NDMiner: Accelerating Graph Pattern Mining Using Near Data Processing

Nishil Talati, Haojie Ye, Yichen Yang, Leul Belayneh, Kuan-Yu Chen, David Blaauw, Trevor Mudge, and Ronald Dreslinski

International Symposium on Computer Architecture (ISCA) 2022
Session 2B: Graph Applications
Graph Pattern Mining (GPM)

**Input Graph**

1 2
0 3 4

**Input Pattern**

Goal: Find the unique instances of input patterns in an input graph

**Mined Patterns in Input Graph**

0
1 3
2 3

**Applications of GPM**

Cyber-security
Bioinformatics
Social Network Analysis
Spam Detection
Many more…
Graph Pattern Mining (GPM)

**Input Graph**

**Mined Patterns in Input Graph**

**Applications of GPM**

---

**Goal**: Find the unique instances of input patterns in an input graph

---

**Are today’s hardware platforms adequate?**

---

**Many more…**
Performance Bottleneck In GPM

• Set operations dominate execution time
  • Set intersection and set difference

• Control flow and memory instructions
  • Data-dependent branch instructions
  • Memory accesses to irregular graph data structure

• Prior hardware works
  • Design domain-specific accelerator architectures
  • Employ generic Near Data Processing (NDP) architectures

FlexMiner [ISCA 2021], SISA [MICRO 2021], IntersectX [arXiv 2021]
Contributions of NDMiner

• NDMiner combines domain specialization and NDP

• **Load elision unit**
  • Reduces unnecessary data transfer

• **Loop nest flattening**
  • Improves algorithmic efficiency

• **Set operation reordering**
  • Enhances bank-level parallelism
In This Talk

• NDMiner combines domain specialization and NDP

• Load elision unit
  • Reduces unnecessary data transfer

• Loop nest flattening
  • Improves algorithmic efficiency

• Set operation reordering
  • Enhances bank-level parallelism
Why NDP For GPM?

<table>
<thead>
<tr>
<th>Workload Property</th>
<th>Feasibility of NDP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Triangle Counting

```python
for u0 in input_graph:
    Nu0 = G.out_neigh(u0)
    for u1 in Nu0:
        if u1 >= u0: break
        Nu1 = G.out_neigh(u1)
        Nu0u1 = Intersection(Nu0, Nu1)
        for u2 in Nu0u1:
            if u2 >= u1: break
            {u0, u1, u2}: triangle
```
## Why NDP For GPM?

<table>
<thead>
<tr>
<th>Workload Property</th>
<th>Feasibility of NDP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wasteful data transfers</td>
<td>In-memory data processing</td>
</tr>
</tbody>
</table>

### Triangle Counting

```python
for u0 in input_graph:
    Nu0 = G.out_neigh(u0)
    for u1 in Nu0:
        if u1 >= u0: break
        Nu1 = G.out_neigh(u1)
        Nu0u1 = Intersection(Nu0, Nu1)
        for u2 in Nu0u1:
            if u2 >= u1: break
            {u0, u1, u2}: triangle
```
## Why NDP For GPM?

<table>
<thead>
<tr>
<th>Workload Property</th>
<th>Feasibility of NDP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wasteful data transfers</td>
<td>In-memory data processing</td>
</tr>
<tr>
<td>Employ simple arithmetic ops</td>
<td>Cost-effective logic integration</td>
</tr>
</tbody>
</table>

### Triangle Counting

```python
for u0 in input_graph:
    Nu0 = G.out_neigh(u0)
    for u1 in Nu0:
        if u1 >= u0: break
    Nu1 = G.out_neigh(u1)
    Nu0u1 = Intersection(Nu0, Nu1)
    for u2 in Nu0u1:
        if u2 >= u1: break
    {u0, u1, u2}: triangle
```
Why NDP For GPM?

<table>
<thead>
<tr>
<th>Workload Property</th>
<th>Feasibility of NDP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wasteful data transfers</td>
<td>In-memory data processing</td>
</tr>
<tr>
<td>Employ simple arithmetic ops</td>
<td>Cost-effective logic integration</td>
</tr>
<tr>
<td>Read-only data structures</td>
<td>No coherence issue with CPU caches</td>
</tr>
</tbody>
</table>

NDP is an attractive candidate to accelerate GPM

**Triangle Counting**

```python
for u0 in input_graph:
    Nu0 = G.out_neigh(u0)
    for u1 in Nu0:
        if u1 >= u0: break
        Nu1 = G.out_neigh(u1)
        Nu0u1 = Intersection(Nu0, Nu1)
        for u2 in Nu0u1:
            if u2 >= u1: break
            {u0, u1, u2}: triangle
```
Baseline NDP Architecture

Type Of Computation

Offload set operations to NDP

Where To Compute

Outside memory banks

Buffer chip of DRAM

- Set Operation Unit
- Network
- DRAM Banks
- Row Buffer
- nPE
- Int/Diff
Wasteful Data Transfer Due To Symmetry Breaking Constraints

Example Values

\[ u_0 = 15, \ u_1 = 12 \]
\[ N_{u_0} = \{5, 7, 11, 19, 22, 40\} \]
\[ N_{u_1} = \{5, 11, 19, 22, 40\} \]

\[ \text{Intersection}(N_{u_0}, N_{u_1}) \]
\[ \{5, 11, 19, 22, 40\} \]

60% data wasted

Typical GPM Algorithm

\[ \ldots \]
\[ N_{u_0u_1} = \text{Intersection}(N_{u_0}, N_{u_1}) \]
\[ \text{for } u_2 \text{ in } N_{u_0u_1}: \]
\[ \text{if } u_2 \geq u_1: \text{ break} \]
\[ \ldots \]

Symmetry Breaking Constraint
Ordering on vertex ordering to avoid redundant counting

Insight: Symmetry breaking constraints results in 66.5% data wastage
Proposal: Load elision unit prevents unnecessary loads by breaking symmetry in hardware.
Insight: Size-limited memory controller queues prevent GPM workloads from exploiting in-DRAM parallelism.
Effect of increasing memory controller window size on GPM performance

Larger window size:
Better reordering opportunity

Insight: Size-limited memory controller queues prevent GPM workloads from exploiting in-DRAM parallelism

How to reorder set operations from a large window size without hurting performance?
Proposal: Set operation reordering based on vertex IDs to improve bank-level parallelism.
More Details In The Paper

• Loop nest flattening for sparse GPM
• ISA extensions to support NDP
• NDMiner programming model
• Detailed NDP unit hardware design
• Modifications on the memory controller to support NDP
• Command scheduling
• . . .
Evaluation Methodology

• Cycle-accurate simulation using Ramulator
  • Modeling of NDP logic based on detailed RTL models

• Comparison with state-of-the-art baselines
  • GraphPi [SC 2020] (software baseline)
  • GraphPi algorithms on GAPBS data structures (software baseline)
  • Pangolin [VLDB 2020] (software baseline)
  • FlexMiner [ISCA 2021] (hardware baseline)

• Five input graph datasets with diverse sizes and connectivity
Evaluation Methodology

Cliques (Dense Shapes)
- Triangle Counting
- 4-Clique Mining
- 5-Clique Mining

Sparse Shapes
- Diamond
- Four Cycle

Mixed Shapes (Dense + Sparse)
- 3-Motif Mining
- 4-Motif Mining
Domain-specific optimizations improve the performance of NDMiner by 12.7x, on average, compared to a baseline NDP system.
NDMiner outperforms SOTA software baselines by 6.4-10.9x, on average, and SOTA hardware baselines by 2.5x, on average.
Conclusion

• GPM workloads bottlenecked by control flow and irregular memory accesses

• NDMiner: domain-specific near data processing (NDP) architecture

• Results in data movement and algorithmic complexity reduction, and improved parallelism to gain performance
Thank You!
Backup Slides
Redundant Loads in Sparse GPM

Redundant fetching of $N_{u0u1}$ or $u2$ and $u3$

Widespread in sparse GPM

Not present in clique mining

Fully connected nature leads to tighter constraints

```
1. for u0 in V:
2.    Nu0 = G.out_neigh(u0)
3. for u1 in Nu0:
4.    if u1 >= u0: break
5.    Nu1 = G.out_neigh(u1)
6.    Nu0u1 = Intersection(Nu0, Nu1)
7. for u2 in Nu0u1: set reads
   Redundant
8. for u3 in Nu0u1:
9.    if u3 >= u2: break
10. num_diamonds++
```

Sparse GPM algorithms involve redundant reads
Loop Nest Flattening

\[ u_2, u_3 \in N_{u_0u_1} = \{0, 1, 2, 5\} \]

\[
\text{for } u_2 \text{ in } N_{u_0u_1}: \\
\quad \text{for } u_3 \text{ in } N_{u_0u_1}: \\
\quad \quad \text{if } u_3 \geq u_2: \text{ break}
\]

\(u_2, u_3 = \text{shift_record}(N_{u_0})\)
# Overhead Analysis

<table>
<thead>
<tr>
<th></th>
<th>Near-data PEs (nPEs)</th>
<th>Load Elision Unit (LEU)</th>
<th>Set Operation Reorder Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Location</strong></td>
<td>Buffer chip of DRAM</td>
<td>Buffer chip of DRAM</td>
<td>CPU memory controller front-end</td>
</tr>
<tr>
<td><strong>Area (mm²)</strong></td>
<td>0.01237</td>
<td>0.00096</td>
<td>0.4147</td>
</tr>
<tr>
<td><strong>Power (mW)</strong></td>
<td>18.45</td>
<td>0.36</td>
<td>32.76</td>
</tr>
</tbody>
</table>

16x nPEs and 32x LEUs in a DRAM DIMM
Energy Consumption

Energy consumption of different NDMiner configurations on a representative patents dataset.
## Qualitative Comparison to Prior Works

<table>
<thead>
<tr>
<th></th>
<th>Symm. Break.</th>
<th>NDP</th>
<th>Load Elision</th>
<th>Loop Nest Flattening</th>
<th>Op Reorder</th>
</tr>
</thead>
<tbody>
<tr>
<td>GraphZero [30]</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>GraphPi [48]</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Gramer [60]</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>FlexMiner [12]</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>SISA [8]</td>
<td>x</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>IntersectX [43]</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>NDMiner</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
## Input Graph Datasets

<table>
<thead>
<tr>
<th>Graph</th>
<th>#Vtx</th>
<th>#Edge</th>
<th>Size (MB)</th>
<th>Avg Degree</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wiki-vote (wi)</td>
<td>7.1k</td>
<td>103.7k</td>
<td>0.5</td>
<td>14.6</td>
<td>Voting network</td>
</tr>
<tr>
<td>pokec (po)</td>
<td>1.6M</td>
<td>30.6M</td>
<td>129.3</td>
<td>19.1</td>
<td>Social network</td>
</tr>
<tr>
<td>patents (pa)</td>
<td>3.7M</td>
<td>16.5M</td>
<td>91.8</td>
<td>4.4</td>
<td>Citation network</td>
</tr>
<tr>
<td>livejournal (lj)</td>
<td>4.0M</td>
<td>34.7M</td>
<td>162.8</td>
<td>8.7</td>
<td>Social network</td>
</tr>
<tr>
<td>orkut (or)</td>
<td>3.1M</td>
<td>117.8M</td>
<td>470.5</td>
<td>38.1</td>
<td>Social network</td>
</tr>
</tbody>
</table>