Hardware/Software Co-Programmable Framework for Computational SSDs to Accelerate Deep Learning Service on Large-Scale Graphs

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
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
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
1. Background

2. Motivation and Design Considerations

3. Overview of HolisticGNN Framework

4. Details of HolisticGNN Components

5. Evaluation
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”

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

Why is it emerging?

Conventional CNN Model

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

Response of CNN model

“Women near the sofa”

Query image

Characteristic: “pain”

Emerging GNN Model

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

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

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>0.8</td>
<td>1</td>
<td>0.2</td>
<td>0</td>
<td>1</td>
<td>0.8</td>
</tr>
<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>
</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>
</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>
</tr>
</tbody>
</table>

Node embedding
Graph Neural Networks (GNN)

GNN algorithm

Graph structure

Node embedding
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

MLP
Graph Neural Networks (GNN)

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
1. Background

2. Motivation and Design Considerations

3. Overview of HolisticGNN Framework

4. Details of HolisticGNN Components

5. Evaluation
End-to-End GNN Inference

Visualization
End-to-End GNN Inference

Execution time analysis

Graph size (# of edges)

Normalized Execute Time (%)

Embedding I/O (CPU)

Batch preprocessing (CPU)

Pure inference (GPU)

Graph preprocessing (CPU)

Entire edge array

Entire adj. list

Entire embed

Sampled embed

SSD

Host CPU

Host DRAM

GPU

GPU DRAM

OAD

11010101001100100111001111

110101001010

chameleon
citeseer
corail
dblpfull
cs
corafull
physics
road-tx
road-pa
youtube
road-ca
wikitalk
ljournal

OOM
Oops.. **Graph preprocessing** and **embedding I/O** is dominant contributor of the end-to-end GNN inference (NOT pure GNN inference!)
**Design Questions**
Then, what does GNN acceleration look like?

- **Graph preprocessing (CPU)**
  - Store graph directly as a neighbor-oriented format (But also, update-efficient)

- **Embedding I/O (CPU)**
  - Process end-to-end GNN inference near storage
1. Background

2. Motivation and Design Considerations

3. Overview of HolisticGNN Framework

4. Details of HolisticGNN Components

5. Evaluation
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

“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
Core0, Core1, Core2, Core3, Core4, Core5, Core6, Core7

Hetero
Core0, Core1, Core2, Core3, Core4, Core5, Core6, Core7

Systolic array
Systolic array
Systolic array
Systolic array
Systolic array
Systolic array
Systolic array
Systolic array

FPGA DRAM DRAM DRAM DRAM

Shell User Co-processor ports System bus lanes

O3 Core Bus Xbuilder Engine
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

O3 Core

Adjacency list

Embedding

FPGA

DRAM

DRAM

DRAM

DRAM
GraphStore
Graph-centric archiving system

GraphStore separates SSD space into two for making sure update-efficient
GraphStore

Graph-centric archiving system

Why are there two types of neighbor list?
GraphStore
Graph-centric archiving system

Insight comes from power-law distribution of degree (# of neighbors)
※ Nature characteristic of graph
GraphStore
Graph-centric archiving system

Page size

frequency(d)

Low-degree type

High-degree type

High-degree type mapping table

V1 → LPN0

LPN0 (High-degree type)

V1

Low-degree type mapping table

V2 LPN1

V2 V3

LPN1 (Low-degree type)

V4 LPN2

V4 V5

LPN2 (Low-degree type)
GraphStore
Graph-centric archiving system

If new neighbor is added, GraphStore allocates one more LPN (LPN4) to Vertex 1 (V1)

High-degree type mapping table

V1 → LPN0 → LPN4

Low-degree type mapping table

V2 → LPN1
V4 → LPN2

LPN0 (High-degree type)

LPN4 (High-degree type)

LPN2 (Low-degree type)
As there are multiple neighbor lists in a single LPN, low-degree type mapping table recorded the lowest vertex ID.
GraphRunner
Programmable inference model

Programmer (user)

GraphRunner

Execution Semantic (e.g., Kernel)

Program semantic (e.g., Operation)

A → Op1 → B
C

A → Kernel1 → B
C

Programmer programs the GNN inference sequence as a DFG (Dataflow graph)

GraphRunner converts operation to kernel and executes the DFG
GraphRunner
Programmable inference model

Programner (user)
GraphRunner
Hardware
Execution Semantic (e.g., Kernel)

BTW, why are there gap between program semantic and execution semantic?
GraphRunner
Programmable inference model

Because there are many devices which can process the same operation!

<table>
<thead>
<tr>
<th>Device table</th>
<th>Operation table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device Name</td>
<td>Priority</td>
</tr>
<tr>
<td>&quot;CPU&quot;</td>
<td>50</td>
</tr>
<tr>
<td>&quot;Accelerator1&quot;</td>
<td>150</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operation Name</th>
<th>Kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Vector addition&quot;</td>
<td>&lt;&quot;CPU&quot;, ptr&gt;</td>
</tr>
<tr>
<td>&lt;&quot;Accelerator1&quot;, ptr&gt;</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Memory space

Vector addition (CPU code)
Vector addition (Accelerator1 code)
### GraphRunner

**Programmable inference model**

Users can add their own accelerator and also accelerator’s kernel 😊

#### Device table

<table>
<thead>
<tr>
<th>Device Name</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;CPU&quot;</td>
<td>50</td>
</tr>
<tr>
<td>&quot;Accelerator1&quot;</td>
<td>150</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

#### Operation table

<table>
<thead>
<tr>
<th>Operation Name</th>
<th>Kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Vector addition&quot;</td>
<td>&lt;&quot;CPU&quot;, ptr&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;&quot;Accelerator1&quot;, ptr&gt;</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

#### Memory space

- Vector addition (CPU code)
- Vector addition (Accelerator1 code)
- Vector addition (Accelerator2 code)
1. Background

2. Motivation and Design Considerations

3. Overview of HolisticGNN Framework

4. Details of HolisticGNN Components

5. Evaluation
Experimental Setup

HolisticGNN prototype
## Experimental Setup

### Graph dataset

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Vertices</th>
<th>Edges</th>
<th>Feature length</th>
</tr>
</thead>
<tbody>
<tr>
<td>chmleon</td>
<td>2.3K</td>
<td>65K</td>
<td>2326</td>
</tr>
<tr>
<td>citeseer</td>
<td>2.1K</td>
<td>9K</td>
<td>3704</td>
</tr>
<tr>
<td>corma</td>
<td>3.0K</td>
<td>19K</td>
<td>2880</td>
</tr>
<tr>
<td>dblpfull</td>
<td>17.7K</td>
<td>123K</td>
<td>1639</td>
</tr>
<tr>
<td>cs</td>
<td>18.3K</td>
<td>182K</td>
<td>6805</td>
</tr>
<tr>
<td>corafull</td>
<td>19.8K</td>
<td>147K</td>
<td>8710</td>
</tr>
<tr>
<td>physics</td>
<td>34.5K</td>
<td>530K</td>
<td>8415</td>
</tr>
<tr>
<td>road-tx</td>
<td>1.39M</td>
<td>3.84M</td>
<td>4353</td>
</tr>
<tr>
<td>road-pa</td>
<td>1.09M</td>
<td>3.08M</td>
<td>4353</td>
</tr>
<tr>
<td>youtube</td>
<td>1.16M</td>
<td>2.99M</td>
<td>4353</td>
</tr>
<tr>
<td>road-ca</td>
<td>1.97M</td>
<td>5.53M</td>
<td>4353</td>
</tr>
<tr>
<td>wikitalk</td>
<td>2.39M</td>
<td>5.02M</td>
<td>4353</td>
</tr>
<tr>
<td>ljournal</td>
<td>4.85M</td>
<td>68.99M</td>
<td>4353</td>
</tr>
</tbody>
</table>

Small (<1M Edge)

Large (>3M Edge)
Evaluation Results

End-to-End latency comparison

Small graph: 1.69x

- RTX 1060
- GTX 3090
- HolisticGNN

Graph showing normalized execution time for various workloads.
Evaluation Results
End-to-End latency comparison

- RTX 1060
- GTX 3090
- HolisticGNN

Norm. Exec. Time

1.0
0.5
0.0

chameleon
ctiseer
coraml
dblpfull
Cs
corafull
physics
road-tx
road-pa
youtube

Large graph

100.4x faster
Evaluation Results

Energy Consumption

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

- Small graph
- Large graph

RTX 1060 | GTX 3090 | HolisticGNN
Evaluation Results
Energy Consumption

Due to low-power computing of FPGA

453.2x lower
Demo

GNN execution in our HolisticGNN prototype
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)