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

Roshan Dathathri    Gurbinder Gill    Loc Hoang
Hoang-Vu Dang    Alex Brooks    Nikoli Dryden
Marc Snir    Keshav Pingali

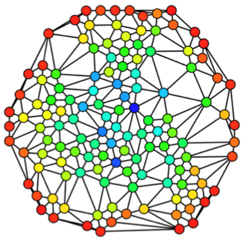
TEXAS
The University of Texas at Austin

ILLINOIS

# Distributed Graph Analytics

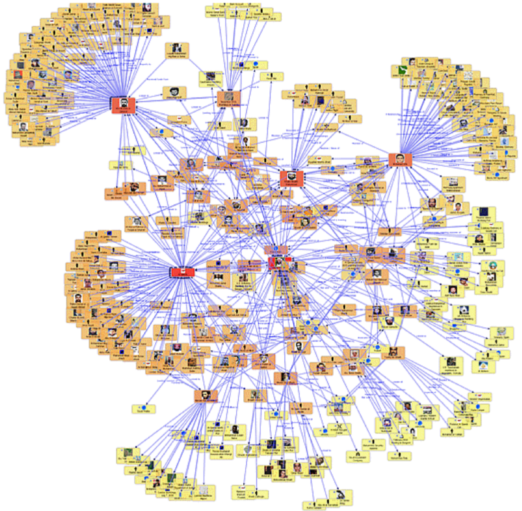
**Applications:**
machine learning and network science





Credits: Wikipedia, SFL Scientific, MakeUseOf

**Datasets:** unstructured graphs



Need TBs of memory

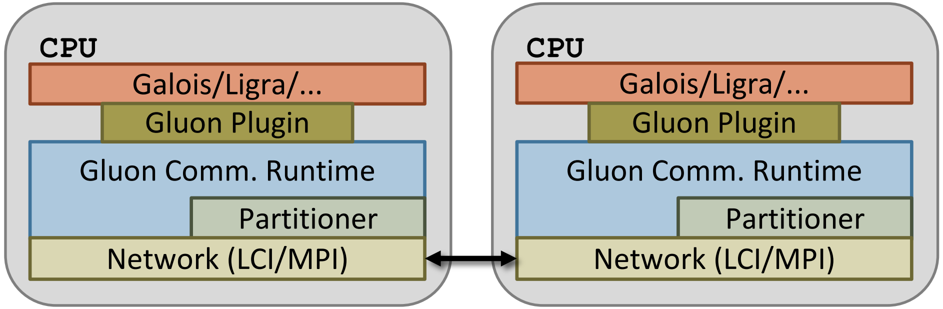
Credits: Sentinel Visualizer

# Gluon [PLDI'18]

- Substrate: single address space applications on distributed, heterogeneous clusters
- Provides:
  - Partitioner
  - High-level synchronization API
  - Communication-optimizing runtime

# How to use Gluon?

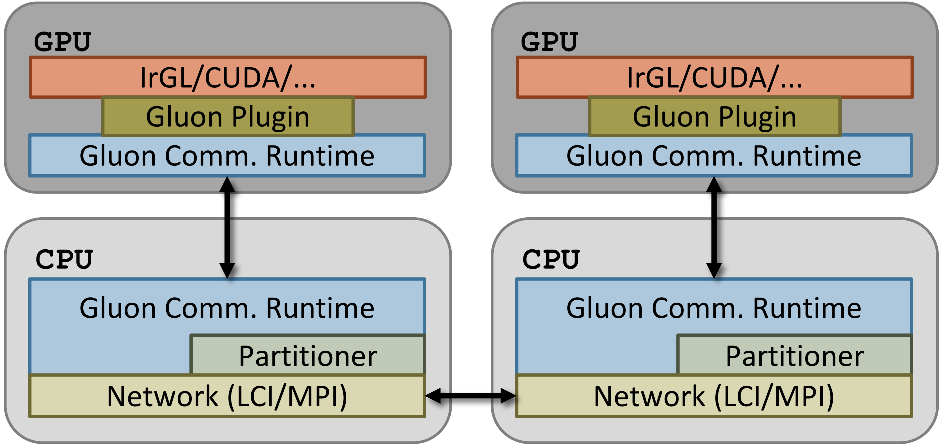
- Programmers:
  - Write shared-memory applications
  - Interface with Gluon using API
- Gluon transparently handles:
  - Graph partitioning
  - Communication and synchronization




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

# How to use Gluon?

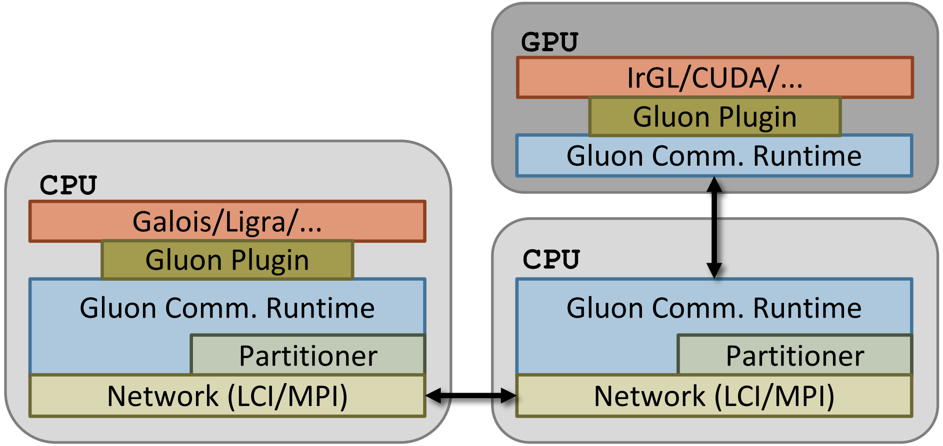
- Programmers:
  - Write shared-memory applications
  - Interface with Gluon using API
- Gluon transparently handles:
  - Graph partitioning
  - Communication and synchronization




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

# How to use Gluon?

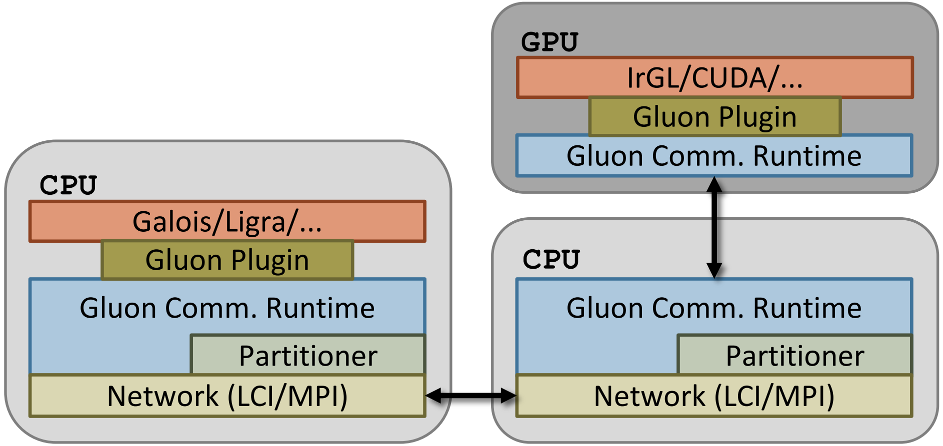
- Programmers:
  - Write shared-memory applications
  - Interface with Gluon using API
- Gluon transparently handles:
  - Graph partitioning
  - Communication and synchronization




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

# Contributions

- Novel approach to build distributed and heterogeneous graph analytics systems out of plug-and-play components
- Novel optimizations that reduce communication volume and time
- Plug-and-play systems built with Gluon outperform the state-of-the-art




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

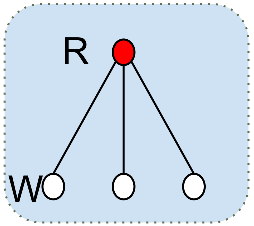
# Outline

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

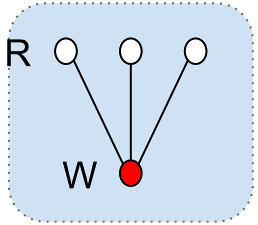
# Gluon Synchronization Approach

# Vertex Programming Model

- Every node has one or more labels
    - 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.

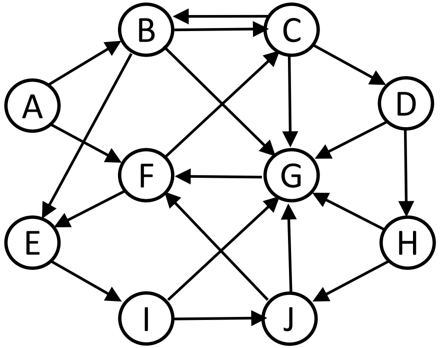



push-style




pull-style

# Partitioning



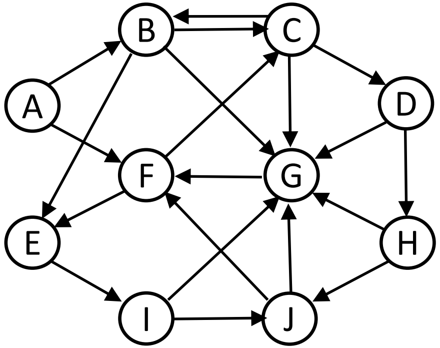

Original graph

Host h1

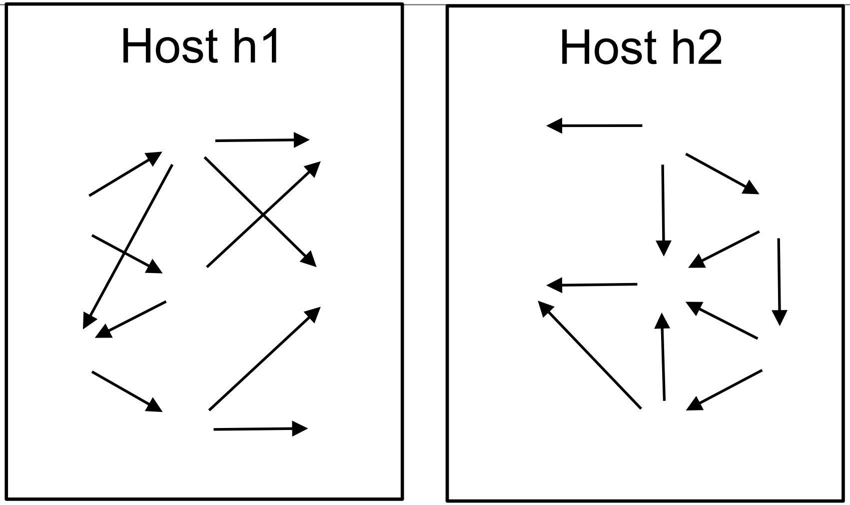
Host h2

Partitions of the graph

# Partitioning



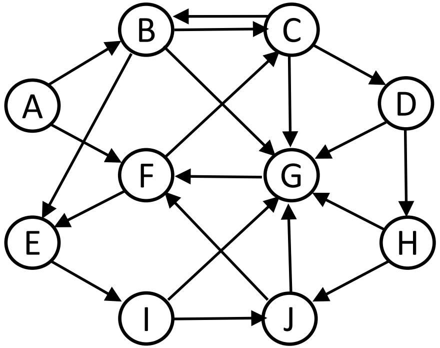

Original graph



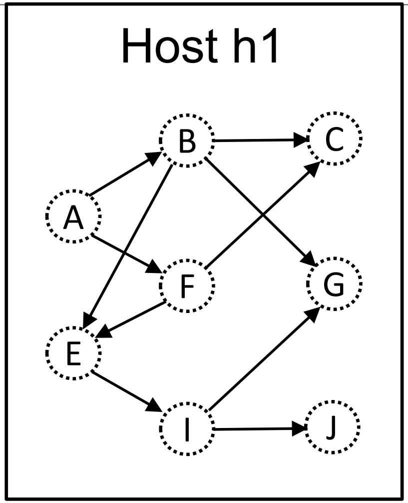

Partitions of the graph

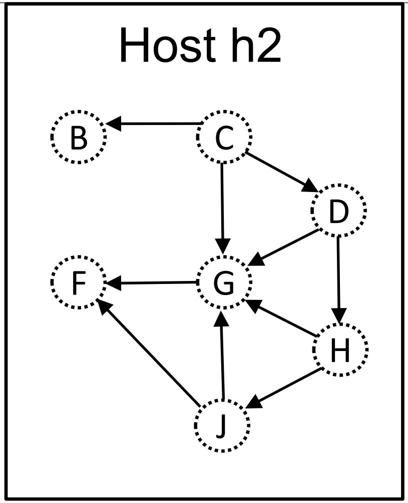
- Each edge is assigned to a unique host

# Partitioning




Original graph

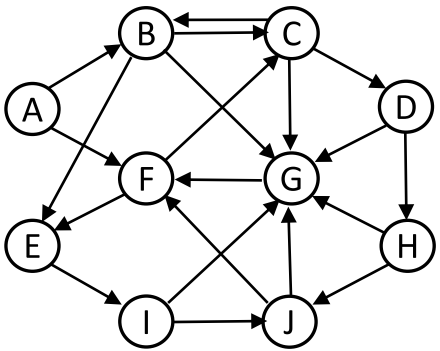


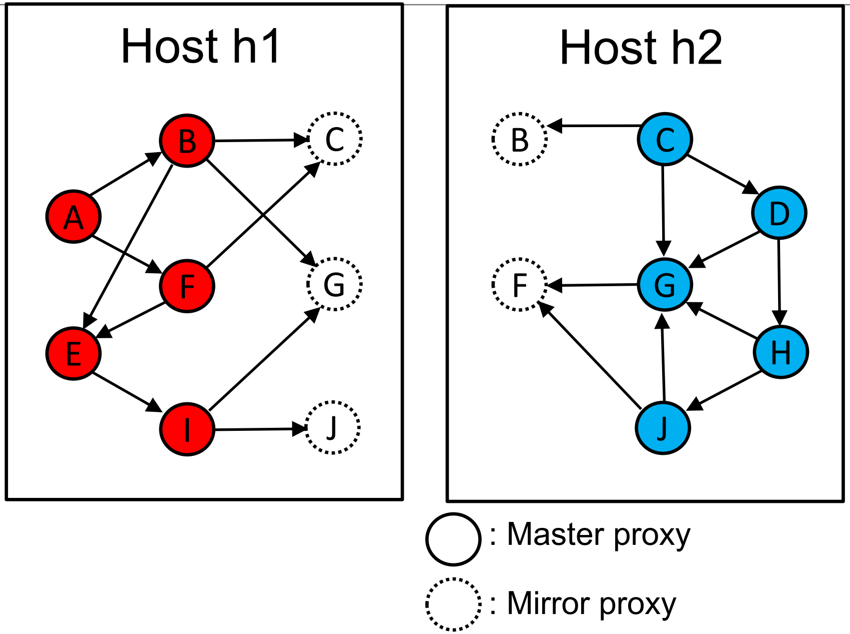

Partitions of the graph

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

# Partitioning



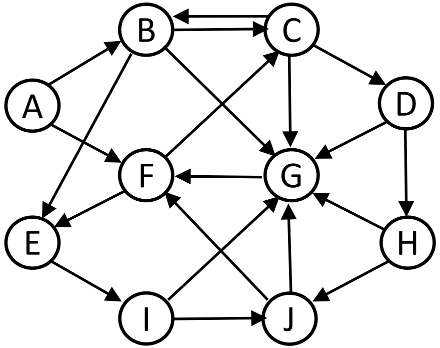

Original graph



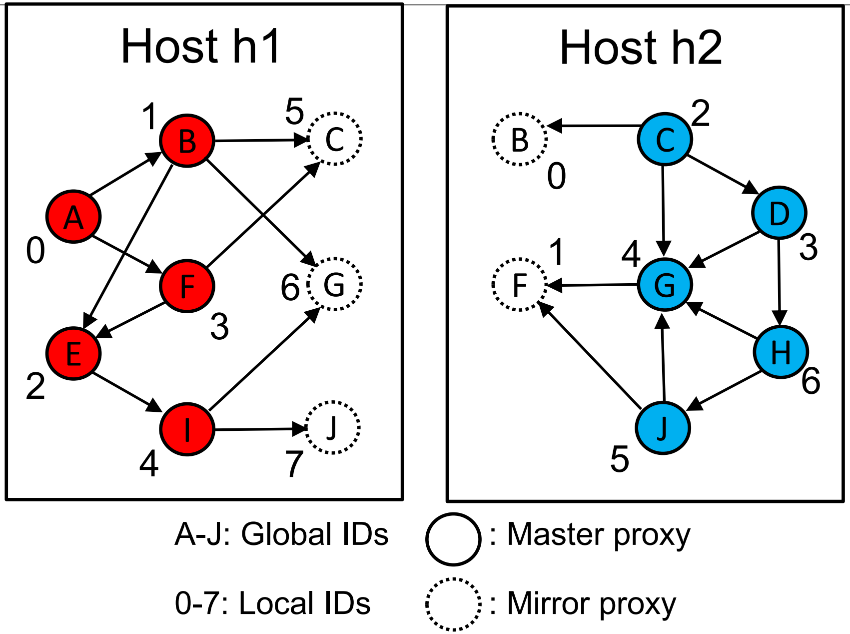

Partitions of the graph

- 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

# Partitioning




Original graph

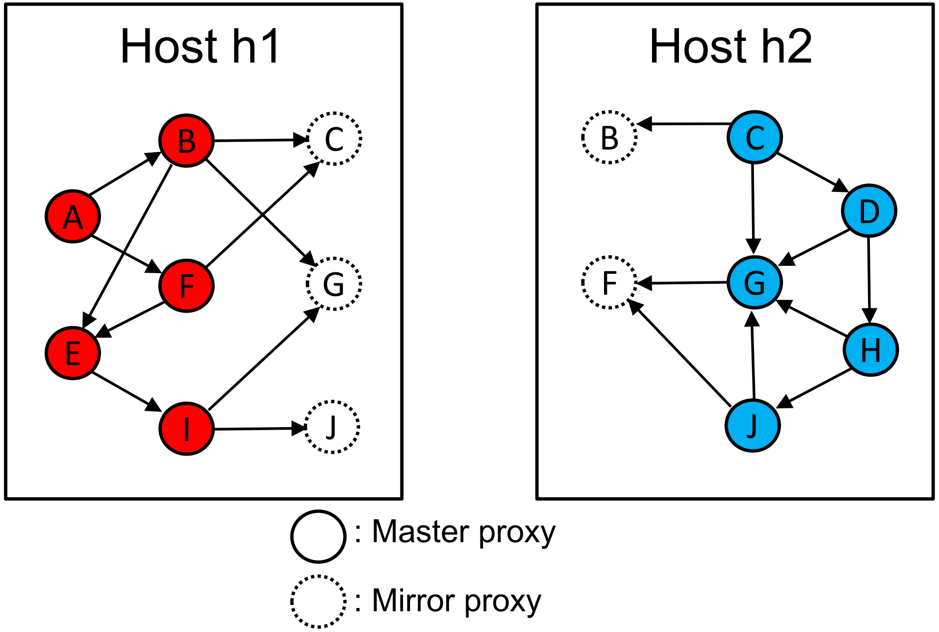



Partitions of the graph

- 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

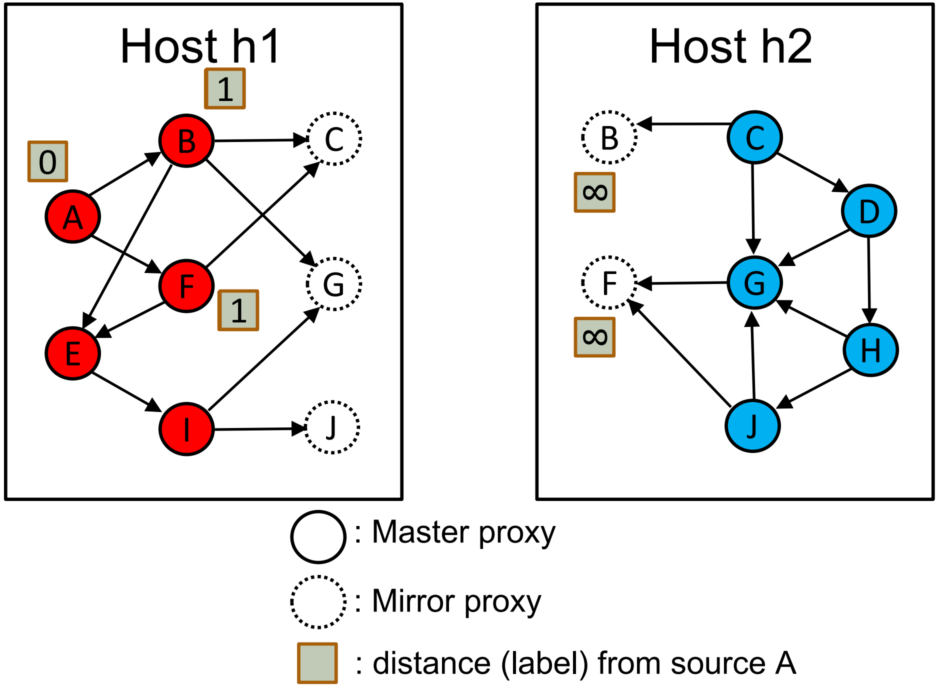
# How to synchronize the proxies?

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



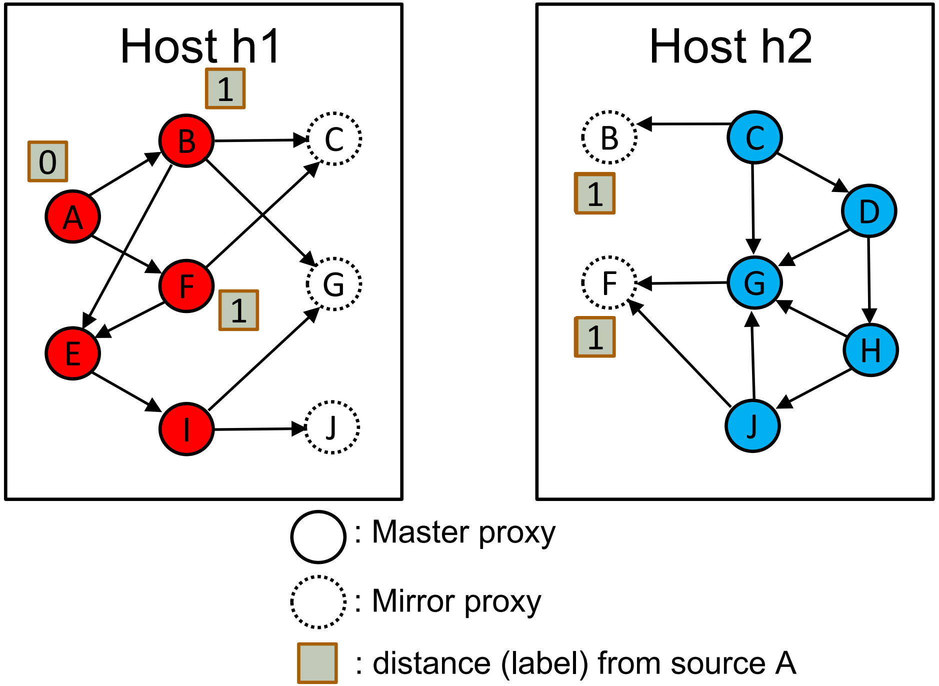
# How does Gluon synchronize the proxies?

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

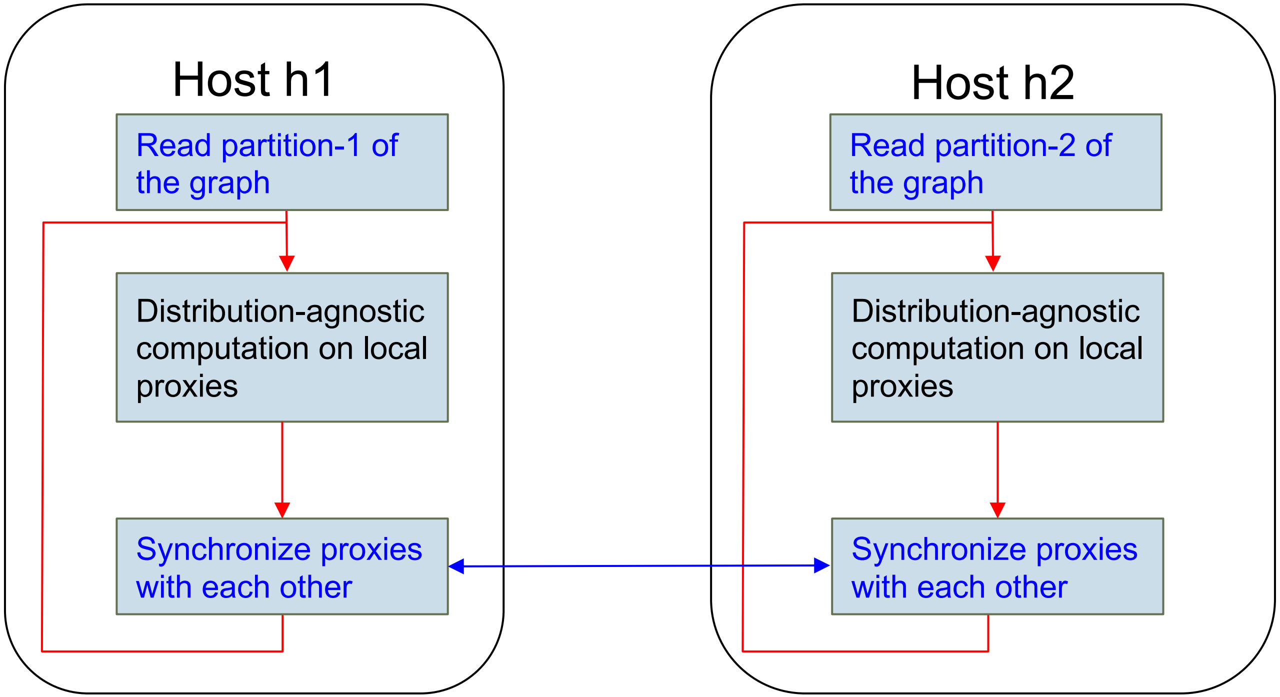


# How does Gluon synchronize the proxies?

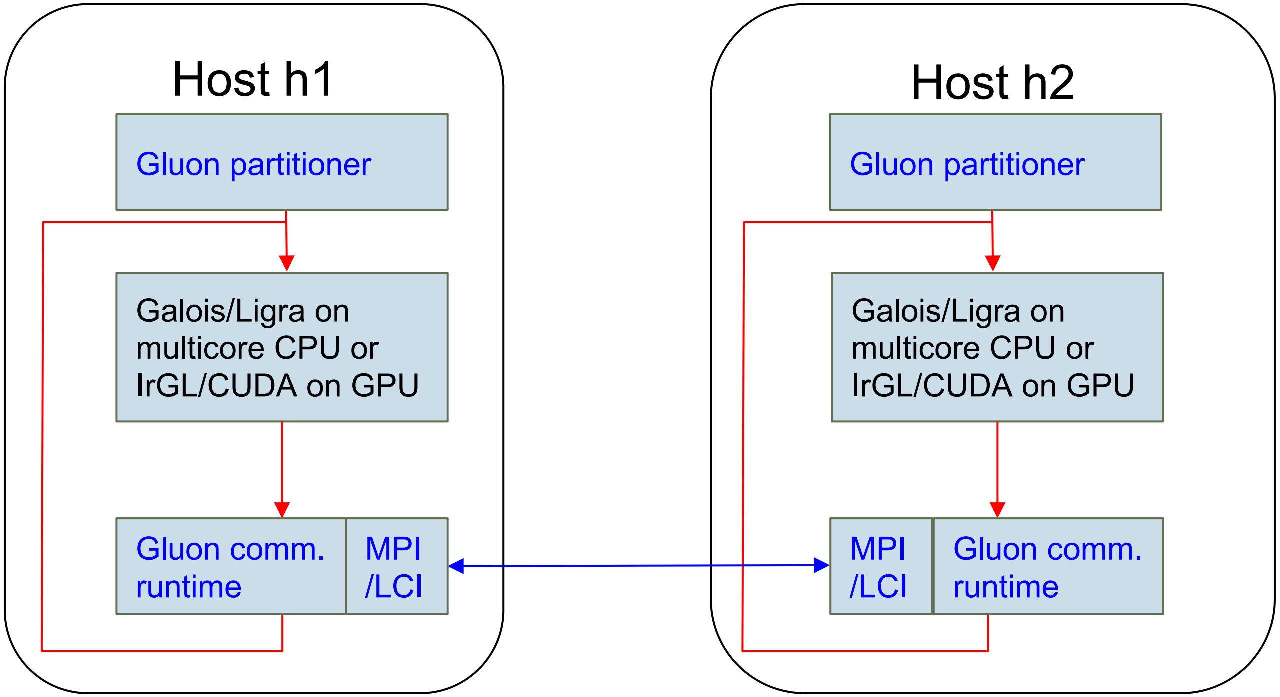
- Exploit domain knowledge
  - Cached copies can be stale as long as they are eventually synchronized
- Use all-reduce:
  - Reduce from mirror proxies to master proxy
  - Broadcast from master proxy to mirror proxies



# When to synchronize proxies?



# Gluon Distributed Execution Model




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

# Gluon Synchronization API

- Application-specific:
  - **What:** Label to synchronize
  - **When:** Point of synchronization
  - **How:** Reduction operator to use
- Platform-specific:
  - Access functions for labels (specific to data layout)

# Exploiting Structural Invariants to Optimize Communication

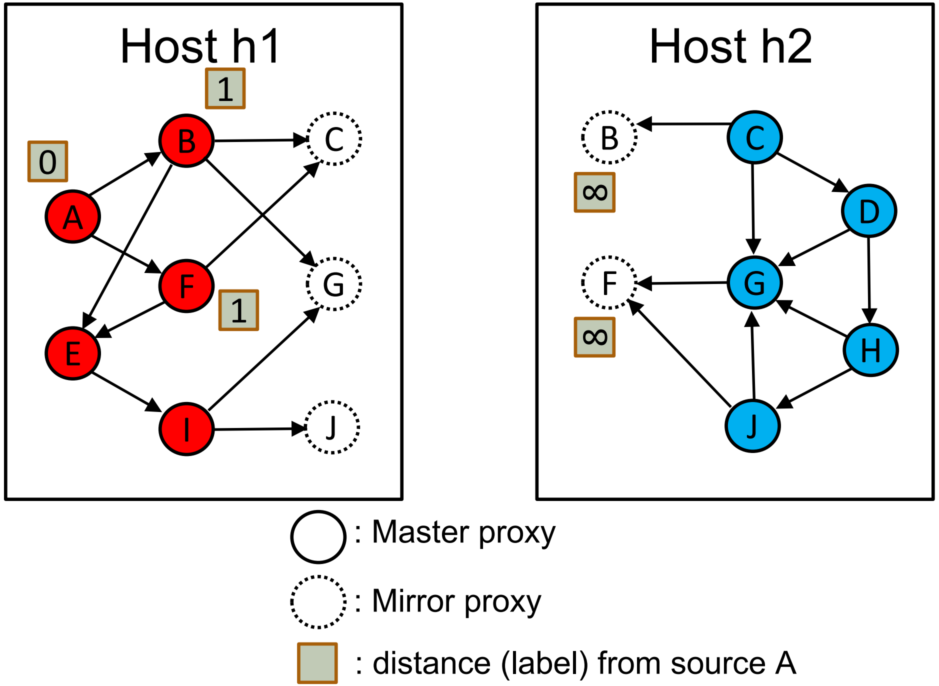
# Structural invariants in the partitioning

Structural invariants in this partitioning:

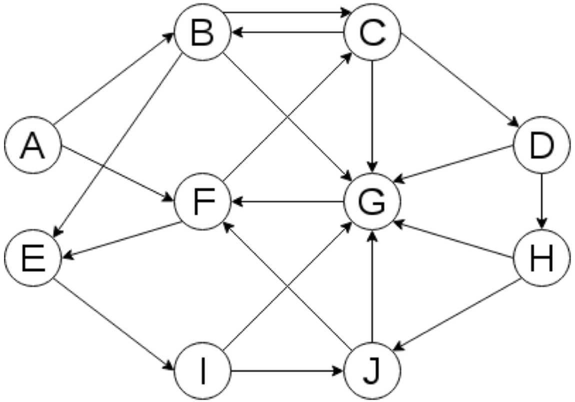
- Mirror proxies do not have outgoing edges

As a consequence, for sssp:

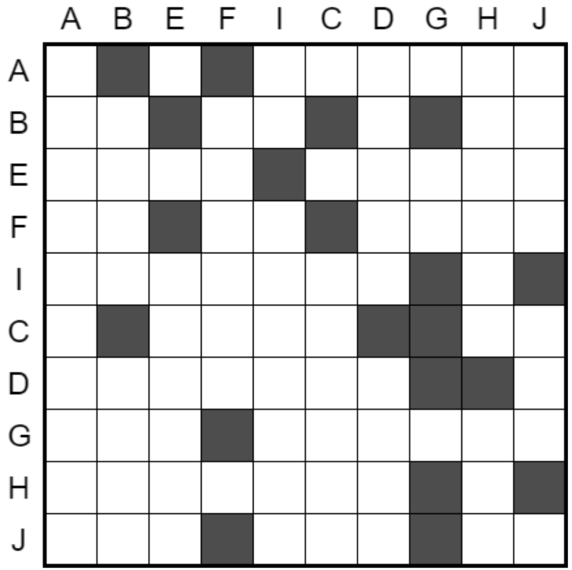
- Mirror proxies do not read their distance label
- Broadcast from master proxy to mirror proxies is not required



# Graph as an adjacency matrix

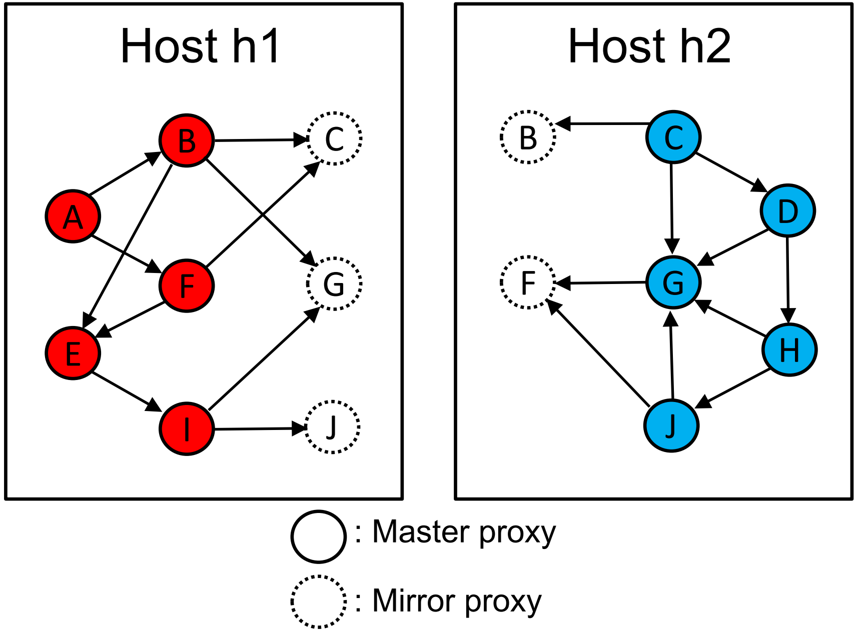



Graph view

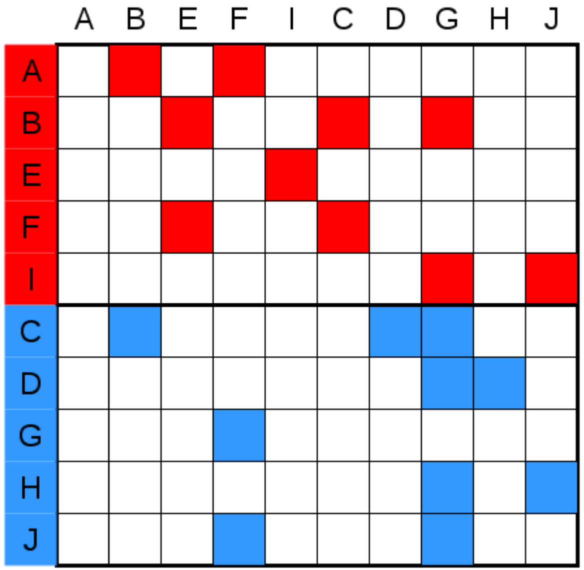



Adjacency matrix view

# Partitioning an adjacency matrix



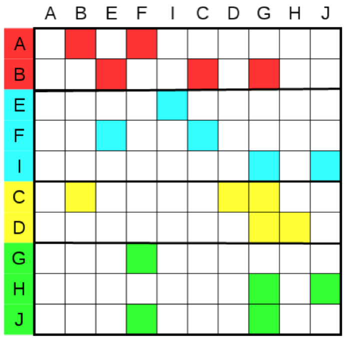

Graph view



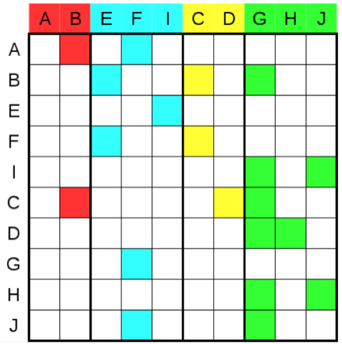

Adjacency matrix view

# Partitioning strategies with 4 partitions

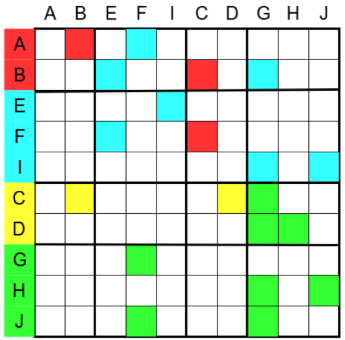
Outgoing Edge Cut (OEC)



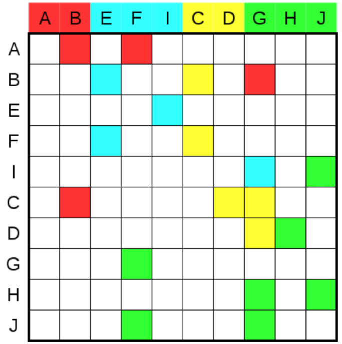

Incoming Edge Cut (IEC)



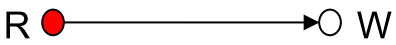

Cartesian Vertex Cut (CVC)




Unconstrained Vertex Cut (UVC)



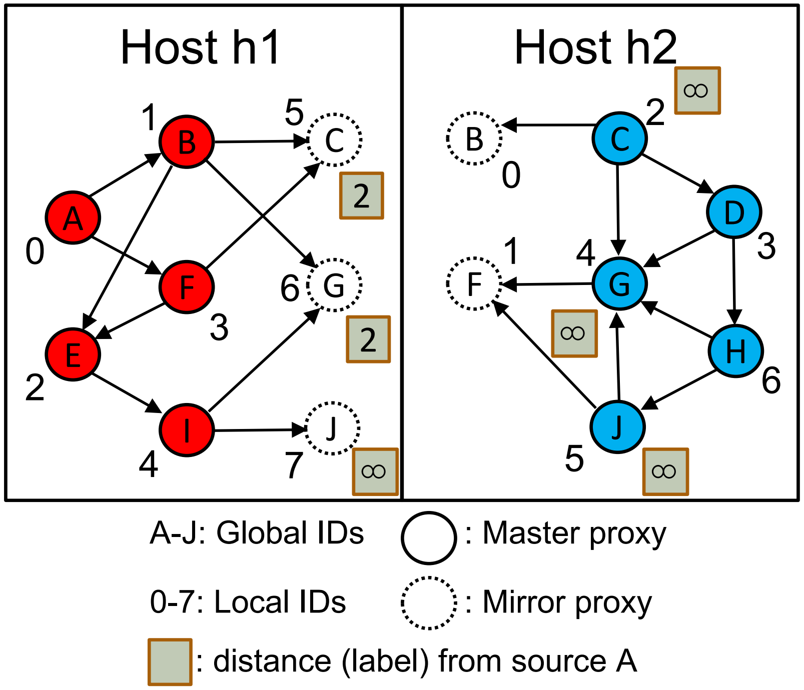
# Partitioning: strategies, constraints, invariants

- Algorithm invariant in SSSP: 


| Strategy | Constraints and Invariants | SSSP: Invariants | SSSP: Sync |
|---|---|---|---|
| **Outgoing Edge-Cut (OEC)** | Mirrors: no outgoing edges | Mirrors: label *not read* | *Reduce* |
| **Incoming Edge-Cut (IEC)** | Mirrors: no incoming edges | Mirrors: label *not written* | *Broadcast* |
| **Cartesian Vertex-Cut (CVC)** | Mirrors: either no outgoing edges or no incoming edges | Mirrors: either label *not read* or *label not written* | *Reduce-partial & Broadcast-partial* |
| **Unconstrained Vertex-Cut (UVC)** | None | None | *Reduce & Broadcast* |

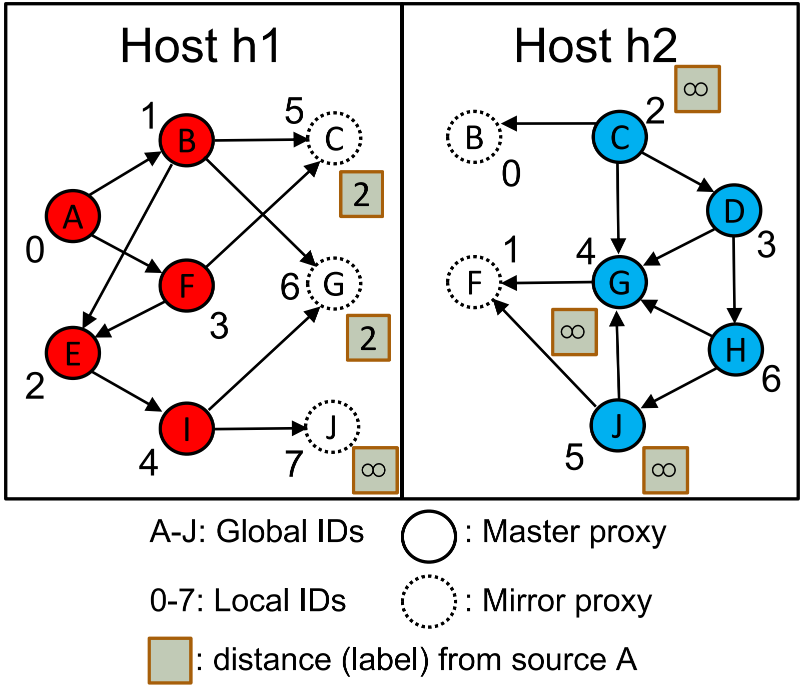
# Exploiting Temporal Invariance to Optimize Communication

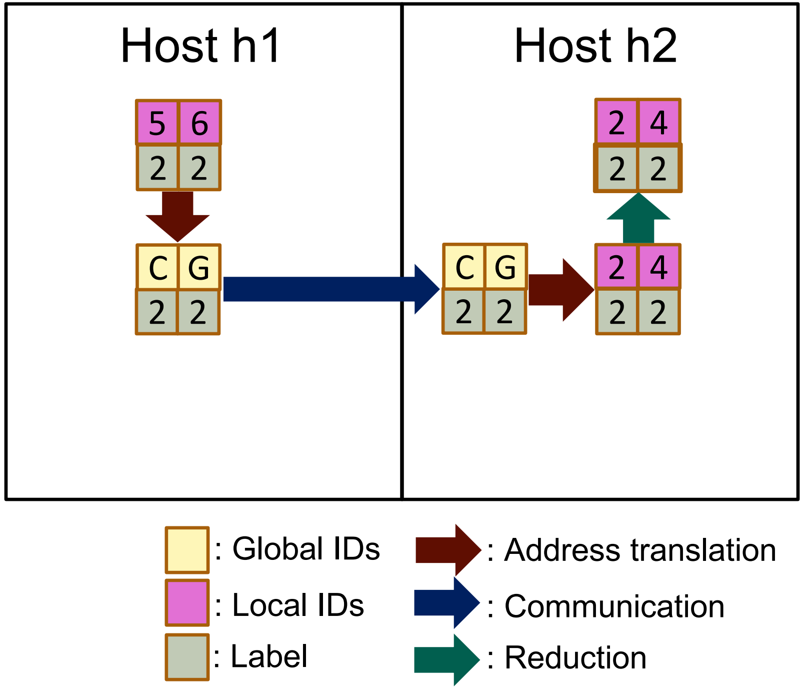
# Bulk-communication



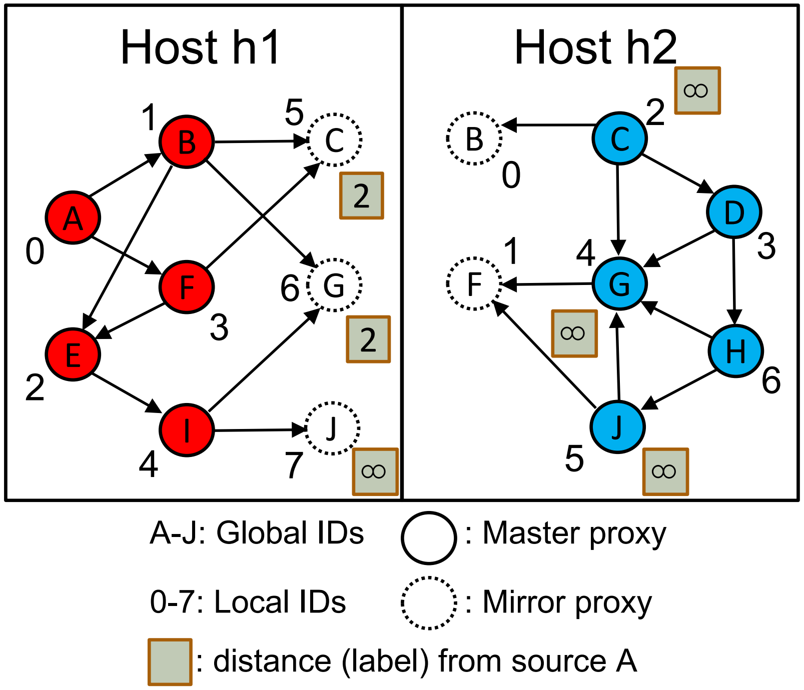

- Proxies of millions of nodes need to be synchronized in a round
  - Not every node is updated in every round
- Address spaces (local-IDs) of different hosts are different
- Existing systems: use address translation and communicate global-IDs along with updated values

# Bulk-communication in existing systems



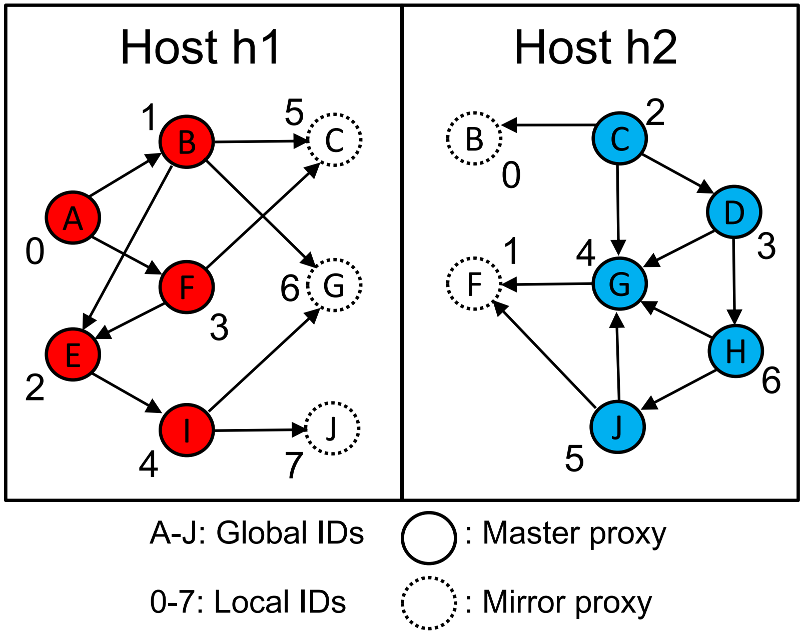



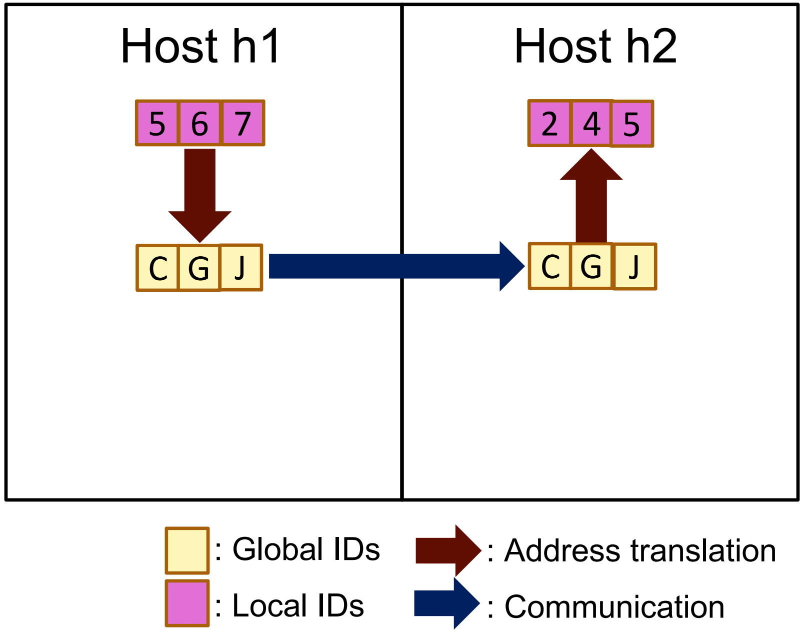
# Bulk-communication in Gluon




- 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

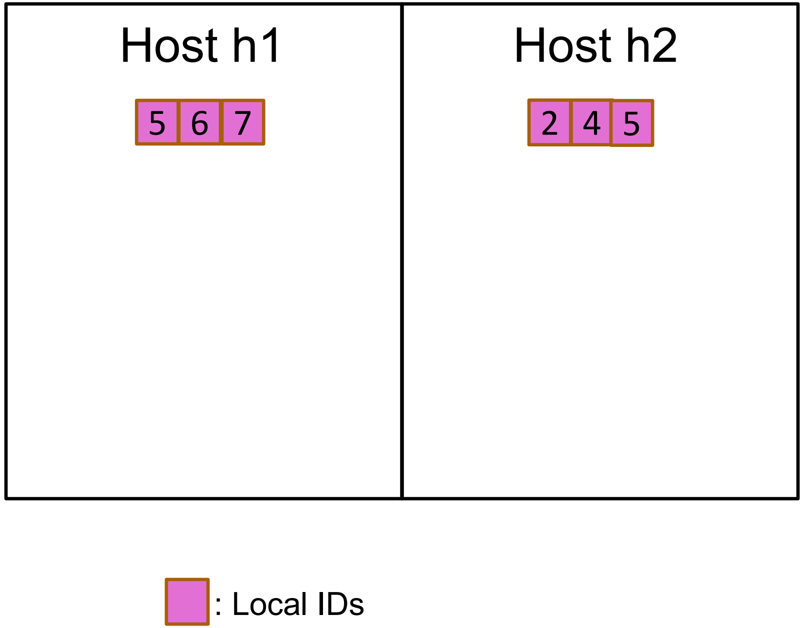
# Optimization I: memoization of address translation



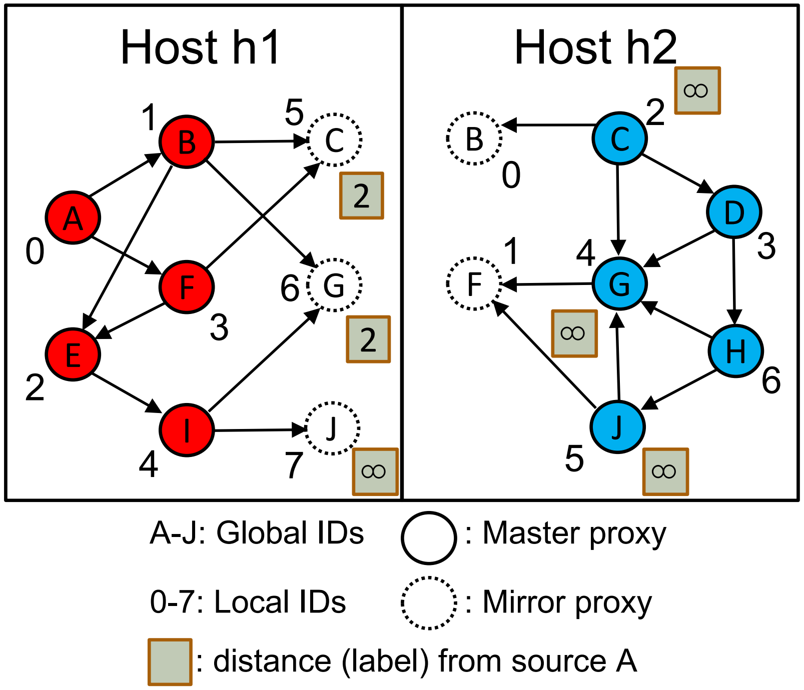



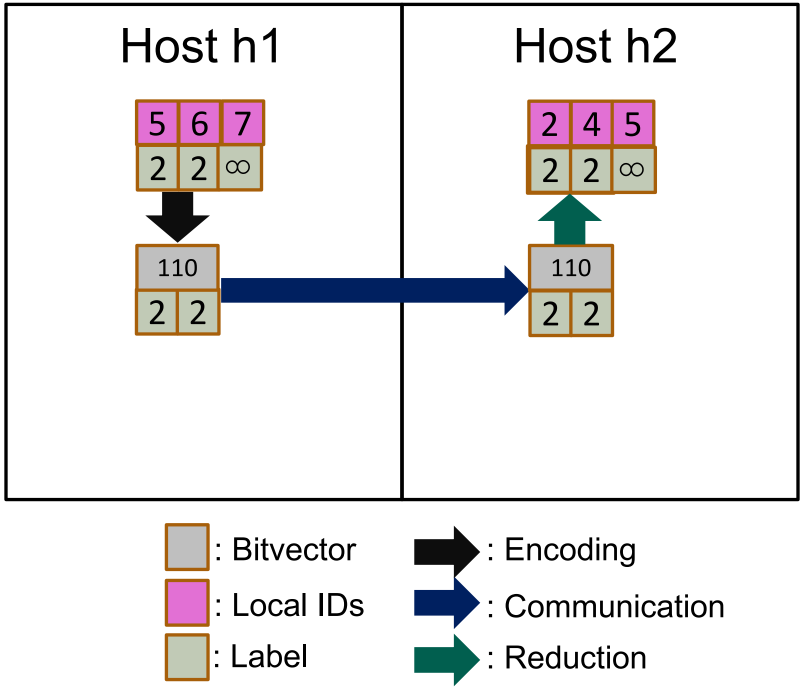
# Communication after memoization

- Memoization establishes the order of local-IDs shared between hosts
- Send and receive labels in each round in the same order
- Only some labels updated in a round:
  - Send simple encoding that captures which of the local-IDs in the order were updated



# Optimization II: encoding of updated local-IDs



# Experimental Results

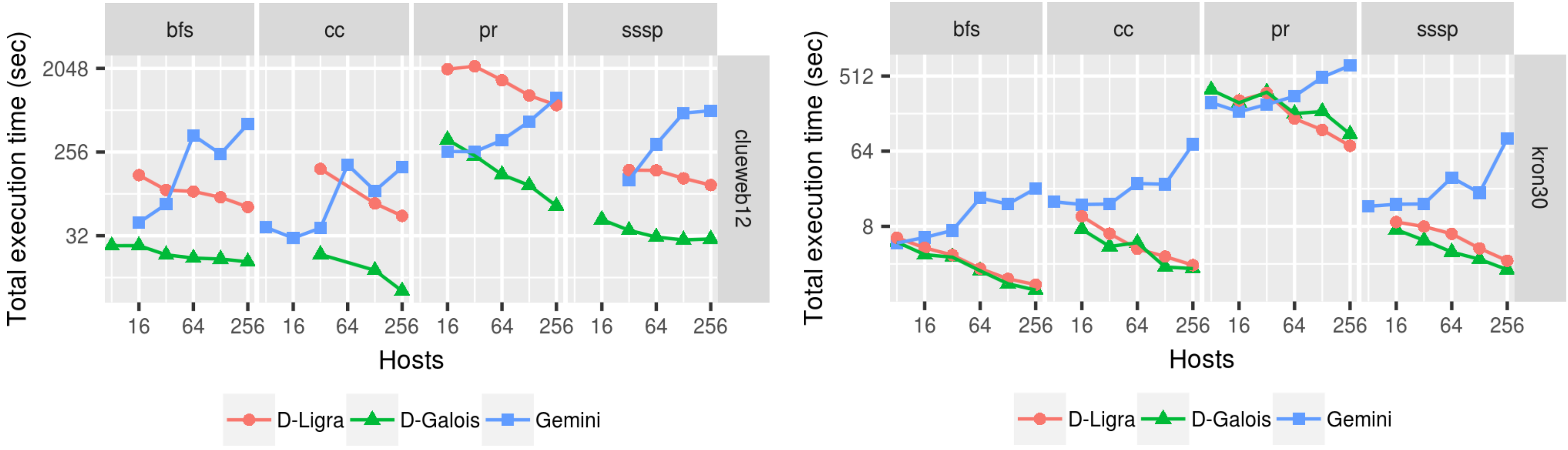
# Experimental setup

- **Systems:**
  - D-Ligra (Gluon + Ligra)
  - D-Galois (Gluon + Galois)
  - D-IrGL (Gluon + IrGL)
  - Gemini (state-of-the-art) [OSDI'16]
- **Benchmarks:**
  - Breadth first search (bfs)
  - Connected components (cc)
  - Pagerank (pr)
  - Single source shortest path (sssp)

| Inputs | rmat28 | kron30 | clueweb12 | wdc12 |
|---|---|---|---|---|
| \|V\| | 268M | 1073M | 978M | 3,563M |
| \|E\| | 4B | 11B | 42B | 129B |
| \|E\|/\|V\| | 16 | 16 | 44 | 36 |
| Size (CSR) | 35GB | 136GB | 325GB | **986GB** |

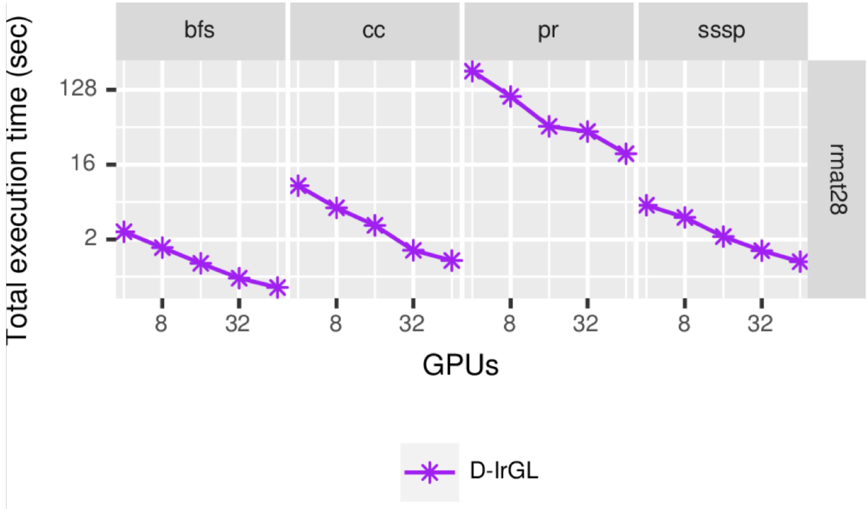
| Clusters | Stampede (CPU) | Bridges (GPU) |
|---|---|---|
| Max. hosts | **256** | **64** |
| Machine | Intel Xeon Phi KNL | 4 NVIDIA Tesla K80s |
| Each host | 272 threads of KNL | 1 Tesla K80 |
| Memory | 96GB DDR3 | 12GB DDR5 |

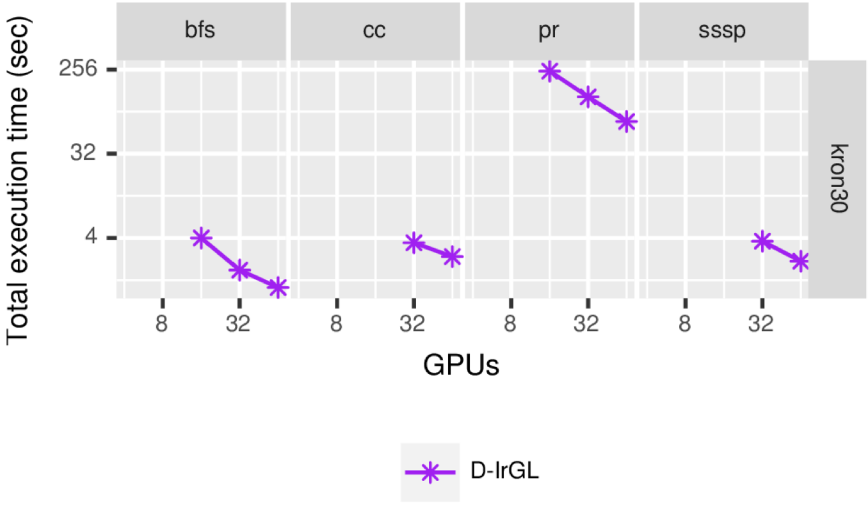
# Strong scaling on Stampede (68 cores on each host)




**D-Galois performs best and scales well**

# Strong scaling on Bridges (4 GPUs share a physical node)

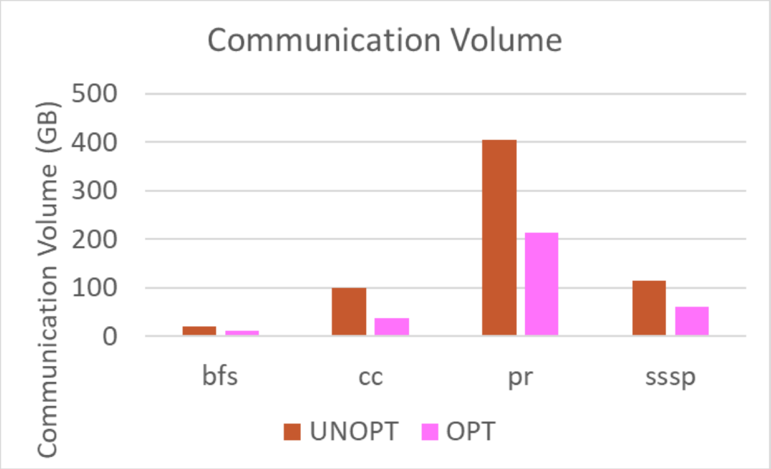


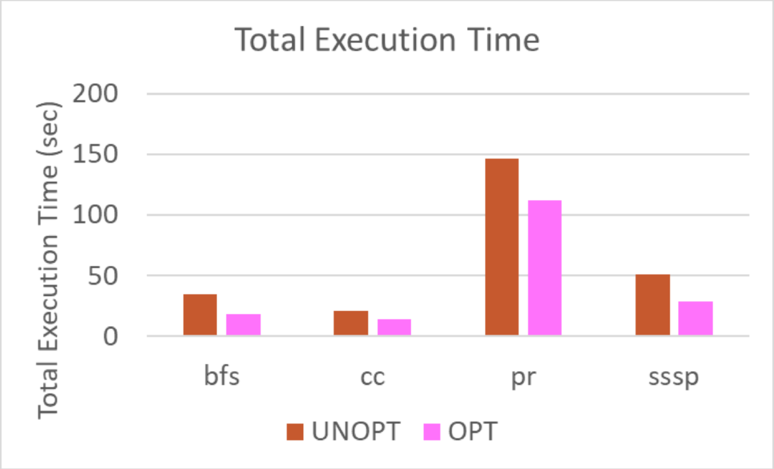




**D-IrGL scales well**

# Impact of Gluon's communication optimizations

D-Galois on 128 hosts of Stampede: clueweb12



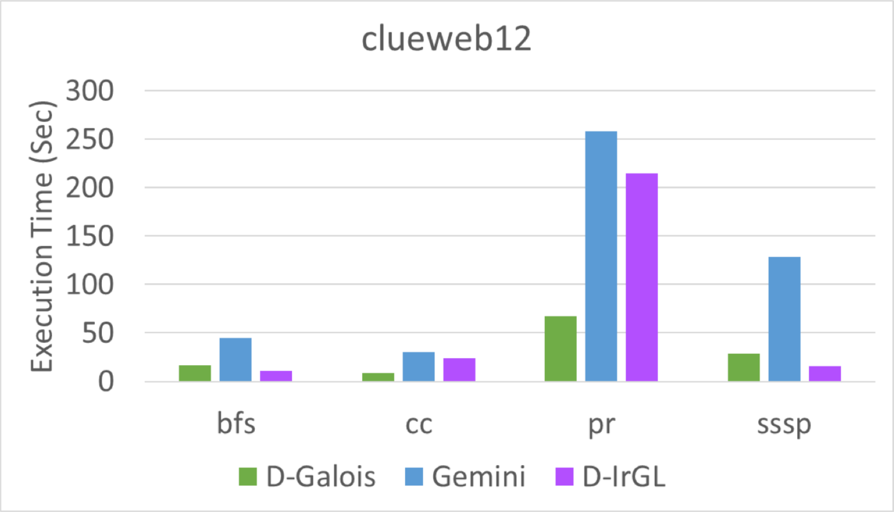



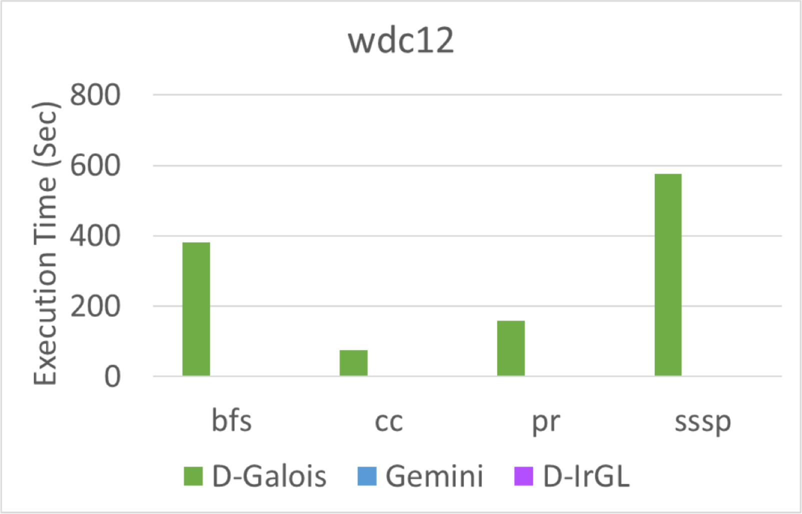

Improvement (geometric mean):

Communication volume: **2x**

Execution time: **2.6x**

# Fastest execution time (sec) using best-performing number of hosts/GPUs




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

# Subsequent work using Gluon

- [EuroPar’18] **Abelian** compiler:
  shared-memory Galois apps ---> distributed, heterogeneous (D-Galois + D-IrGL) apps
- [VLDB’18] Partitioning Policies: Cartesian Vertex Cut performs best at scale
- [PPoPP’19] Min-Rounds Betweenness Centrality (**MRBC**) algorithm
- [ASPLOS’19] **Phoenix**: fault-tolerance without overhead during fault-free execution
- [IPDPS’19] Fast Customizable Streaming Edge Partitioner (**CuSP**)

# 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
- Gluon, D-Galois, and D-IrGL: publicly available in Galois v4.0

  http://iss.ices.utexas.edu/?p=projects/galois

- Use Gluon to scale out your shared-memory graph analytical applications

# Backup slides

# Graph construction time (sec)

| 1 host | rmat26 | twitter40 | rmat28 |
|---|---|---|---|
| **Ligra** | 271.6 | 158.3 | 396.9 |
| **Galois** | **64.9** | **51.8** | **123.9** |
| **Gemini** | 854.3 | 893.5 | 3084.7 |

| 256 hosts | rmat28 | kron30 | clueweb12 | wdc12 |
|---|---|---|---|---|
| **D-Ligra** | 69.4 | 235.8 | 470.5 | 1515.9 |
| **D-Galois** | **65.5** | **225.7** | **396.2** | **1345.0** |
| **Gemini** | 231.0 | 921.8 | 1247.7 | X |

# Execution time (sec) on a single node of Stampede ("-" means out-of-memory)

| Input | twitter40 | | | | rmat28 | | | |
|---|---|---|---|---|---|---|---|---|
| Benchmark | bfs | cc | pr | sssp | bfs | cc | pr | sssp |
| **Ligra** | **0.31** | 2.75 | 175.67 | 2.60 | **0.77** | 17.56 | 542.51 | - |
| **D-Ligra** | 0.44 | 3.16 | 188.70 | 2.92 | 1.21 | 18.30 | 597.30 | - |
| **Galois** | 0.68 | 2.73 | **43.47** | 5.55 | 2.54 | 13.20 | **116.50** | 21.42 |
| **D-Galois** | 1.03 | **1.04** | 86.53 | **1.84** | 4.05 | **7.02** | 326.88 | **5.47** |
| **Gemini** | 0.85 | 3.96 | 80.23 | 3.78 | 3.44 | 20.34 | 351.65 | 41.77 |

# Execution time (sec) on a single node of Bridges with 4 K80 GPUs (“-” means out-of-memory).

| Input | rmat26 | | | | twitter40 | | | |
|---|---|---|---|---|---|---|---|---|
| **Benchmark** | **bfs** | **cc** | **pr** | **sssp** | **bfs** | **cc** | **pr** | **sssp** |
| **Gunrock** | - | 1.81 | 51.46 | 1.42 | 0.88 | 1.46 | 37.37 | 2.26 |
| **D-IrGL(OEC)** | 3.61 | 5.72 | 55.72 | 4.13 | 1.03 | 1.57 | 62.81 | 1.99 |
| **D-IrGL(IEC)** | **0.72** | 7.88 | **7.65** | **0.84** | **0.73** | 1.55 | **35.03** | **1.44** |
| **D-IrGL(HVC)** | 0.82 | **1.53** | 8.54 | 0.95 | 1.08 | 1.58 | 44.35 | 2.04 |
| **D-IrGL(CVC)** | 2.11 | 4.22 | 46.91 | 2.24 | 0.87 | **1.39** | 46.86 | 2.32 |

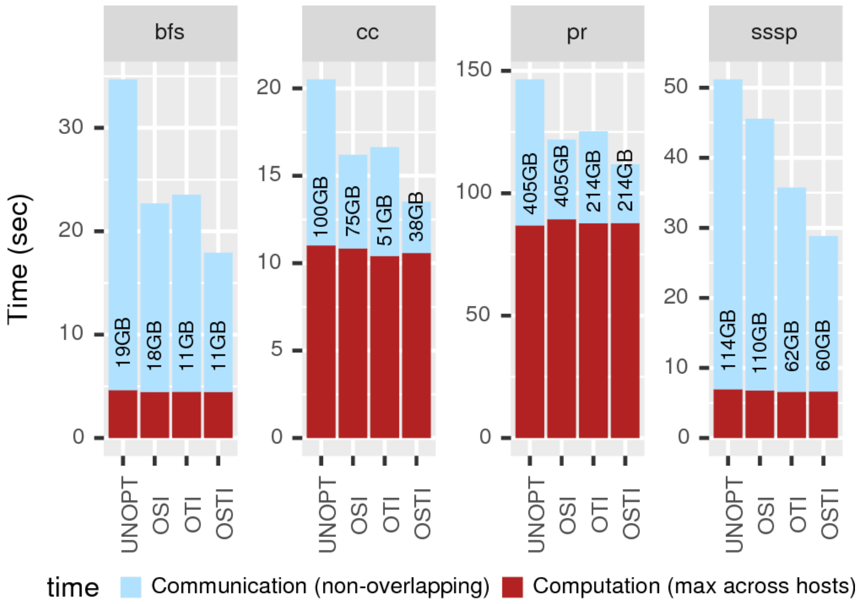
# Fastest execution time (sec) of all systems using best-performing number of hosts/GPUs

| Bench-mark | Input | CPUs | | | GPUs |
|---|---|---|---|---|---|
| | | D-Ligra | D-Galois | Gemini | D-IrGL |
| bfs | rmat28 | 1.0 (128) | **0.8 (256)** | 2.3 (8) | **0.5 (64)** |
| | kron30 | 1.6 (256) | **1.4 (256)** | 5.0 (8) | **1.2 (64)** |
| | clueweb12 | 65.3 (256) | **16.7 (256)** | 44.4 (16) | **10.8 (64)** |
| | wdc12 | 1995.3 (64) | **380.8 (256)** | X | - |
| cc | rmat28 | 1.4 (256) | **1.3 (256)** | 6.6 (8) | **1.1 (64)** |
| | kron30 | 2.7 (256) | **2.5 (256)** | 14.6 (16) | **2.5 (64)** |
| | clueweb12 | 52.3 (256) | **8.1 (256)** | 30.2 (16) | **23.8 (64)** |
| | wdc12 | 176.6 (256) | **75.3 (256)** | X | - |

| Bench-mark | Input | CPUs | | | GPUs |
|---|---|---|---|---|---|
| | | D-Ligra | D-Galois | Gemini | D-IrGL |
| pr | rmat28 | **19.7 (256)** | 24.0 (256) | 108.4 (8) | **21.6 (64)** |
| | kron30 | **74.2 (256)** | 102.4 (256) | 190.8 (16) | **70.9 (64)** |
| | clueweb12 | 821.1 (256) | **67.0 (256)** | 257.9 (32) | **215.1 (64)** |
| | wdc12 | 663.1 (256) | **158.2 (256)** | X | - |
| sssp | rmat28 | 2.1 (256) | **1.4 (256)** | 6.3 (4) | **1.1 (64)** |
| | kron30 | 3.1 (256) | **2.4 (256)** | 13.9 (8) | **2.3 (64)** |
| | clueweb12 | 112.5 (256) | **28.8 (128)** | 128.3 (32) | **15.8 (64)** |
| | wdc12 | 2985.9 (256) | **574.9 (256)** | X | - |

D-Galois and D-IrGL are faster than Gemini by factors of **3.9x** and **4.9x** on the average.

# 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**

# Impact of Gluon's communication optimizations

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

| Benchmark | Communication volume (GB) | | Total execution time (sec) | |
|---|---|---|---|---|
| | UNOPT | OPT | UNOPT | OPT |
| bfs | 19 | 11 | 34.7 | 17.9 |
| cc | 100 | 38 | 20.5 | 13.5 |
| pr | 405 | 214 | 146.5 | 111.8 |
| sssp | 114 | 60 | 51.2 | 28.8 |

Improvement (geometric mean):

- Communication volume: **2x**
- Execution time: **2.6x**

# Fastest execution time (sec) of all systems using best-performing number of hosts/GPUs

clueweb12

| Benchmark | D-Galois | Gemini | D-IrGL |
|---|---|---|---|
| bfs | 16.7 (256) | 44.4 (16) | 10.8 (64) |
| cc | 8.1 (256) | 30.2 (16) | 23.8 (64) |
| pr | 67.0 (256) | 257.9 (32) | 215.1 (64) |
| sssp | 28.8 (128) | 128.3 (32) | 15.8 (64) |

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

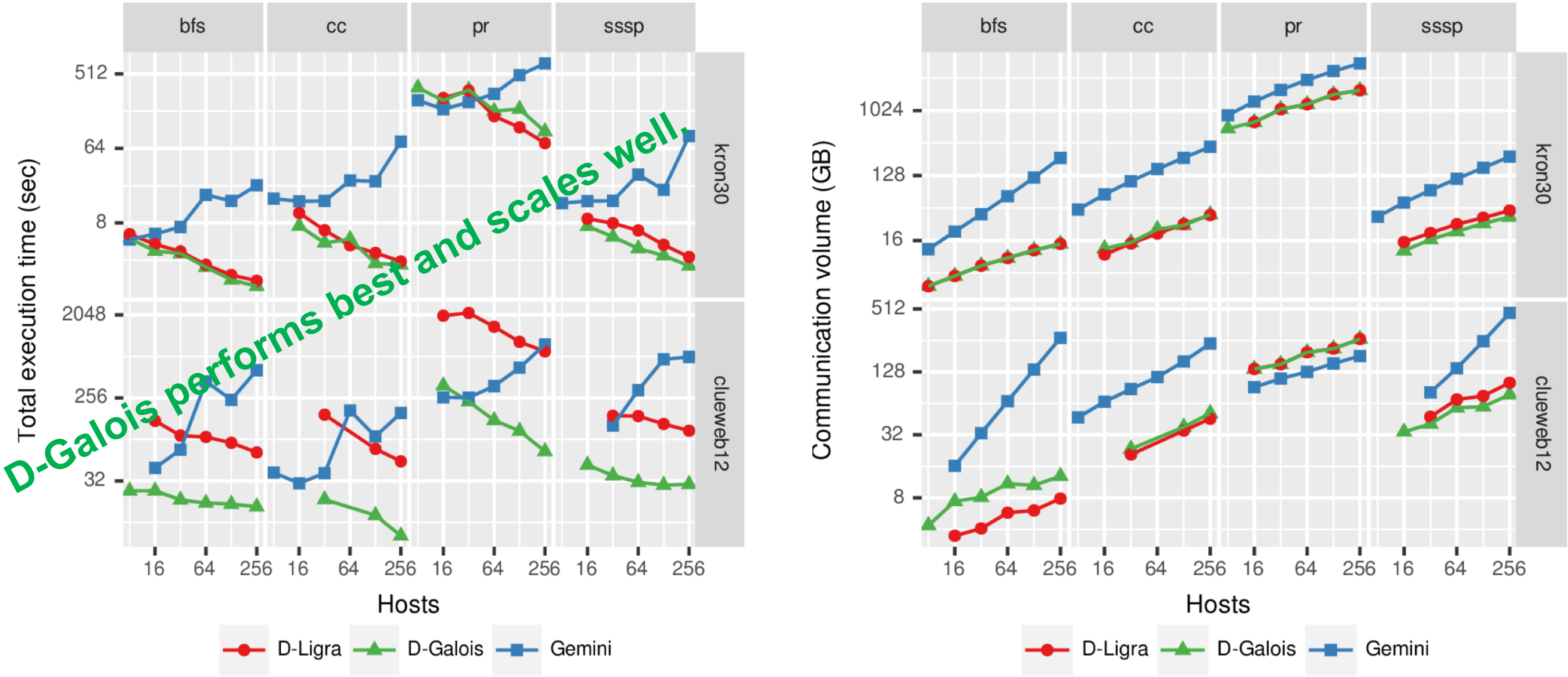
# Fastest execution time (sec) of all systems using best-performing number of hosts

wdc12

| Bench mark | D-Ligra | D-Galois | Gemini |
|---|---|---|---|
| bfs | 1995 | 381 | X |
| cc | 177 | 75 | X |
| pr | 663 | 158 | X |
| sssp | 2986 | 575 | X |

**D-Galois performs best**

# Strong scaling on Stampede (68 cores on each host)



# Motivation

- Distributed CPU-only graph analytics:
  - Gemini [OSDI’16], PowerGraph [OSDI’12], ...
  - *No way to reuse infrastructure, such as to leverage GPUs*
- Decouple communication from computation
- Enable communication optimization at runtime