Gluon: A Communication-Optimizing Substrate for Distributed Heterogeneous Graph Analytics

PLDI 2018

Roshan Dathathri
Hoang-Vu Dang
Marc Snir

Gurbinder Gill
Alex Brooks

Loc Hoang
Nikoli Dryden
Keshav Pingali

TEXAS
The University of Texas at Austin

ILLINOIS
What is Gluon?

- Substrate: enable single address space applications to run on distributed, heterogeneous clusters

- Provides:
  - Partitioner
  - High-level synchronization API
  - Communication-optimizing runtime
How to use Gluon?

- **Programmers:**
  - Write shared-memory applications
  - Interface with Gluon using API

- **Gluon transparently handles:**
  - Graph partitioning
  - Communication and synchronization

Galois [SoSP’13]
Ligra [PPOPP’13]
IrGL [OOPSLA’16]
LCI [IPDPS’18]
How to use Gluon?

- **Programmers:**
  - Write shared-memory applications
  - Interface with Gluon using API

- **Gluon transparently handles:**
  - Graph partitioning
  - Communication and synchronization

Galois [SoSP’13]
Ligra [PPoPP’13]
IrGL [OOPSLA’16]
LCI [IPDPS’18]
Outline

- Gluon Synchronization Approach
- Optimizing Communication
  - Exploiting Structural Invariants of Partitions
  - Exploiting Temporal Invariance of Partitions
- Experimental Results
Gluon Synchronization Approach
Vertex Programming Model

- Every node has a label
  - e.g., distance in single source shortest path (SSSP)

- Apply an operator on an active node in the graph
  - e.g., relaxation operator in SSSP

- Operator: computes labels on nodes
  - Push-style: reads its label and writes to neighbors’ labels
  - Pull-style: reads neighbors’ labels and writes to its label

- Applications: breadth first search, connected component, pagerank, single source shortest path, betweenness centrality, k-core, etc.
Partitioning

Original graph

Partitions of the graph

Host h1

Host h2
Partitioning

Original graph

Partitions of the graph

Host h1

Host h2

Each edge is assigned to a unique host
Partitioning

- Each edge is assigned to a unique host
- All edges connect proxy nodes on the same host
Partitioning

- Each edge is assigned to a unique host
- All edges connect proxy nodes on the same host
- A node can have multiple proxies: one is master proxy; rest are mirror proxies

Original graph

Partitions of the graph

Host h1

Host h2

: Master proxy

: Mirror proxy
Partitioning

● Each edge is assigned to a unique host
● All edges connect proxy nodes on the same host
● A node can have multiple proxies: one is master proxy; rest are mirror proxies
How to synchronize the proxies?

- Distributed Shared Memory (DSM) protocols
  - Proxies act like cached copies
  - Difficult to scale out to distributed and heterogeneous clusters

![Diagram showing network topology with master and mirror proxies]
How does Gluon synchronize the proxies?

- Exploit domain knowledge
  - Cached copies can be stale as long as they are eventually synchronized

![Diagram showing synchronization between hosts h1 and h2 with nodes A to J and labels 0 to ∞ representing distance (label) from source A. Red nodes represent Master proxies, blue nodes Mirror proxies, and brown nodes are intermediate nodes with their respective labels.](image)
How does Gluon synchronize the proxies?

- Exploit domain knowledge
  - Cached copies can be stale as long as they are eventually synchronized

- Use all-reduce:
  - Reduce from mirror proxies to master proxy
  - Broadcast from master proxy to mirror proxies
When to synchronize proxies?

Host h1
- Read partition-1 of the graph
  - Distribution-agnostic computation on local proxies
    - Synchronize proxies with each other

Host h2
- Read partition-2 of the graph
  - Distribution-agnostic computation on local proxies
    - Synchronize proxies with each other
Gluon Distributed Execution Model

Host h1
- Gluon partitioner
  - Galois/Ligra on multicore CPU or IrGL/CUDA on GPU
    - Gluon comm. runtime
    - MPI/LCI

Host h2
- Gluon partitioner
  - Galois/Ligra on multicore CPU or IrGL/CUDA on GPU
    - MPI/LCI
    - Gluon comm. runtime

Galois [SoSP’13]
Ligra [PPoPP’13]
IrGL [OOPSLA’16]
LCI [IPDPS’18]
Gluon Synchronization API

- **Application-specific:**
  - **What:** Field to synchronize
  - **When:** Point of synchronization
  - **How:** Reduction operator to use

- **Platform-specific:**
  - Access functions for fields (specific to data layout)
Exploiting Structural Invariants to Optimize Communication
Structural invariants in the partitioning:

- Mirror proxies do not have outgoing edges

As a consequence, for sssp:
- Mirror proxies do not read their distance label
- Broadcast from master proxy to mirror proxies is not required
Partitioning: strategies, constraints, invariants

- Algorithm invariant in SSSP:

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Constraints and Invariants</th>
<th>SSSP: Invariants</th>
<th>SSSP: Sync</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outgoing Edge-Cut (OEC)</td>
<td>Mirrors: no outgoing edges</td>
<td>Mirrors: label <em>not read</em></td>
<td>Reduce</td>
</tr>
<tr>
<td>Incoming Edge-Cut (IEC)</td>
<td>Mirrors: no incoming edges</td>
<td>Mirrors: label <em>not written</em></td>
<td>Broadcast</td>
</tr>
<tr>
<td>Cartesian Vertex-Cut (CVC)</td>
<td>Mirrors: either no outgoing edges or no incoming edges</td>
<td>Mirrors: either label <em>not read</em> or <em>label not written</em></td>
<td>Reduce-partial &amp; Broadcast-partial</td>
</tr>
<tr>
<td>Unconstrained Vertex-Cut (UVC)</td>
<td>None</td>
<td>None</td>
<td>Reduce &amp; Broadcast</td>
</tr>
</tbody>
</table>
Exploiting Temporal Invariance to Optimize Communication
Bulk-communication

- Proxies of millions of nodes need to be synchronized in a round
  - Not every node is updated in every round
- Address spaces (local-IDs) of different hosts are different
- Existing systems: use address translation and communicate global-IDs along with updated values
Bulk-communication in existing systems

Host h1

A-J: Global IDs
0-7: Local IDs
∞: distance (label) from source A

Host h2

∞: Master proxy
∞: Mirror proxy

Host h1

Host h2

5 6
2 2
2 2

5 6
2 2
2 2

2 4
2 2

2 4
2 2

:: Global IDs
:: Local IDs
:: Label
:: Address translation
:: Communication
:: Reduction
**Bulk-communication in Gluon**

- Elides address translation during communication in each round
- Exploits temporal invariance in partitioning
  - Mirrors and masters are static
  - e.g., only labels of C, G, and J can be reduced from h1 to h2
- Memoize address translation after partitioning
Experimental Results
Experimental setup

**Systems:**
- D-Ligra (Gluon + Ligra)
- D-Galois (Gluon + Galois)
- D-IrGL (Gluon + IrGL)
- Gemini (state-of-the-art)

**Benchmarks:**
- Breadth first search (bfs)
- Connected components (cc)
- Pagerank (pr)
- Single source shortest path (sssp)

<table>
<thead>
<tr>
<th>Inputs</th>
<th>rmat28</th>
<th>kron30</th>
<th>clueweb12</th>
<th>wdc12</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Size (CSR)</td>
<td>35GB</td>
<td>136GB</td>
<td>325GB</td>
<td>986GB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Clusters</th>
<th>Stampede (CPU)</th>
<th>Bridges (GPU)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Max. hosts</td>
<td>256</td>
</tr>
<tr>
<td></td>
<td>Machine</td>
<td>Intel Xeon Phi KNL</td>
</tr>
<tr>
<td></td>
<td>Each host</td>
<td>272 threads of KNL</td>
</tr>
<tr>
<td></td>
<td>Memory</td>
<td>96GB DDR3</td>
</tr>
</tbody>
</table>
Strong scaling on Stampede
(68 cores on each host)
Strong scaling on Bridges (4 GPUs share a physical node)

D-IrGL scales well
Impact of Gluon’s communication optimizations

D-Galois on 128 hosts of Stampede: clueweb12 with CVC

Improvement (geometric mean):

Communication volume: 2x
Execution time: 2.6x
Fastest execution time (sec) of all systems using best-performing number of hosts/GPUs

D-Galois and D-IrGL are faster than Gemini by factors of 3.9x and 4.9x
Conclusions

- Novel approach to build distributed, heterogeneous graph analytics systems: scales out to 256 multicore-Cpus and 64 GPUs
- Novel communication optimizations: improve execution time by 2.6x
- [EuroPar’18] Abelian compiler: shared-memory Galois apps ---> distributed, heterogeneous (D-Galois + D-IrGL) apps
- Gluon, D-Galois, and D-IrGL: publicly available in Galois v4.0
  [http://iss.ices.utexas.edu/?p=projects/galois](http://iss.ices.utexas.edu/?p=projects/galois)
- Use Gluon to scale out your shared-memory graph analytical framework