Automatic Generation of High-Performance Inference Kernels for Graph Neural Networks on Multi-Core Systems

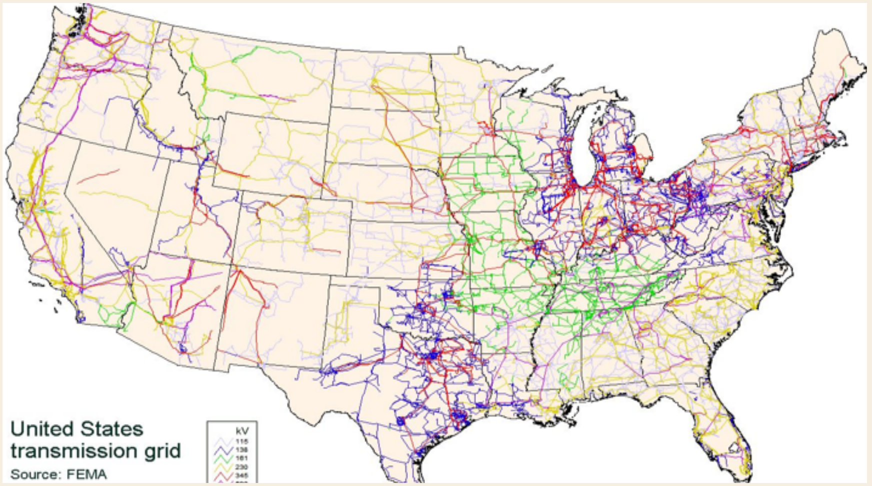
Qiang Fu H. Howie Huang

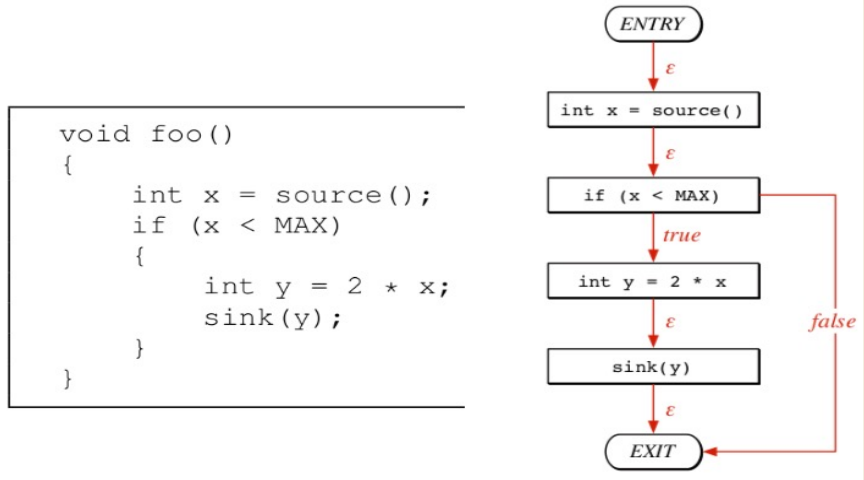
The George Washington University

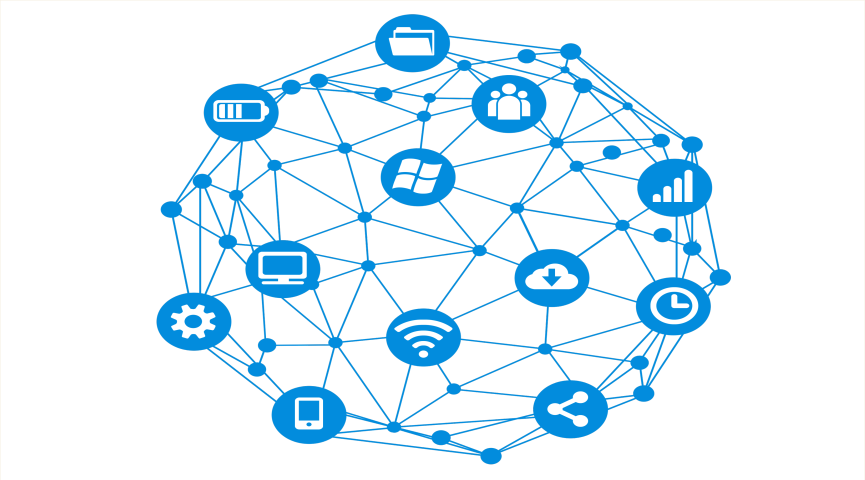
INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING

acm In-Cooperation
sighpc

Graph is Everywhere

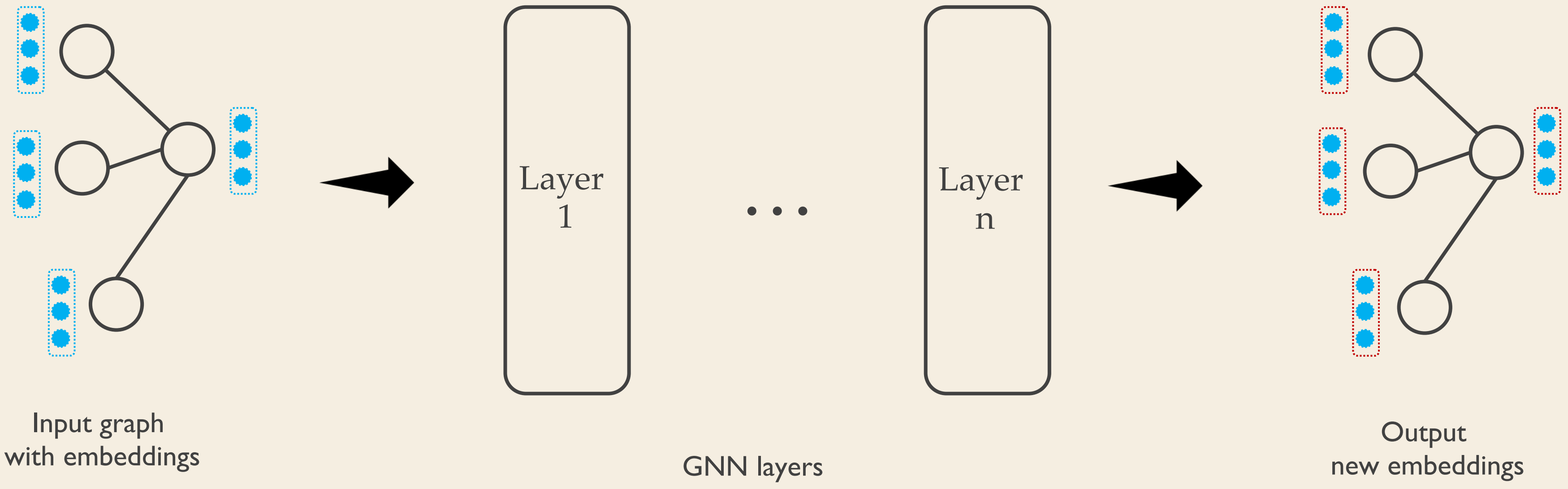






Graph Neural Networks (GNNs)

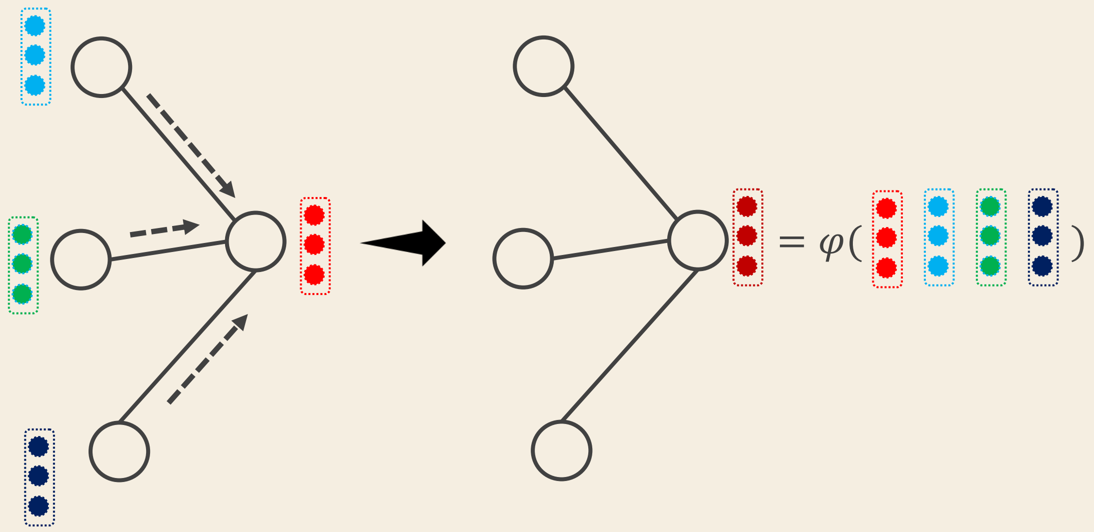
- GNNs take as input graph and initial embedding.
- Go through a series of convolution layers.
- Output new embeddings incorporating graph structure information.
- Successful application in social network mining, recommender system, molecule analysis etc.



Computation in Each Layer of GNNs

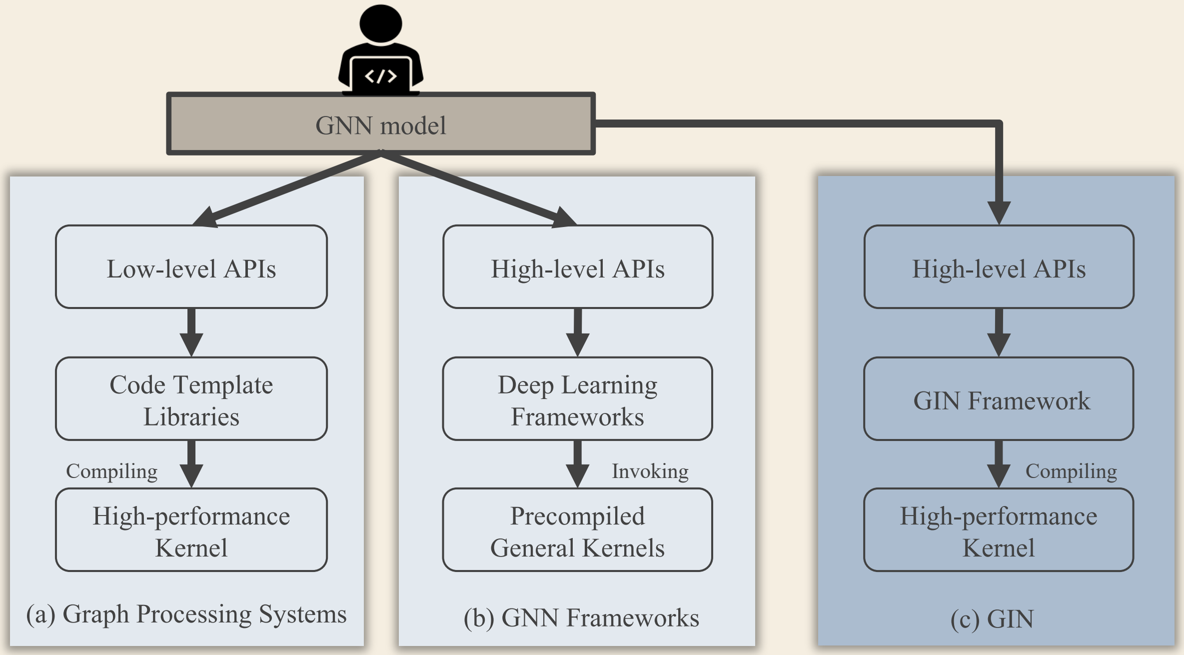
- Each vertex computes a new embedding by aggregating features (messages) from its neighbors.
- Example: Graph Convolution Network (GCN)

$$h_v^{(l+1)} = \sigma\left(\sum_{u \in N(v)} \frac{1}{\sqrt{d_v \cdot d_u}} h^{(l)} W^{(l)}\right)$$



Motivations

- Graph Processing Systems
  - Hard to programming GNNs
- GNN Frameworks
  - Suffer from poor performance
- We propose our GIN framework
  - A compiler-based approach generating high-performance kernels while offering easy-to-use APIs.



GIN Framework

- ACG programming model
- Dataflow Graph IR
- Code generator
- Optimizations

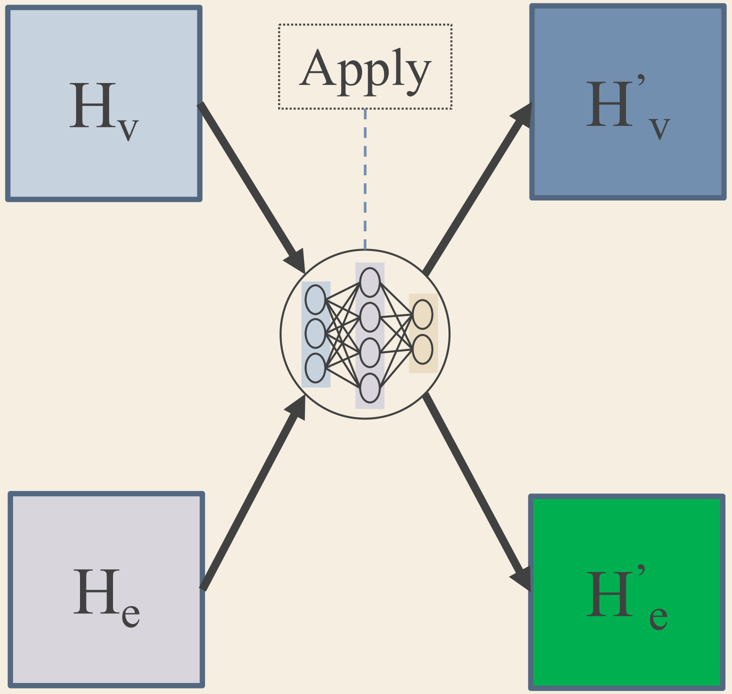
ACG Programming Model

- Apply:
  - Operations on feature matrices of vertices or edges before traversing the graph.
- In GCN:

$$h_v^{(l+1)} = \sigma\left(\sum_{u \in N(v)} \frac{1}{\sqrt{d_v \cdot d_u}} h^{(l)} W^{(l)}\right)$$

- Code:

```
Apply() {
    vdata.H = MatMul(vdata.H, vars.W);
}
```



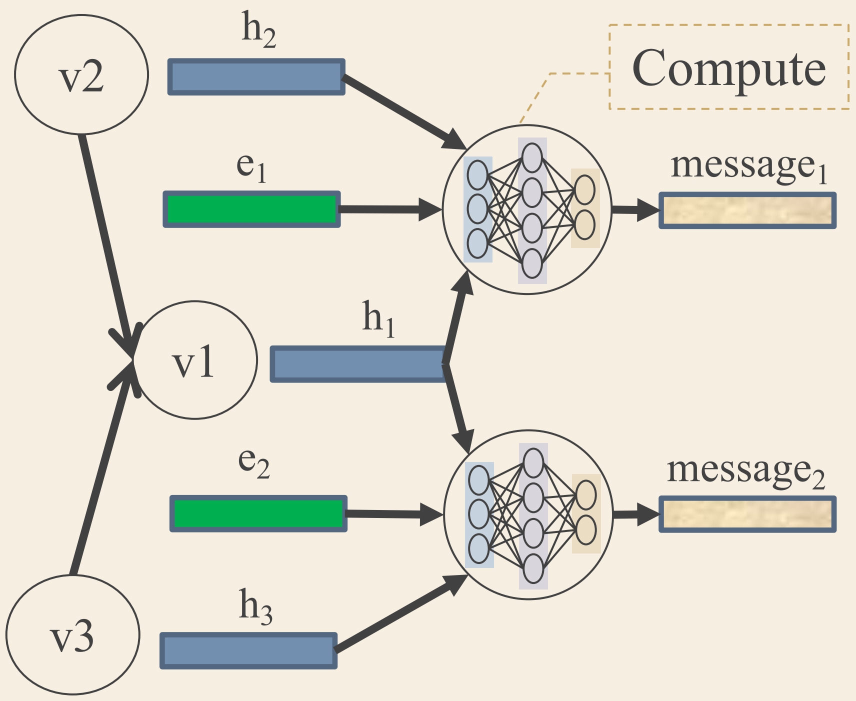
ACG Programming Model

- Compute
  - Operations defined on each edge to calculate the message.
- In GCN:

$$h_v^{(l+1)} = \sigma\left(\sum_{u \in N(v)} \frac{1}{\sqrt{d_v \cdot d_u}} h^{(l)} W^{(l)}\right)$$

- Code:

```
Compute(edge) {
    ret = edge.src.deg * edge.dst.deg;
    ret = Rsqrt(ret);
    ret = ret * edge.src.H;
    return ret;
}
```



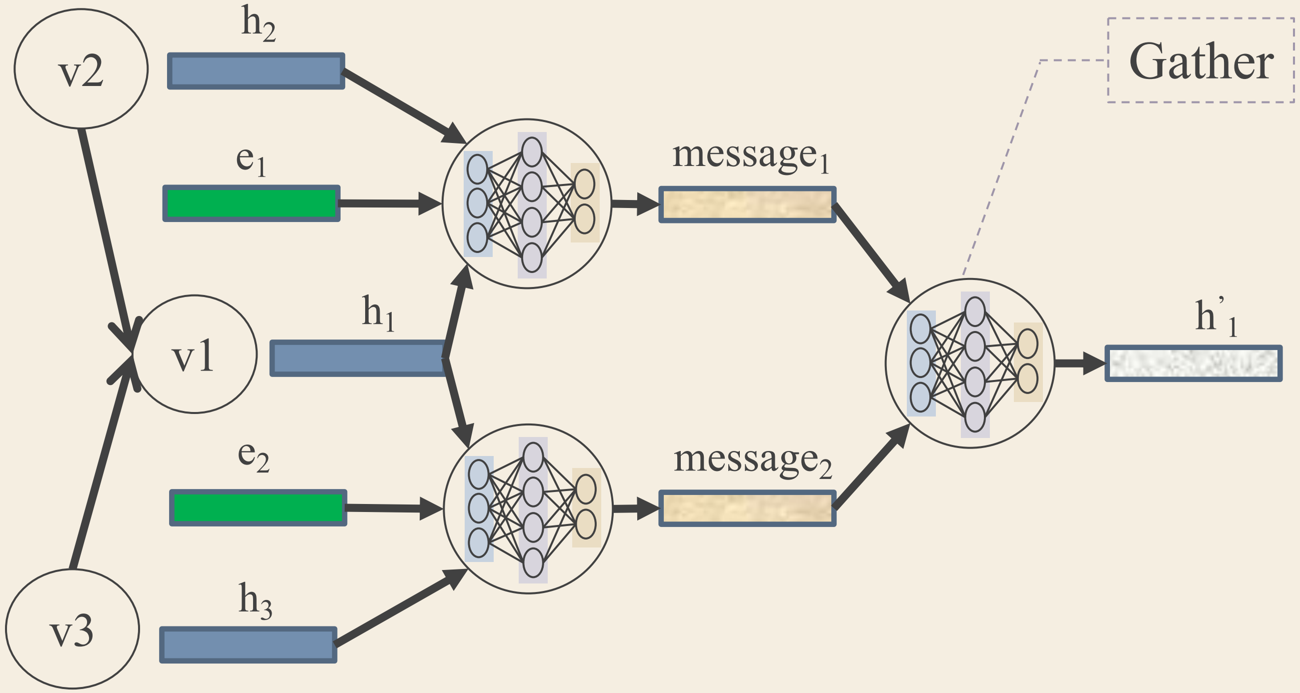
ACG Programming Model

- Gather
  - How to aggregate messages.
- In GCN:

$$h_v^{(l+1)} = \sigma\left(\sum_{u \in N(v)} \frac{1}{\sqrt{d_v \cdot d_u}} h^{(l)} W^{(l)}\right)$$

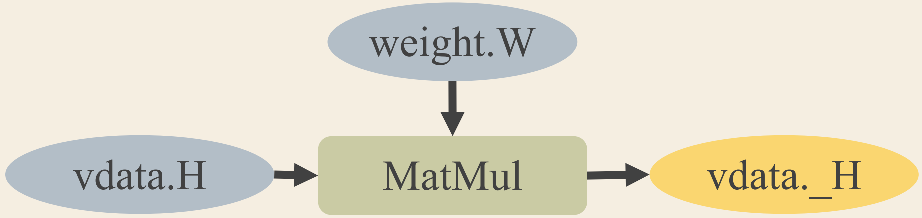
- Code:

```
Gather(messages[]) {
    ret = Sum(messages);
    return Relu(ret)
}
```



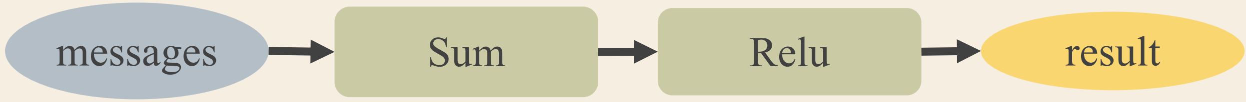
Dataflow Graph IR

```
Apply() {
    vdata.H = MatMul(vdata.H, vars.W);
}
```




```
Compute(edge) {
    ret = edge.src.deg * edge.dst.deg;
    ret = Rsqrt(ret);
    ret = ret * edge.src.H;
    return ret;
}
```


```
Gather(messages[]) {
    ret = Sum(messages);
    return Relu(ret)
}
```



Code Generator

- Start with a C++ code template
  - Graph traversal.
  - Blank code blocks corresponding to the three functions in the interface.
- Code generating
  - Iterate the nodes of the IR in topological order.
  - Emit C++ codes executing the computation represented by the IR.

```
Tensor Kernel_name (/* Input tensors list */) {

    /*  Code block 1 to initialize memory of intermediate
        and output tensors. */

    /*  Code block 2 to execute the computation in Appy
        function.  */

     parallel for each vertex v in graph {

         for each edge e in v's incoming edge list {
            /*  Code block 3 to execute the computation
             defined by Compute function, calculating
                the message on edge e. */
         }
         /*  Code block 4 to execute the computation defined
         in Gather function, merging all message from
            neighbors and updating features on vertex v. */
    }
    return output_tensor;  // Returning the result
}
```

Optimizations

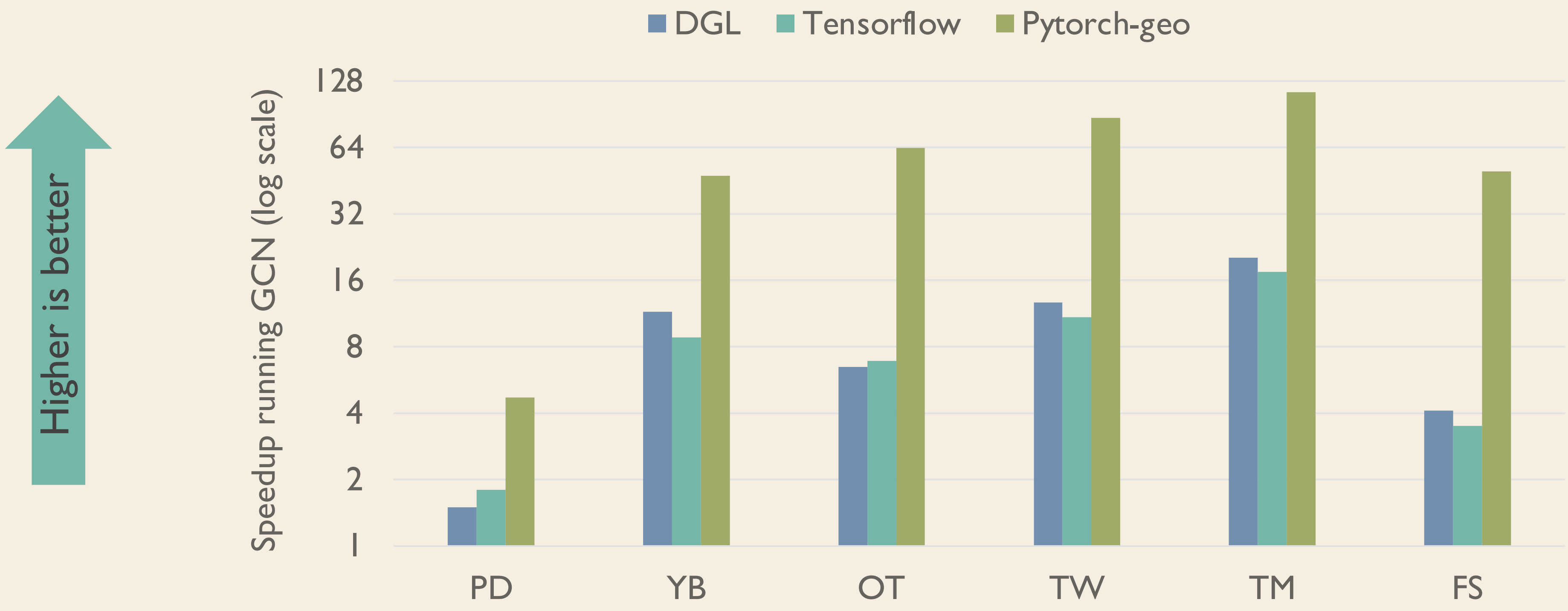
- Memory usage reduction
  - Delta-based updating on aggregating results
  - In-place operations such as activation function *relu*.
- Dynamic workload assignment
  - Each thread dynamically request workload of vertices from the task pool to avoid the workload imbalance.

Experiment Setups

- GNN models
  - CommNet, GCN, GGCN, GAT
- Datasets
- Baselines
  - DGL, Tensorflow, Pytorch-geometrics
- Computing environment
  - 2.6 GHz Intel Xeon(R) Gold 6126 processor (24 cores)
  - 1.5TB DRAM
  - Centos 7

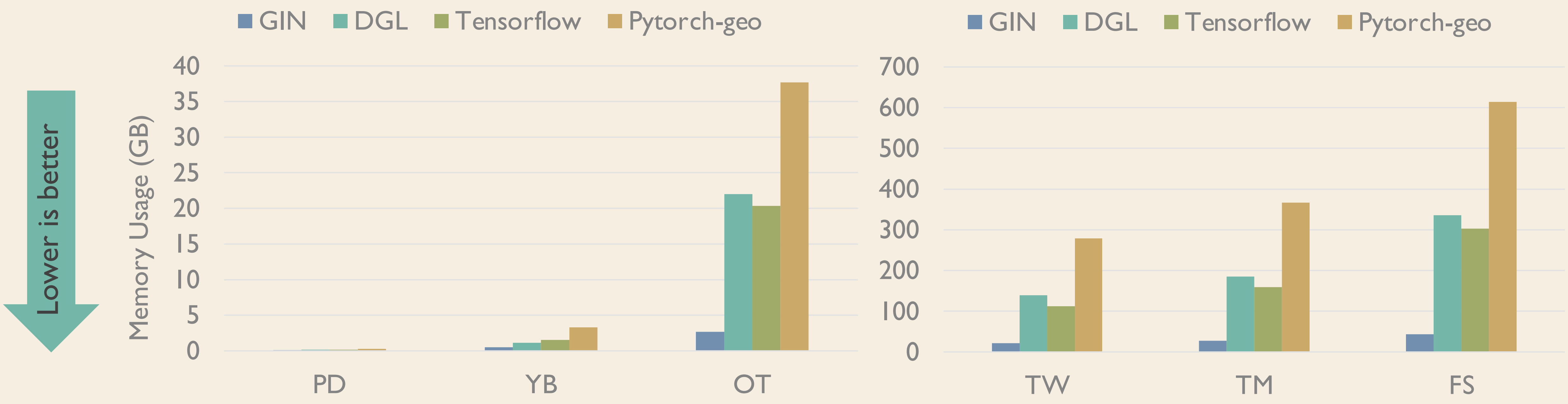
Graph (Abbr.)	|Vertex|	|Edge|	Avg.degree
Pubmed (PD)	19.7K	108.4K	5
Youtube (YB)	1.1M	5.9M	6
Orkut (OT)	3.3M	117.1M	39
Twitter-www (TW)	41.6M	1.4B	34
Twitter-mpi (TM)	52.5M	1.9B	37
Friendster (FS)	65.6M	3.6B	56

Speedup over Baselines



- Overall speedups: 10.81x over DGL, 10.21x over Tensorflow, 71.64x over Pytorch-geo

Memory Usage



- Average memory reduction: 86% over DGL, 72% over Tensorflow, 92% over Pytorch-geo

Conclusion

- Existing solutions for GNN inference are suffering from poor performance or high programming complexity.
- We propose GIN, a compiler-based framework for high-performance GNN inference.
- Average 31.44x speedup over existing solutions.

Thank You