Zeng; Zhang; Xia; Srivastava; Malevich; Kannan; Prasanna; Jin; Chen. "Deep Graph Neural Networks with Shallow Subgraph Samplers." NeuRIPS 2021.



# Thanks

fpga.usc.edu

dslab.usc.edu

prasanna@usc.edu

sites.usc.edu/prasanna



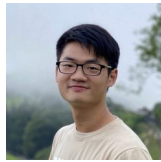
# Team



Rajgopal Kannan  
Research Faculty



Omer  
Akgul



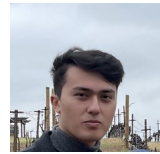
Jason  
Lin



Yuan  
Meng



Ta-Yang  
Wang



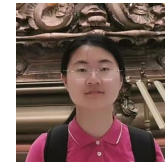
Samuel  
Wiggins



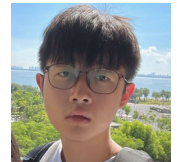
Sasindu  
Wijeratne



Sachini  
Wickramasinghe



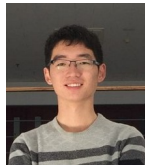
Yuhong  
Liu



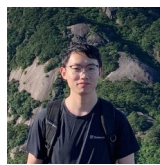
Gangda  
Deng



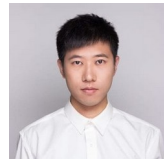
Yang  
Yang



Tian  
Ye



Bingyi  
Zhang



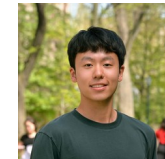
Pengmiao  
Zhang



Hongkuan  
Zhou



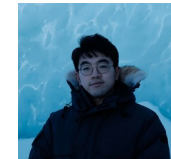
Nikunj  
Gupta



Zhihan  
Xu



Dhruv  
Parikh



Xu  
Wang



Yuxin  
Yang



USC University of  
Southern California