CGraph: A Correlations-aware Approach for Efficient Concurrent Iterative Graph Processing

Yu Zhang\textsuperscript{1}, Xiaofei Liao\textsuperscript{1}, Hai Jin\textsuperscript{1}, Lin Gu\textsuperscript{1}, Ligang He\textsuperscript{2}, Bingsheng He\textsuperscript{3}, Haikun Liu\textsuperscript{1}

\textsuperscript{1} SCTS/CGCL, Huazhong University of Science and Technology, China
\textsuperscript{2} University of Warwick, UK
\textsuperscript{3} National University of Singapore, Singapore
Part 1

Background and Challenges
Many concurrent graph processing jobs are daily executed over the same graph (or its different snapshots) to provide various information for different products.
What is CGP Job

PageRank  k-means  SSSP

Graph Data

Platform

Shared

Concurrent graph processing jobs
What is CGP Job

(a) Number of the CGP jobs

(b) Ratio of shared graph data

The information traced over a large social network
What is CGP Job

More than 20 CGP jobs to concurrently analyze the same graph at the peak time.

The information traced over a large social network.
What is CGP Job

(a) Number of the CGP jobs

(b) Ratio of shared graph data

The information traced over a large social network
Challenges: Data Access Problems in the CGP Jobs

The average execution time of each job is significantly prolonged as the number of jobs increases due to higher data access cost.
Challenges: An Example

➢ The CGP jobs access the shared graph partitions in an individual manner along different graph paths
➢ The processing time of each partition is various for different jobs

Reason: The CGP jobs contend for data access channel, memory and cache
Motivations

Observations:
- Spatial correlation
- Temporal correlation
Motivations

Observations:

- **Spatial correlation**: The intersections of the set of graph partitions to be handled by different CGP jobs in each iteration are large (more than 75% of all active partitions on average).

- **Temporal correlation**
Motivations

Observations:

- **Spatial correlation**: The intersections of the set of graph partitions to be handled by different CGP jobs in each iteration are large (more than 75% of all active partitions on average).

- **Temporal correlation**: Some graph partitions may be accessed by multiple CGP jobs (may be more than 16 jobs) within a short time duration.
Motivations

Observations:

- **Spatial correlation**: The intersections of the set of graph partitions to be handled by different CGP jobs in each iteration are large (more than 75% of all active partitions on average).

- **Temporal correlation**: Some graph partitions may be accessed by multiple CGP jobs (may be more than 16 jobs) within a short time duration.

Develop a solution for efficient use of cache/memory and the data access channel to achieve a higher throughput by fully exploiting the spatial/temporal correlations.
Motivations: An Example

➢ Spatial Correlations
  • Load the shared partitions for the related jobs along a common order to provide opportunity to consolidate the accesses to the shared graph structure and store a single copy of the shared data in the cache to serve multiple CGP jobs at the same time.

➢ Temporal Correlations
Motivations: An Example

- **Spatial Correlations**
  - Load the shared partitions for the related jobs along a common order to provide opportunity to consolidate the accesses to the shared graph structure and store a single copy of the shared data in the cache to serve multiple CGP jobs at the same time.

- **Temporal Correlations**
  - Take into account the temporal correlations, e.g., the usage frequency of the graph partitions, when loading them into the cache.

![Diagram](image)
Part 2

Related Work
Existing Graph Processing Systems

Single graph processing

GraphChi  X-Stream  GridGraph  NXgraph  CLIP  ...
Existing Graph Processing Systems

Single graph processing

- Higher sequential memory bandwidth
- Better data locality
- Less redundant data accesses
- Less memory consumption

Mainly focus on single graph processing job
Existing Graph Processing Systems

- **Single graph processing**
  - GraphChi
  - X-Stream
  - GridGraph
  - NXgraph
  - CLIP
  - **Mainly focus on single graph processing job**
  - Higher sequential memory bandwidth
  - Better data locality
  - Less redundant data accesses
  - Less memory consumption

- **Concurrent graph processing**
Part 3

Our Approach:
A Correlations-aware Data-centric Execution Model
Main Goals

Minimize the redundant accessing and storing cost of the shared graph structure data (occupies more than 70% of the total memory of each job) by fully exploiting the spatial/temporal correlations between the CGP jobs.
Data-centric LTP Execution Model

- Traditional approach:

\[ D_1 = (V_1, S_1, E_1, W_1) \]
\[ D_2 = (V_2, S_2, E_2, W_2) \]
\[ \vdots \]
\[ D_J = (V_J, S_J, E_J, W_J) \]

Most graph structure data \( G=(V, E, W) \) is the same for different CGP jobs.
Data-centric LTP Execution Model

➢ Traditional approach:

\[ D_1 = (V_1, S_1, E_1, W_1) \]
\[ D_2 = (V_2, S_2, E_2, W_2) \]
\[ \vdots \]
\[ D_J = (V_J, S_J, E_J, W_J) \]

Most graph structure data \( G=(V, E, W) \) is the same for different CGP jobs.

➢ Load-Trigger-Pushing (denoted by LTP) model:

\[ D = (V, S, E, W) \]
\[ G = (V, E, W), \text{ where } G = \bigcup_i G^i \]
\[ S_1, S_2, \ldots, S_J \]
Data-centric LTP Execution Model

- Load-Trigger-Pushing (denoted by LTP) model:
  - Graph Loading:
    \[ L^i \leftarrow L(G^i, \bigcup_{j \in J} S^i_j) \]

```
Global Space (Storing the shared graph structure data)

Memory/Disk

Load of graph structure data
```
Data-centric LTP Execution Model

- **Load-Trigger-Pushing (denoted by LTP) model:**
  - **Graph Loading:**
    \[ L^i \leftarrow L(G^i, \bigcup_{j \in J} S^i_j) \]
  - **Trigger and Parallel Execution:**
    \[ S_{\text{new}}^i \leftarrow \bigcup_{j \in J} T_j(G^i, S^i_j) \]
Load-Trigger-Pushing (denoted by LTP) model:

- **Graph Loading:**
  \[ L^i \leftarrow L(G^i, \bigcup_{j \in J} S^i_j) \]

- **Trigger and Parallel Execution:**
  \[ \tilde{S}^{i_{\text{new}}} \leftarrow \bigcup_{j \in J} T_j(G^i, S^i_j) \]

- **State Pushing:**
  \[ S^j_{\text{new}} = \bigcup_i S^{i_{\text{new}}} \]
Illustration of Our LTP Model

**PageRank job**

- `IsNotConvergent(vh)`: 
  ```
  return |vh.Δvalue| > ε
  ```

- `Acc(value1, value2)`:
  ```
  return value1 + value2
  ```

- `Compute(Gi, vh)`:// Processing of each vertex
  ```
  vh.value = Acc(vh.value, vh.Δvalue)
  <links> = look up outlinks of vh from Gi
  for (each link <vh, ve> ∈ <links>){
    Δvalue = Δx vh.Δvalue/Gi[vh].OutDegree
    ve.Δvalue = Acc(ve.Δvalue, Δvalue)
  }
  ```

**SSSP job**

- `IsNotConvergent(vh)`:
  ```
  return |vh.Δvalue| ≤ 0
  ```

- `Acc(value1, value2)`:
  ```
  return min(value1, value2)
  ```

- `Compute(Gi, vh)`:// Processing of each vertex
  ```
  vh.value = Acc(vh.value, vh.Δvalue)
  <links> = look up outlinks of vh from Gi
  for (each link <vh, ve> ∈ <links>){
    Δvalue = Δx vh.Δvalue+<vh, ve> distance
    ve.Δvalue = Acc(ve.Δvalue, Δvalue)
  }
  ```

**Global Space**

- **Vertex ID**
  - `v1`: 0.2
  - `v2`: 0.1
  - `v3`: 0.25

**Cache**

- **Vertex ID**
  - `v1`: 1.2
  - `v2`: 0
  - `v3`: 2.9

**Memory/Disk**

- **Partition 1**
  - `v1` (1.2)
  - `v2` (2.9)
  - `v3` (2.9)

- **Partition 2**
  - `v4` (0.9)
  - `v5` (1.5)

**Scheduler (Arranging the Loading order of graph structure partitions)**
Implementations: Graph Storage for Multiple CGP Jobs

**PageRank Job**

<table>
<thead>
<tr>
<th>Vertex ID</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>v1</td>
<td>0.2</td>
</tr>
<tr>
<td>v2</td>
<td>0.1</td>
</tr>
<tr>
<td>v3</td>
<td>0.25</td>
</tr>
</tbody>
</table>

**SSSP Job**

<table>
<thead>
<tr>
<th>Vertex ID</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>v1</td>
<td>1.2</td>
</tr>
<tr>
<td>v2</td>
<td>0</td>
</tr>
<tr>
<td>v3</td>
<td>2.9</td>
</tr>
</tbody>
</table>

**Graph Structure Partitions**

<table>
<thead>
<tr>
<th>Vertex ID</th>
<th>Edge List</th>
<th>Flag</th>
<th>Master Location</th>
<th>Information Associated with Its Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>v3</td>
<td>v5</td>
<td>Mirror</td>
<td>Partition 1</td>
<td>1.5</td>
</tr>
<tr>
<td>v4</td>
<td>v3, v5</td>
<td>Master</td>
<td>Partition 2</td>
<td>0.9, 2.5</td>
</tr>
<tr>
<td>v5</td>
<td>Ø</td>
<td>Master</td>
<td>Partition 2</td>
<td>Ø</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Vertex ID</th>
<th>Edge List</th>
<th>Flag</th>
<th>Master Location</th>
<th>Information Associated with Its Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>v1</td>
<td>v3</td>
<td>Master</td>
<td>Partition 1</td>
<td>1.1</td>
</tr>
<tr>
<td>v2</td>
<td>v1, v3</td>
<td>Master</td>
<td>Partition 1</td>
<td>1.2, 2.9</td>
</tr>
<tr>
<td>v3</td>
<td>Ø</td>
<td>Master</td>
<td>Partition 1</td>
<td>Ø</td>
</tr>
</tbody>
</table>

**Private Table Partitions**
Implementations: Details to Store Evolving Graph Structure
## Implementations: Load of Partitions

<table>
<thead>
<tr>
<th>(a) There is only one job J1</th>
<th>(b) J2 has been submitted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partition 1</td>
<td>J1</td>
</tr>
<tr>
<td></td>
<td>J1</td>
</tr>
<tr>
<td></td>
<td>J1</td>
</tr>
<tr>
<td></td>
<td>J1</td>
</tr>
<tr>
<td></td>
<td>J1</td>
</tr>
</tbody>
</table>

| Partition 1                | J1                        |
| Partition 2                | J1                        |
| Partition 3                | J1                        |
| Partition 4                | J1                        |

| Partition 1                | J1                        |
| Partition 2                | J1                        |
| Partition 3                | J2                        |
| Partition 4                | J1                        |

| Partition 1                | J1                        |
| Partition 2                | J1                        |
| Partition 3                | J1                        |
| Partition 4                | J1                        |

<table>
<thead>
<tr>
<th>(c) J3 has been submitted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partition 1</td>
</tr>
<tr>
<td>Partition 2</td>
</tr>
<tr>
<td>Partition 3</td>
</tr>
<tr>
<td>Partition 4</td>
</tr>
</tbody>
</table>

| Partition 1                | J1                        |
| Partition 2                | J1                        |
| Partition 3                | J2                        |
| Partition 4                | J3                        |

| Partition 1                | J1                        |
| Partition 2                | J1                        |
| Partition 3                | J1                        |
| Partition 4                | J1                        |
Implementations: Load of Partitions

A core-subgraph based scheduling algorithm can be used to maximize the utilization ratio of each partition loaded into the cache.
Implementations: Parallel Processing of Graph Partition
Implementations: Parallel Processing of Graph Partition
Implementations: Vertex State Synchronization

Synchronization from **Mirrors** to **Master**

**Non-optimized:**
- P1:v3→P2:v3
- P1:v6→P3:v6
- P1:v4→P2:v4
  - ...

**Optimized:**
- P1:v3→P2:v3
- P1:v4→P2:v4
  - ...

Synchronization from **Master** to **Mirrors**

**Non-optimized:**
- P2:v3→P1:v3
- P2:v3→P3:v3
- P2:v4→P1:v4
- P2:v4→P3:v4
  - ...

**Optimized:**
- P2:v3→P1:v3
- P2:v4→P1:v4
- P2:v3→P3:v3
- P2:v4→P3:v4
  - ...
Part 4

Performance Evaluation
Evaluation

Experimental setup

- **Machine information**
  - CPU: 4-way 8-core Intel Xeon CPU E5-2670; each CPU has 20 MB LLC
  - Main Memory: 64 GB

- **Typical graph algorithms**
  - PageRank, SSSP, SCC, BFS

- **Data sets**

<table>
<thead>
<tr>
<th>Data sets</th>
<th>Vertices</th>
<th>Edges</th>
<th>Sizes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Twitter</td>
<td>41.7 M</td>
<td>1.4 B</td>
<td>17.5 GB</td>
</tr>
<tr>
<td>Friendster</td>
<td>65 M</td>
<td>1.8 B</td>
<td>22.7 GB</td>
</tr>
<tr>
<td>uk2007</td>
<td>105.9 M</td>
<td>3.7 B</td>
<td>46.2 GB</td>
</tr>
<tr>
<td>uk-union</td>
<td>133.6 M</td>
<td>5.5 B</td>
<td>68.3 GB</td>
</tr>
<tr>
<td>hyperlink14</td>
<td>1.7 B</td>
<td>64.4 B</td>
<td>480.0 GB</td>
</tr>
</tbody>
</table>
Evaluation

Total execution time for the four jobs with different solutions

Execution time breakdown of different jobs on hyperlink14
Evaluation

Volume of data swapped into the cache for the four jobs

I/O overhead for the four jobs with different solutions
Evaluation

- **Graph 1**: Normalized execution time for the four jobs with and without our scheduler.
- **Graph 2**: Ratio of spared accessed data on the hyperlink14 data set.

Data sets include: Twitter, Friendster, uk2007, and uk-union.

Number of jobs: 1, 2, 4, 8.
Part 5

Conclusions
Conclusions

➢ **What CGraph brings in graph processing**
   - Analysis of temporal/spatial correlations in concurrent graph processing
   - A novel data-centric LTP model for concurrent graph processing
   - A core-subgraph based scheduling scheme

➢ **Future work**
   - How to further optimize the approach for evolving graph analysis
   - How to ensure QoS for some real-time CGP jobs
   - How to extend it to a distributed platform and also heterogeneous platform consisting of GPU, FPGA and even ASIC for higher throughput.
Thanks!