Fast Graph Pattern Mining Made Easy

Xuhao Chen
Outline

- Graph Pattern Mining (GPM) Use Cases
- Pangolin Overview
- Pangolin System Internals
  - How to Achieve Easy Programming?
  - How to Achieve High Performance?
- Evaluation
- Sandslash (optional)
Graph Pattern Mining (GPM)

- **Inputs**: a graph $G$ and a set of patterns $Q = \{P_1, P_2, ..., P_n\}$
- **Goal**: count or list all subgraphs in $G$ that matches $P_i \in Q$

We only consider connected patterns

**Input Graph $G$**

**Pattern $P$**

**Matched subgraphs**
Graph Analytics vs. Graph Pattern Mining

**Graph Analytics**
- Updates vertex/edge labels of the input graph

**Graph Pattern Mining**
- Discovers structural patterns in the input graph

**Shortest Path**

**PageRank**

**Motif Counting**

**Clique Listing**
Why Graph Pattern Mining? (1/3)

Social Networks

Recommender System

Bio-medicine

Chemical Engineering

Graph Database
Why Graph Pattern Mining? (2/3)

- Drug discovery

Chemical and Graphical representation of Flucytosine

Figure modified from K. Borgwardt and X. Yan (KDD’08)
Why Graph Pattern Mining? (3/3)

Transactions and involved entities

Small deposits followed by large withdrawal

Figure modified from Anand Padmanabha Iyer, et.al., ASAP: Fast, Approximate Graph Pattern Mining at Scale (OSDI’18)
Outline

- Graph Pattern Mining (GPM) Use Cases
- Pangolin Overview
  - Pangolin System Internals
    - How to Achieve Easy Programming?
    - How to Achieve High Performance?
- Evaluation
Pangolin: Efficient & Productive GPM Programming

Sequential! easy to code
run fast

Pattern pruning scheme

Pangolin API
Pangolin Runtime System (transparent parallelism)

Too Difficult!

OpenMP
nvidia. CUDA.

Input Graph
Pattern pruning scheme
Easy Programming in Pangolin

- \( k \)-Clique Listing

```cpp
bool toExtend(Subgraph sg, Vertex v) {
    return (sg.getLastVertex() == v);
}

bool toAdd(Subgraph sg, Vertex u) {
    for (v ∈ sg.getVertices() except last:
        if (!isConnected(v, u)) return false;
    return true;
}
```

<table>
<thead>
<tr>
<th>Hand Written</th>
<th>Pangolin</th>
</tr>
</thead>
<tbody>
<tr>
<td>State-of-the-art: (KList: WWW'18)</td>
<td>8 Lines of code !!!</td>
</tr>
</tbody>
</table>

3-clique i.e. triangle

4-clique

Pangolin API
user defined functions
Outline

- Graph Pattern Mining (GPM) Use Cases
- Pangolin Overview

- **Pangolin System Internals**
  - How to Achieve Easy Programming?
  - How to Achieve High Performance?

- Evaluation
Graph Mining Challenges

Easy Programming

High Performance
A “Naive” Abstraction

- Enumerate all the possible vertex combinations!!
- $G$ has $n$ vertices, $P$ has $k$ vertices: $\binom{n}{k}

\begin{align*}
\binom{3}{4} & = 4 : <1,2,3> \quad <1,2,4> \quad <1,3,4> \quad <2,3,4>
\end{align*}

Graphs are sparse!

We only consider connected patterns

Isomorphism test
The “Subgraph Tree” Abstraction

One abstraction for all the GPM problems
Pangolin does a lot for you!

<table>
<thead>
<tr>
<th>User responsibilities</th>
<th>Pangolin responsibilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pattern Specification</td>
<td>Subgraph Extension</td>
</tr>
<tr>
<td>Search space Pruning</td>
<td>Isomorphism Test</td>
</tr>
<tr>
<td></td>
<td>Symmetry Breaking</td>
</tr>
<tr>
<td></td>
<td>Parallelism Management</td>
</tr>
</tbody>
</table>
Work Done By Pangolin

Input Graph

Extend Stage

Subgraphs

Transparently parallel!

Reduce Stage

Pattern Map

[ , 1] [ , 2]
Graph Mining Challenges

Easy Programming

- High level abstraction
- Subgraph search tree

Work Division
- Simple API

Transparent parallelism
- Code in sequential, not in parallel

High Performance
Search Space Pruning

search space pruning is the key for GPM performance
Pattern-aware Search Space Pruning

Problem: 4-Clique Listing

Pattern

Input Graph

Level 1

Level 2

Level 3

Triangles are promising
Wedges are not

Problem: 4-Clique Listing
Exposing Interface for User Defined Pruning

Extend Stage

procedure Extend(SubgraphList in, SubgraphList out) {
  for each subgraph sg ∈ in
    for each vertex v ∈ sg
      if (toExtend(sg, v) == true)
        for each vertex u ∈ adj(v)
          if (toAdd(sg, u) == true)
            out.insert(sg ∪ u)
}

Two of the user defined functions*:

bool toExtend(Subgraph sg, Vertex v) {
  return true;
}

bool toAdd(Subgraph sg, Vertex u) {
  return true;
}

* The full list of API functions can be found in the Pangolin paper: Chen et. al, VLDB 2020
Easy Programming in Pangolin

- **k-Clique Listing**
  - Extend only the last vertex \( v \)
  - Check if new vertex \( u \) is connected to all previous vertices

```python
bool toExtend(Subgraph sg, Vertex v) {
    return (sg.getLastVertex() == v);
}

bool toAdd(Subgraph sg, Vertex u) {
    for v ∈ sg.getVertices() except last:
        if (!isConnected(v, u)) return false;
    return true;
}
```

<table>
<thead>
<tr>
<th>Hand Written</th>
<th>Pangolin</th>
</tr>
</thead>
<tbody>
<tr>
<td>State-of-the-art: (KClis: WWW’18) 395 Lines of Code !!!</td>
<td>8 Lines of code !!!</td>
</tr>
</tbody>
</table>
Graph Mining Challenges

Easy Programming

High Performance

Search space pruning

Architectural Optimizations
Unlocking GPU Horsepower for GPM (1/3)

Previous solution:
array of structures

Extend Stage

Input subgraph list

Output subgraph list

Output subgraph list

Extend

subgraph list is the key data structure

Uncoalesced memory accesses!!!

Not GPU friendly!
Unlocking GPU Horsepower for GPM (2/3)

Previous solution:
array of structures

```
<table>
<thead>
<tr>
<th>thread 1</th>
<th>thread 2</th>
<th>thread 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

subgraph list data
memory address 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09

Not GPU friendly!

Key idea: Re-structure data layout

Our solution:
structures of arrays

```
<table>
<thead>
<tr>
<th>t1</th>
<th>t2</th>
<th>t3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>
```

subgraph list data
memory address 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09

GPU friendly!

Coalesced memory accesses!!!

Previous solution: one by one ✗
Our solution: level by level ✓
Unlocking GPU Horsepower for GPM (3/3)

Coalesced memory accesses!!!

Real implementation: Build a prefix tree level by level

Prefix tree

Extend Stage

GPU friendly!
Graph Mining Challenges

Easy Programming

High Performance

- Search space pruning
- Architectural Optimizations
Outline

- Graph Pattern Mining (GPM) Use Cases
- Pangolin Overview
- Pangolin System Internals
  - How to Achieve Easy Programming?
  - How to Achieve High Performance?
- Evaluation
Comparing with Existing Systems on CPU

Geomean speedup of $49\times$, $88\times$, $80\times$ over Arabesque[1], RStream[2], Fractal[3]

Input graph: Patent-citations

TC: Triangle Counting
CL: Clique Listing
MC: Motif Counting
FSM: Frequent Subgraph Mining

Intel Xeon Gold 5120 CPU (2.2GHz) 2 sockets (14 cores each), 190GB memory, and 3TB SSD

Comparing with Hand-written Code on CPU

![Bar Chart]

**Lower is better!**

TC: Triangle Counting  
CL: Clique Listing  
MC: Motif Counting  
FSM: Frequent Subgraph Mining

Input graph: Patent-citations
Pangolin GPU Speedup over Pangolin CPU

- **Pangolin on V100 GPU gets 15× speedup over Pangolin CPU**

- **Speedup of Pangolin on GPU over Pangolin on 28-thread CPU**
  - TC: Triangle Counting
  - CL: Clique Listing
  - MC: Motif Counting
  - FSM: Frequent Subgraph Mining
  - Mi: Mico
  - Pa: Patent-citations
  - Yo: Youtube
Summary: GPM System Design

Easy Programming
- High level abstraction
- Work Division
- Transparent parallelism

Easy coding!

High Performance
- Search space pruning
- Architectural optimizations

100× ~ 1000× speedup!!!
Outline

- Graph Pattern Mining (GPM) Use Cases
- Pangolin Overview
- Pangolin System Internals
  - How to Achieve Easy Programming?
  - How to Achieve High Performance?
- Evaluation
- **Sandslash (optional)**
Sandslash*: A Two-level GPM System on CPU

- **Easier programming**
  - High-level: pattern specification
  - Low-level (*optional*): user defined pruning

- **Higher performance**
  - DFS exploration instead of BFS
  - High-level optimizations: automated
  - Low-level optimizations (*optional*)

* Sandslash: A Two-Level Framework for Efficient Graph Pattern Mining, to appear in ICS’21
Search Tree Exploration: BFS vs. DFS

- Pangolin (both CPU and GPU) uses BFS
  - Good for massive parallelism on GPU
  - Large memory footprint for intermediate data
  - Lots of redundant computation

- Sandslash (CPU-only) uses DFS
  - Small and confined memory footprint
  - Avoid redundancy by memoization

(a) BFS exploration

(b) DFS exploration
Automated High-level Optimizations

- triangle
- 4-cycle
- diamond

Sandslash Runtime

Pattern analysis → Apply high-level optimizations → Customized DFS search
User Defined Low-level Optimizations

Sandslash Runtime

Pattern analysis ➔ Automated high-level optimizations ➔ Customized DFS search

User specified low-level optimizations
Summary: GPM System on CPU

Easier Programming

- API separation
- Automated high-level optimizations

Higher Performance

- On CPU 8x and 14x speedup over Pangolin and AutoMine*
- Avoid redundancy with memoization
- Low-level user defined pruning

Easier coding!

* D Mawhirter et. al., AutoMine: Harmonizing High-Level Abstraction and High Performance for Graph Mining, SOSP’19
Thank You!

Q&A