Large-scale Graph Neural Network Services through Computational SSD and In-Storage Processing Architectures

Miryeong Kwon, Donghyun Gouk, Sangwon Lee, Myoungsoo Jung

Computer Architecture and Memory systems Laboratory
First Step
High-level summary of talk

GNN have shown great success

High accuracy
Well accelerated

GNN preprocessing is missed out on

Current GNN works are only focusing on GNN algorithms

Now, we need “HolisticGNN”

By leveraging Computational SSD

GNN preprocessing
GNN algorithm
Graph Neural Networks (GNN)

Why is it emerging?

**Conventional CNN Model**

Regular data in Euclidean space
(Learning information: "Euclidean distance")

**Emerging GNN Model**

Irregular data in non-Euclidean space
(Learning information: "Relationship")

**Response of CNN model**

“Women near the sofa”

**Query image**

Characteristic: “pain”

**Response of GNN model**

“pain”

How can GNN algorithm learn the relationship?

*Image source: Personalized Image Retrieval with Sparse Graph Representation Learning (KDD’20)*
Graph Neural Networks (GNN)

GNN algorithm

Input

Graph structure

<table>
<thead>
<tr>
<th>0.1</th>
<th>0.8</th>
<th>1</th>
<th>0.2</th>
<th>0</th>
<th>1</th>
<th>0.8</th>
<th>0.7</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0.1</td>
<td>1</td>
<td>0.8</td>
<td>0.1</td>
<td>1</td>
<td>0.2</td>
<td>0</td>
</tr>
<tr>
<td>0.4</td>
<td>0.8</td>
<td>1</td>
<td>0.1</td>
<td>0.2</td>
<td>0.8</td>
<td>0.2</td>
<td>0</td>
<td>0.4</td>
</tr>
<tr>
<td>0.2</td>
<td>0.3</td>
<td>0.2</td>
<td>0.8</td>
<td>0.5</td>
<td>0.4</td>
<td>0.6</td>
<td>0.9</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Node embedding

#1: Aggregation

#2: Transformation

What do we have to do before GNN algorithm execution?
Graph Neural Networks (GNN)

GNN algorithm

1. We have to prepare neighbor-oriented graph structure

2. We need small input data which can be loaded into accelerator memory
GNN Preprocessing

Graph preprocessing: to prepare neighbor-oriented graph structure

Graph structure is stored as “edge array” which is update-friendly

Graph preprocessing converts edge array to “adjacency list” which is neighbor-oriented
GNN Preprocessing

Batch preprocessing: to prepare small graph

Insight: “Node sampling” can significantly reduce the amount of data to process without an accuracy loss
GNN Preprocessing

Batch preprocessing: to prepare small graph

Graph structure sampling

Embedding sampling
End-to-End GNN Inference

Visualization
End-to-End GNN Inference

Execution time analysis

Oops.. **Graph preprocessing** and **embedding I/O** is a dominant contributor to the end-to-end GNN inference (NOT pure GNN inference!)

Graph size (# of edges)

Normalised Execute Time (%)
Design Questions
Then, what does GNN acceleration look like?

- **Graph preprocessing** (CPU)
- **Embedding I/O** (CPU)
- **Store graph directly** as a neighbor-oriented format (But also, update-efficient)
- **Process end-to-end GNN inference** near storage
HolisticGNN
Adopts the concept of computational SSD (CSSD)

CSSD decouples the compute unit from the storage resources unlike conventional ISP (In-Storage Processing)
HolisticGNN

“Hardware/Software Co-Programmable Framework” for CSSDs

Our proposed Hardware/Software co-programmable framework is executing on FPGA
HolisticGNN
“Hardware/Software Co-Programmable Framework” for CSSDs

Shell region is for essential HW logics of HolisticGNN
HolisticGNN

“Hardware/Software Co-Programmable Framework” for CSSDs

User region is for GNN inference acceleration (user-customizable)
HolisticGNN

“Hardware/Software Co-Programmable Framework” for CSSDs

HolisticGNN also provides three types of algorithm accelerators

Octa-core
- Core0
- Core1
- Core2
- Core3
- Core4
- Core5
- Core6
- Core7

Many SAs
- Systolic array
- Systolic array
- Systolic array

Hetero
- Vector processor
- Systolic array

FPGA
- DRAM
- DRAM
- DRAM
- DRAM

User
- Co-processor ports
- System bus lanes

Shell
HolisticGNN
“Hardware/Software Co-Programmable Framework” for CSSDs

O3 core executes **GraphStore** and **GraphRunner**
HolisticGNN
“Hardware/Software Co-Programmable Framework” for CSSDs

GraphStore converts edge array to adjacency list and store it to SSD
HolisticGNN
“Hardware/Software Co-Programmable Framework” for CSSDs

GraphRunner processes both GNN preprocessing and algorithm

GraphRunner can access graph data via GraphStore APIs

FPGA

DRAM

DRAM

DRAM

DRAM

FPGA
Experimental Setup
HolisticGNN prototype

FPGA
DRAM
DRAM
DRAM
DRAM

FPGA
CTRL
DRAM
NAND Flash
NAND Flash
NAND Flash
NAND Flash

SSD
14nm FPGA

NVMe SSD
Evaluation Results
End-to-End latency comparison

Small graph: 1.69x

- RTX 1060
- GTX 3090
- HolisticGNN

Norm. Exec. Time

Large graph

100.4x faster

chameleon
cites
cora
ml
dblp
small
cora
full
physics
road-tx
road-pa
large
youtube

AMD Ryzen 3900X
DDR4-2666 64GB
14 nm FPGA
CSSD Prototype
SSD

GTX 1060

0.00
0.01
0.02
0.03

0
0.1
1
Evaluation Results

Energy Consumption

33.2x and 16.3x better than GTX 3090, RTX 1060

Due to low-power computing of FPGA

453.2x lower
Demonstration

GNN execution in our HolisticGNN prototype

Demonstration Video Link: https://www.youtube.com/watch?v=b5fZBESH1TM
Conclusion

HolisticGNN is a “hardware/software co-programmable framework for computational SSDs”

1) Holistic solution for both GNN algorithm and preprocessing
2) Fast and energy-efficient near-storage inference infrastructure
3) Easy-to-use and user-customizable
Thank You

Contact: Miryeong Kwon (mkwon@camelab.org)


Acknowledgment: This research is supported by Samsung Research Funding & Incubation Center of Samsung Electronics (SRFC-IT2101-04). Myoungsoo Jung is the corresponding author.