GraphSAR: A Sparsity-Aware Processing-in-Memory Architecture for Large-Scale Graph Processing on ReRAMs

Guohao Dai\textsuperscript{1}, Tianhao Huang\textsuperscript{1,2}, Yu Wang\textsuperscript{1}, Huazhong Yang\textsuperscript{1}, John Wawrzynek\textsuperscript{3}

\textsuperscript{1}Tsinghua University, \textsuperscript{2}MIT, \textsuperscript{3}UCB
dgh14@mails.tsinghua.edu.cn

1/22/2019

Special thank to Dr. Shaungchen Li and Gushu Li from UCSB
Content

• Background
• Motivation
• Related Work
• GraphSAR design
• Experiment Results
• Conclusion
Content

• Background
  • Motivation
  • Related Work
• GraphSAR design
• Experiment Results
• Conclusion
Graphs are widely used!

- Graph: represent data and their relationships
- Application: social network analysis, neural network modeling, user behavior analysis, brain network modeling …
- System & Architecture: support for graph-based applications
Application I: PageRank

- Sorting billions of pages according to key words in one second
  - Graph: 2.9 b, 0.36 s
  - PageRank: 0.2 b, 0.30 s

- Google PageRank Algorithm
  - The rank of a page depends on ranks of pages which link to it

- Network = Graph

\[
\text{PageRank}(p_i) = \frac{1-d}{N} + d \sum_{p_j \in M(p_i)} \frac{\text{PageRank}(p_j)}{L(p_j)}
\]
Application II: Film Recommendation

• Collaborative filtering based on similar users

• **ALS**: Alternating Least Squares
  – Minimize Mean Square Error (MSE)
    • Calculating using tags
    • Recommending using non-tags

• Sparse matrix = Graph

\[
\begin{align*}
  u_i & = \arg \min_w \sum_{j \in N[i]} (r_{ij} - m_j \cdot w)^2 \\
  m_j & = \arg \min_w \sum_{i \in N[j]} (r_{ij} - u_i \cdot w)^2
\end{align*}
\]

Application III: Deep Learning

- Data & relationship = Graph
  - Neuron: vertex
  - Synapse: edge
  - Stimulus intensity: value

- Using graphs to represent different neural networks

Generality requirement

- High-level abstraction model
  - Read-based/Queue-based Model for BFS/APSP [Stanford, PACT’10] ×
  - GAS Model [Google, SIGMOD’10] ✓
- In GAS (Gather-Apply-Scatter) Model
  - Different algorithms → Different Apply functions
  - Traverse edges and scatter src to dst

Content

• Background

• Motivation

• Related Work

• GraphSAR design

• Experiment Results

• Conclusion and Future Work
Characteristics of graph processing

• Example: Breadth-First Search
  – Root: v1
  – Generate BFS tree

• Unstructured
  – Degree = 1: v7, 8, 9, 10
  – Degree = 8: v5
  – Data driven
    – Example: v5, transfer updated value to 5 vertices

• Unbalanced
  – Balanced in Lv.1 → unbalanced in Lv.2

• Poor locality
  – 4 vertices in Lv.1 spread all over graph
Challenges in graph processing

- Key in graph processing: **efficient data transferring**

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Difficulties</th>
<th>Solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unstructured</td>
<td><strong>Heavy traffics</strong> [Zhu_OSDI_2016]</td>
<td>Lower traffics</td>
</tr>
<tr>
<td>Unbalanced</td>
<td>System</td>
<td>PowerG.</td>
</tr>
<tr>
<td>Data Driven</td>
<td>Mem. Ref.</td>
<td>95.8G</td>
</tr>
<tr>
<td>Poor Locality</td>
<td>Comm. (GB)</td>
<td>115</td>
</tr>
<tr>
<td></td>
<td><strong>Memory-hungry</strong> [Nai_HPCA_2017]</td>
<td>Higher bandwidth</td>
</tr>
<tr>
<td></td>
<td>IPC (8 cores Xeon E5)</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Cache miss</strong> [Zhu_OSDI_2016]</td>
<td>Sequential access</td>
</tr>
<tr>
<td></td>
<td>System</td>
<td>Ligra</td>
</tr>
<tr>
<td></td>
<td>IPC</td>
<td>0.408</td>
</tr>
<tr>
<td></td>
<td>LLC Miss</td>
<td>43.9%</td>
</tr>
</tbody>
</table>
Memristor & ReRAM

- **Memristor**
  - Resistance can be changed by voltage

- **Storage**
  - Memristor crossbar
  - Using changeable resistance to store information

- **Computation**
  - Processing-in-memory, 10x ~ 100x energy efficiency improvement compared with conventional von Neumann architecture
Content

• Background
• Motivation
• **Related Work**
• GraphSAR design
• Experiment Results
• Conclusion
RPBFS [Hong Kong PolyU, NVMSA 17]

• Designed for Breadth-First Search
  – Using 1 bit to represent status of a vertex
  – Graph Bank: edge storage
    • Parallel processing
  – Master Bank: vertex storage
    • Centralized processing

• Performance
  – 33.8x speedup against CPU
  – 16.0x speedup against GPU

• However…
  – Only for BFS
  – Centralization scheme → Scalability problem
HyVE [Ours, DATE 18]

- Memory energy efficiency difference due to patterns
  - Different patterns \(\rightarrow\) corresponding memories \(\rightarrow\) hybrid memory

- Vertex: random Read/Write locally \(\rightarrow\) SRAM
  - sequential Read/Write globally \(\rightarrow\) DRAM

- Edge: sequential Read globally \(\rightarrow\) ReRAM

- Performance
  - 114x energy efficiency improvement against CPU+DRAM
  - Memory subsystem energy consumption < 50%
GraphR [Duke, HPCA 18]

- Matrix-vector representation for graph processing
  - Src vertex vector, adjacency matrix $\rightarrow$ Dst vertex vector
  - MVM (e.g., PageRank): direct mapping
  - Non-MVM (e.g., BFS): activating each row sequentially

- Divide a large adjacency matrix into small blocks

(a) Vertex Program in Graph View

(b) Vertex Program in Matrix View
GraphR [Duke, HPCA 18]

- MVM (e.g., PageRank): direct mapping
GraphR [Duke, HPCA 18]

- Non-MVM (e.g., BFS): activating each row sequentially
GraphR [Duke, HPCA 18]

- Compared with CPU
  - Speedup: 16.01x
  - Energy efficiency: 33.82x
- Compared with GPU
  - Speedup: 1.69x ~ 2.19x
  - Energy efficiency: 4.77x ~ 8.91x
• Energy efficiency of graph processing can be improved by using ReRAM

<table>
<thead>
<tr>
<th></th>
<th>RPBFS</th>
<th>HyVE</th>
<th>GraphR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm</td>
<td>Only BFS</td>
<td>General purposed</td>
<td>General purposed</td>
</tr>
<tr>
<td>Storage</td>
<td>RRAM</td>
<td>Hybrid</td>
<td>RRAM</td>
</tr>
<tr>
<td>Computation</td>
<td>CMOS</td>
<td>CMOS</td>
<td>RRAM/CMOS</td>
</tr>
</tbody>
</table>
Content

• Background
• Motivation
• Related Work

• GraphSAR design
• Experiment Results
• Conclusion
However...

- **GraphR: Writing ReRAM (adjacency list → block)**

  \[(\text{row}, \text{val}) \rightarrow \text{colptr} \]

<table>
<thead>
<tr>
<th>row</th>
<th>val</th>
<th>colptr</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3.4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>0.3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>1.7</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>0.8</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

  block

- **Write-and-verify scheme of ReRAM:** Heavy writing overheads

- **Sparsity of graphs:** Low parallelism
GraphSAR Design

- Design I: Processing-in-memory

- Design II: Sparsity-aware partitioning
Design I: Processing-in-memory

- Conversion leads to low parallelism/heavy writing overheads
- Directly storing blocks on ReRAM

Problem I:
- Heavy writing overheads
- Low parallelism
Design II: Sparsity-aware partitioning

- Storing 8*8 blocks leads to memory space overheads

- Sparsity-aware partitioning
  - Divide large sparse block into small ones
  - Drop empty blocks

GraphR  \[\rightarrow\]  GraphSAR

Problem II:
- Memory space overheads
Opt. I: Single bit implementation

- For algorithms on unweighted graphs
  - e.g., PageRank, BFS, etc.
  - Scatter the same value to neighbors
  - One bit of an edge to represent connectivity

GraphR

GraphSAR
Opt. II: lightweight clustering

- Vertex clustering → less blocks to be processed
  - Consecutive vertices in the original adjacency list tend to gather
  - Original indices are not continuous
  - Assign new indices to vertices when reading edges
  - Only $O(n)$ complexity

Original edge list

<table>
<thead>
<tr>
<th>src</th>
<th>dst</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>28</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
</tr>
<tr>
<td>3</td>
<td>39</td>
</tr>
<tr>
<td>3</td>
<td>54</td>
</tr>
<tr>
<td>3</td>
<td>108</td>
</tr>
<tr>
<td>3</td>
<td>152</td>
</tr>
<tr>
<td>3</td>
<td>178</td>
</tr>
<tr>
<td>3</td>
<td>182</td>
</tr>
<tr>
<td>3</td>
<td>214</td>
</tr>
<tr>
<td>3</td>
<td>271</td>
</tr>
<tr>
<td>3</td>
<td>286</td>
</tr>
<tr>
<td>3</td>
<td>300</td>
</tr>
<tr>
<td>3</td>
<td>348</td>
</tr>
<tr>
<td>3</td>
<td>349</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>src</th>
<th>dst</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>371</td>
</tr>
<tr>
<td>3</td>
<td>567</td>
</tr>
<tr>
<td>3</td>
<td>581</td>
</tr>
<tr>
<td>3</td>
<td>584</td>
</tr>
<tr>
<td>3</td>
<td>586</td>
</tr>
<tr>
<td>3</td>
<td>590</td>
</tr>
<tr>
<td>3</td>
<td>604</td>
</tr>
<tr>
<td>3</td>
<td>611</td>
</tr>
<tr>
<td>3</td>
<td>828</td>
</tr>
</tbody>
</table>

Edge list

<table>
<thead>
<tr>
<th>src</th>
<th>dst</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
</tr>
</tbody>
</table>

Interval\(_0\) = \{0, 1, 2\}
Interval\(_1\) = \{3, 4, 5\}
Interval\(_2\) = \{6, 7, 8\}
• According to the sparsity-aware partitioning
  – Edges are stored into edge lists and block lists
  – Edge lists and block lists are stored into different banks for the alignment purpose
Working flow

• Selecting: activate a block for processing
  – GraphR: activate a row

• Processing: process edge list and block list separately
  – GraphR: treat a block with one edge as a block
Content

• Background
• Motivation
• Related Work
• Architecture and Detailed Implementation

• **Experiment Results**

• Conclusion and Future Work
Configuration

• **Datasets**

<table>
<thead>
<tr>
<th></th>
<th>WV</th>
<th>HP</th>
<th>GG</th>
<th>YT</th>
<th>PK</th>
<th>CA</th>
<th>WT</th>
<th>TW</th>
<th>FS</th>
<th>YH</th>
</tr>
</thead>
<tbody>
<tr>
<td># V</td>
<td>7.12k</td>
<td>34.5k</td>
<td>0.88m</td>
<td>1.13m</td>
<td>1.63m</td>
<td>1.97m</td>
<td>2.39m</td>
<td>41.7m</td>
<td>65.6m</td>
<td>1.41b</td>
</tr>
<tr>
<td># E</td>
<td>0.10m</td>
<td>0.42m</td>
<td>5.11m</td>
<td>2.99m</td>
<td>30.6m</td>
<td>2.77m</td>
<td>5.02m</td>
<td>1.47b</td>
<td>1.81b</td>
<td>6.64b</td>
</tr>
<tr>
<td>type</td>
<td>social</td>
<td>citation</td>
<td>web</td>
<td>community</td>
<td>social</td>
<td>road</td>
<td>communication</td>
<td>social</td>
<td>community</td>
<td>web</td>
</tr>
</tbody>
</table>

• **Algorithms**
  – PageRank, Breadth-first Search, Connected Components

• **Configuration**
  – ReRAM simulator: NVSim
    • read/write energy consumption: 1.08pJ/7.4pJ
    • read/write latency: 29.31ns/50.88ns
    • HRS/LRS resistance: 25MΩ/50KΩ
    • read/write voltage: 0.7V/2V
    • current of LRS/HRS: 40µA/2µA

Haewoon Kwak et al. What is twitter, a social network or a news media?
Results – Opt. I

- Single bit implementation
  - Speedup: 1.15x
  - Energy efficiency improvement: 2.37x
  - Energy-Delay Product reduction: 2.73x
Results – Opt. II

- Lightweight clustering
  - Speedup: 1.30x
  - Energy efficiency improvement: 1.37x
  - Energy-Delay Product reduction: 1.78x
Results – Overall performance

• Compared with GraphR (already used Opt. I & II)
  – Speedup: 1.85x
  – Energy efficiency improvement: 4.43x
  – Energy-Delay Product reduction: 8.19x
Results – Memory space overheads

• Compared with using adjacency list (need to write ReRAM)
  – Only 1.54x storage overheads
  – 46.87x storage overheads when storing 8*8 blocks
Content

• Background
• Motivation
• Related Work
• Architecture and Detailed Implementation
• Experiment Results

• Conclusion
Conclusion

• **GraphSAR**
  – Improving energy efficiency/Accelerating graph processing using ReRAM
  – Design for different graph algorithms
  – Both computation and storage optimization
References


Thank you!

Q & A