X-Stream

Pranav Talluri
Background: Graph Processing Problems

- Weakly/Strongly Connected Components
- Single-Source Shortest Paths
- Minimum Cost Spanning Tree
- Maximal Independent Set
- Conductance
- PageRank
- Alternating Least Squares
- Bipartite Matching
- Clustering
- ...

- Social Networks
- Internet (Search)
- Telecommunications
- Recommendation Systems
- Transport Networks
- IoT
- Epidemiology
Background: Graph Processing Systems

Partitioning
- Edge-Cut
- Vertex-Cut
- Hybrid-Cut
- Stream-Based
- Distributed
- Dynamic

Unit
- Vertex-Centric
- Edge-Centric

Model
- Superstep
- Scatter-Gather
- Gather-Apply-Scatter (GAS)
- Frontier

Dynamism
- Static
- Temporal
- Streaming

Workload
- Local
- Global

Computing
- Single-Machine
- Distributed
Background: Pregel

Partitioning:
- Edge-Cut
- Vertex-Cut
- Hybrid-Cut
- Stream-Based
- Distributed
- Dynamic

Unit:
- Vertex-Centric
- Edge-Centric

Model:
- Superstep
- Scatter-Gather
- Gather-Apply-Scatter (GAS)
- Frontier

Dynamism:
- Static
- Temporal
- Streaming

Workload:
- Local
- Global

Computing:
- Single-Machine
- Distributed
Background: PowerGraph

- **Partitioning**: Edge-Cut, Vertex-Cut, Hybrid-Cut, Stream-Based, Distributed, Dynamic
- **Unit**: Vertex-Centric, Edge-Centric
- **Model**: Superstep, Scatter-Gather, Gather-Apply-Scatter (GAS), Frontier
- **Dynamism**: Static, Temporal, Streaming
- **Workload**: Local, Global
- **Computing**: Single-Machine, Distributed
Background: Problem

- Lack of solutions for single-machine graph processing
- Previous solutions do not take advantage of sequential bandwidth
- Pre-processing (e.g., sorting edge lists) is very expensive
  - Ligra requires pre-processing to produce an inverted edge list
  - Requires random access to a large data structure to switch the edges
    - Source to destination ----> destination to source
  - Dominates overall runtime
Background: X-Stream

Partitioning
- Edge-Cut
- Vertex-Cut
- Hybrid-Cut
- Stream-Based
- Distributed
- Dynamic

Unit
- Vertex-Centric
- Edge-Centric

Model
- Superstep
- Scatter-Gather
- Gather-Apply-Scatter (GAS)
- Frontier

Dynamism
- Static
- Temporal
- Streaming

Workload
- Local
- Global

Computing
- Single-Machine
- Distributed
Approach: Overview

• Single machine system
• System operates on an abstraction of slow and fast memory
• Data is divided into streaming partitions, which are streamed from slow to fast memory
• Scatter-Shuffle-Gather iterations applied to streaming partitions to implement various algorithms
Approach: Fast and Slow Memory

- Designed for two systems
  1. Graph fits in system memory (**in-memory**)
     - Main Memory -> On-Core Caches
  2. Graph fits in secondary storage (**out-of-core**)
     - SSD/HDD/Magnetic -> Main Memory

- Edge-centric approach identifies that storage offers greater sequential bandwidth than random access bandwidth
  - We stream unordered edges

- Trade-off between a few slower accesses, or many faster accesses
Approach: Streaming Partitions

• A streaming partition consists of:
  • A vertex set
  • An edge list
  • An update list

• The **vertex sets** are mutually disjoint so that their union forms the vertex set of the entire graph

• The **edge lists** contains the edges whose source is in the vertex set

• The **update list** contains all the updates whose destination is in the vertex set

• A fixed number of partitions are chosen based on various constraints
Approach: Edge-Centric Scatter

1. Read vertex set
2. Stream edge list
3. Produce stream of updates

scatter phase:
   for each streaming_partition p
       read in vertex set of p
       for each edge e in edge list of p
           edge_scatter(e): append update to Uout
Approach: Edge-Centric Shuffle

1. Rearrange updates into update list of streaming partition containing destination vertex

shuffle phase:
   for each update $u$ in $U_{out}$
     let $p = \text{partition containing target of } u$
     append $u$ to $U_{in}(p)$
   destroy $U_{out}$
Approach: Edge-Centric Gather

1. Read vertex set
2. Stream update list
3. Compute new vertex values

```
gather phase:
  for each streaming_partition p
    read in vertex set of p
    for each update u in Uin(p)
      edge_gather(u)
    destroy Uin(p)
```
Example

(a) Graph example
(b) Graph data processing
Approach: Out-of-Core Graphs

- We want to achieve sequential access for the shuffle phase as well
- We achieve this by merging the scatter and shuffle phases
- We then use an in-memory buffer to hold updates
  - This is shuffled in-memory when it becomes full
- We also have additional in-memory data structures to hold input from the disk, input/output for the shuffle, and output to the disk

```plaintext
merged scatter/shuffle phase:
    for each streaming partition s
        while edges left in s
            load next chunk of edges into input buffer
            for each edge e in memory
                edge_scatter(e) appending to output buffer
                if output buffer is full or no more edges
                in-memory shuffle output buffer
                for each streaming partition p
                    append chunk p to update file for p

gather phase:
    for each streaming_partition p
        read in vertex set of p
        while updates left in p
            load next chunk of updates into input buffer
            for each update u in input buffer
                edge_gather(u)
            write vertex set of p
```
Approach: In-Memory Graphs

- Utilise all cores to maximise bandwidth
  - In-Memory Streaming Buffer is sliced
  - Parallelisation as threads are executing different streaming partitions – they atomically reserve and fill space in the buffer

- Far more partitions
  - Multi-stage shuffler to deal with increased partition count, as more partitions in a single stage shuffler would lead to challenges in exploiting sequential bandwidth
  - Partitions are placed in a tree hierarchy
Performance

<table>
<thead>
<tr>
<th>Threads</th>
<th>Ligra (s)</th>
<th>X-Stream (s)</th>
<th>Ligra-pre (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BFS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>11.10</td>
<td>168.50</td>
<td>1250.00</td>
</tr>
<tr>
<td>2</td>
<td>5.59</td>
<td>86.97</td>
<td>647.00</td>
</tr>
<tr>
<td>4</td>
<td>2.83</td>
<td>45.12</td>
<td>352.00</td>
</tr>
<tr>
<td>8</td>
<td>1.48</td>
<td>26.68</td>
<td>209.40</td>
</tr>
<tr>
<td>16</td>
<td>0.85</td>
<td>18.48</td>
<td>157.20</td>
</tr>
<tr>
<td></td>
<td>Pagerank</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>990.20</td>
<td>455.06</td>
<td>1264.00</td>
</tr>
<tr>
<td>2</td>
<td>510.60</td>
<td>241.56</td>
<td>654.00</td>
</tr>
<tr>
<td>4</td>
<td>269.60</td>
<td>129.72</td>
<td>355.00</td>
</tr>
<tr>
<td>8</td>
<td>145.40</td>
<td>83.42</td>
<td>211.40</td>
</tr>
<tr>
<td>16</td>
<td>79.24</td>
<td>50.06</td>
<td>160.20</td>
</tr>
</tbody>
</table>

BFS on scale-free graph (32M vertices/256M edges)

Twitter graph, 16 threads
## Performance

<table>
<thead>
<tr>
<th>memory</th>
<th>WCC</th>
<th>SCC</th>
<th>SSSP</th>
<th>MCST</th>
<th>MIS</th>
<th>Cond.</th>
<th>SpMV</th>
<th>Pagerank</th>
<th>BP</th>
</tr>
</thead>
<tbody>
<tr>
<td>amazon0601</td>
<td>0.61s</td>
<td>1.12s</td>
<td>0.83s</td>
<td>0.37s</td>
<td>3.31s</td>
<td>0.07s</td>
<td>0.09s</td>
<td>0.25s</td>
<td>1.38s</td>
</tr>
<tr>
<td>cit-Patents</td>
<td>2.98s</td>
<td>0.69s</td>
<td>0.29s</td>
<td>2.35s</td>
<td>3.72s</td>
<td>0.19s</td>
<td>0.19s</td>
<td>0.74s</td>
<td>6.32s</td>
</tr>
<tr>
<td>soc-livejournal</td>
<td>7.22s</td>
<td>11.12s</td>
<td>9.60s</td>
<td>7.66s</td>
<td>15.54s</td>
<td>0.78s</td>
<td>0.74s</td>
<td>2.90s</td>
<td>1m 21s</td>
</tr>
<tr>
<td>dimacs-usa</td>
<td><strong>6m 12s</strong></td>
<td><strong>9m 54s</strong></td>
<td><strong>38m 32s</strong></td>
<td><strong>46.8s</strong></td>
<td><strong>9.60s</strong></td>
<td><strong>0.26s</strong></td>
<td><strong>0.65s</strong></td>
<td><strong>2.58s</strong></td>
<td><strong>12.01s</strong></td>
</tr>
</tbody>
</table>

| disk | Friendster | 38m | 38s | 44m | 3s | 1h | 56m | 58s | 32m | 25s | 19m | 13s | 1h | 16m | 29s | 1m | 40s | 3m | 41s | 15m | 31s | 52m | 24s |
|      | sk-2005   | 19m | 19s |     |    | 1h | 51m | 35s | 35m | 23s | 19m | 10s | 1h | 16m | 29s | 1m | 14s | 1m | 29s | 8m | 9s | 56m | 29s |
|      | Twitter   |     |    |     |    | 1h | 32m | 32s |     |    | 1h | 32m | 32s |     |    |     |    |    | 6m | 12s | 42m | 52s |

| disk | Friendster | 1h | 17m | 18s | 2h | 29m | 39s | 4h | 41m | 9s | 1h | 32m | 9s | 1h | 41m | 44s | 39m | 12s | 4m | 45s | 3m | 13s | 13m | 21s | 2h | 8m | 13s |
|      | sk-2005   | 1h | 30m | 3s | 2h | 39m | 26s | 4h | 41m | 26s | 1h | 30m | 9s |     |    |     |    |     |    |    | 1h | 21m | 14s | 8h | 2m | 58s |
|      | Twitter   |     |    |     |    | 1h | 10m | 12s |     |    | 1h | 42m | 14s | 1h | 42m | 14s | 1m | 38s | 1m | 38s | 14m | 40s | 1h | 21m | 14s |
|      | yahoo-web |     |    |     |    |     |    |     |    |     |    |     |    |     |    |     |    |    |     |    |     |    |    |     |     |

<table>
<thead>
<tr>
<th>memory</th>
<th># iters</th>
<th>ratio</th>
<th>wasted</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>amazon0601</td>
<td>19</td>
<td>2.58s</td>
<td>63</td>
<td></td>
</tr>
<tr>
<td>cit-Patents</td>
<td>21</td>
<td>2.20s</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>soc-livejournal</td>
<td>13</td>
<td>2.13s</td>
<td>57</td>
<td></td>
</tr>
<tr>
<td>dimacs-usa</td>
<td><strong>6263</strong></td>
<td><strong>1.94</strong></td>
<td><strong>98</strong></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Graph</th>
<th># steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>In-memory</td>
<td></td>
</tr>
<tr>
<td>amazon0601</td>
<td>19</td>
</tr>
<tr>
<td>cit-Patents</td>
<td>20</td>
</tr>
<tr>
<td>soc-livejournal</td>
<td>15</td>
</tr>
<tr>
<td>dimacs-usa</td>
<td><strong>8122</strong></td>
</tr>
<tr>
<td>Out-of-core</td>
<td></td>
</tr>
<tr>
<td>sk-2005</td>
<td>28</td>
</tr>
<tr>
<td>yahoo-web</td>
<td><strong>over 155</strong></td>
</tr>
</tbody>
</table>
Evaluation: Positives

• Single-machine lowers barrier to entry
  • Compared to large-scale distributed systems, at the cost of size capacity
• Exploits sequential bandwidth
• Flexible framework for in-memory and cache
• Performant for certain applications
• Removes need for sorting
  • Major consideration for other systems
Evaluation: Negatives

- The Scatter-Gather programming model can be restrictive
  - X-Stream supports more, but the paper does not go into detail
- Parallelism for in-memory graphs is dependent on effective work distribution
  - X-Stream supports work stealing, but the paper does not go into detail
- Some issues in evaluation
  - Comparisons against other systems is very limited and seems cherry-picked
  - Especially inadequate in comparing to well-established distributed systems, which would ground results
- There is no discussion of fault tolerance, which suggests it was not a major consideration
  - Makes the system unsuitable for many applications
- Does choosing sequential over random access have power consumption implications?
Thank You