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

PLDI 2018

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

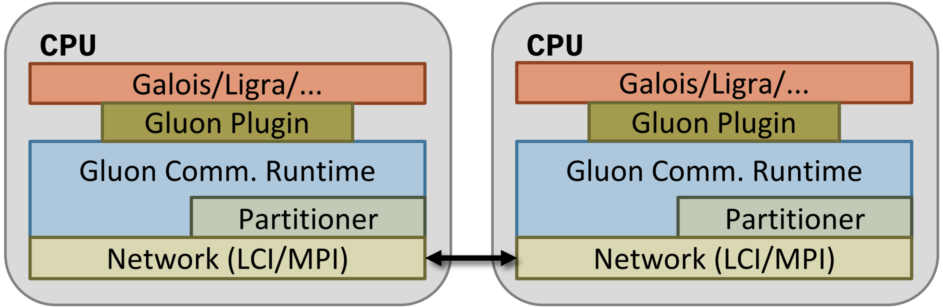
TEXAS The University of Texas at Austin
ILLINOIS

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?

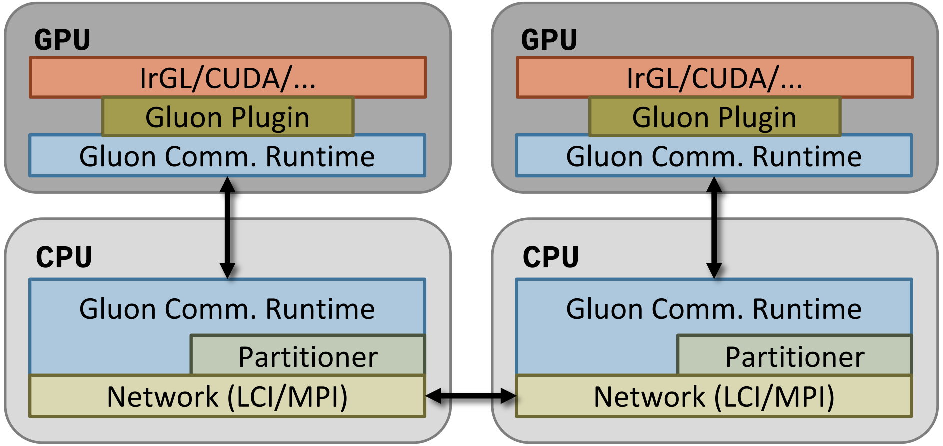
- Programmers:
  - Write shared-memory applications
  - Interface with Gluon using API
- Gluon transparently handles:
  - Graph partitioning
  - Communication and synchronization



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]

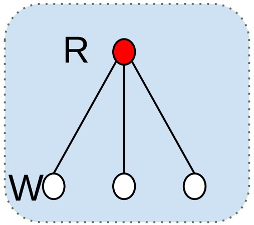
Outline

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

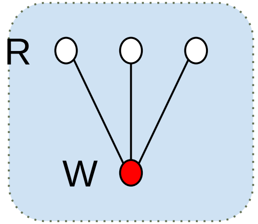
Gluon Synchronization Approach

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.

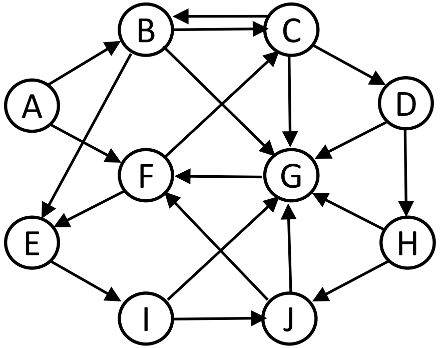


push-style



pull-style

Partitioning



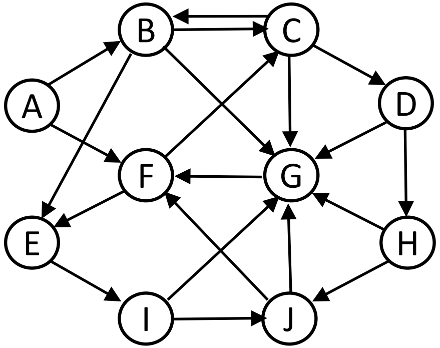
Original graph

Host h1

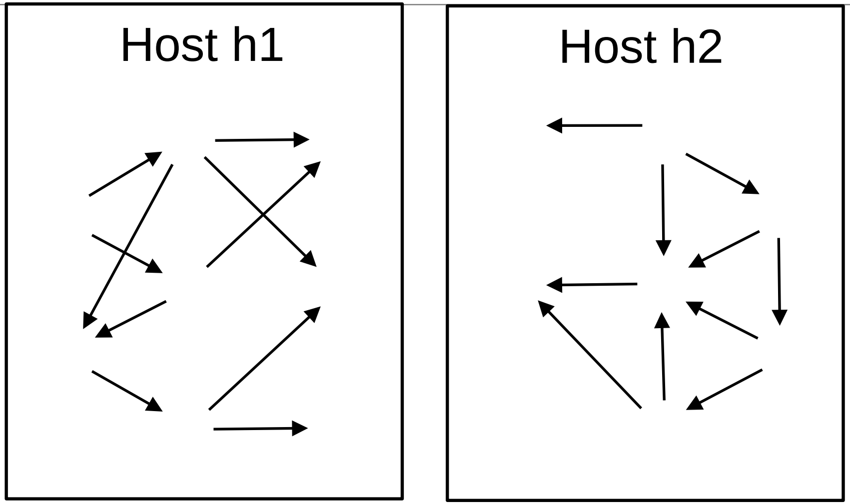
Host h2

Partitions of the graph

Partitioning



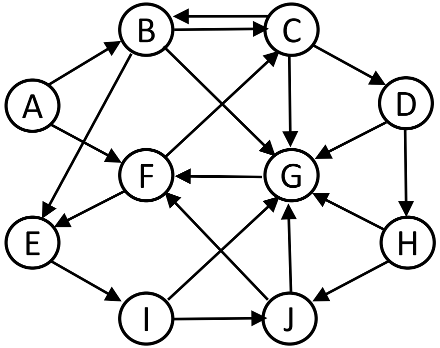
Original graph



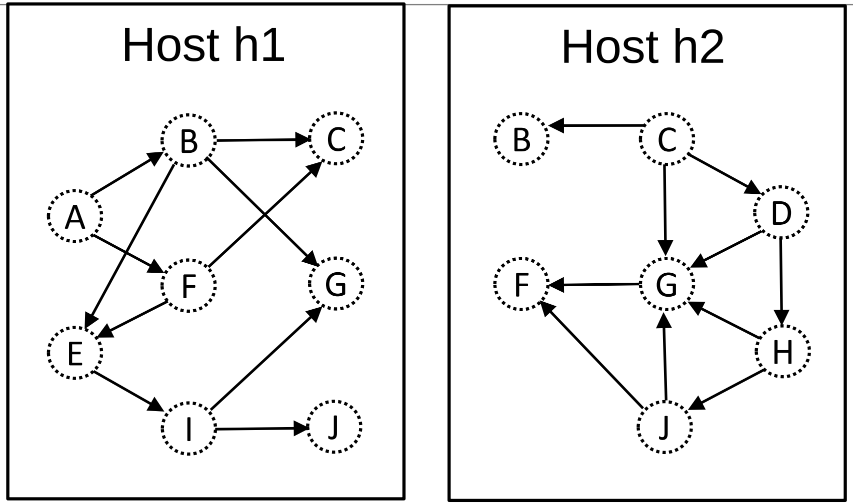
Partitions of the graph

- Each edge is assigned to a unique host

Partitioning



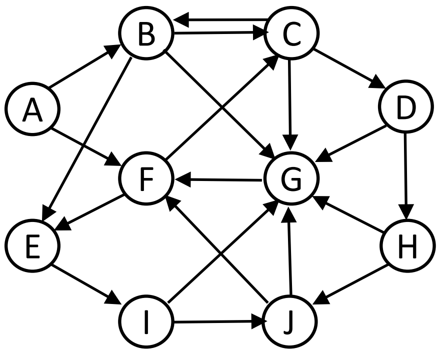
Original graph



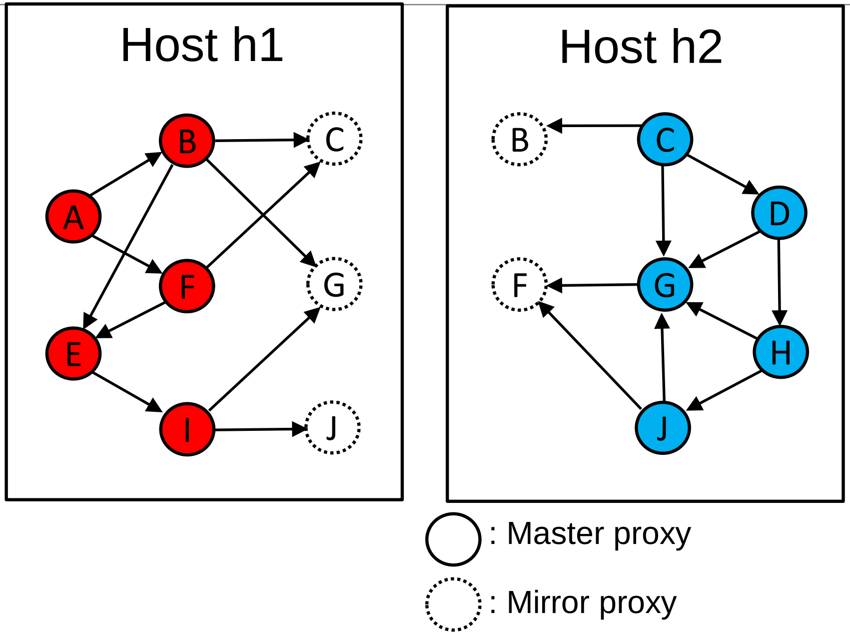
Partitions of the graph

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

Partitioning



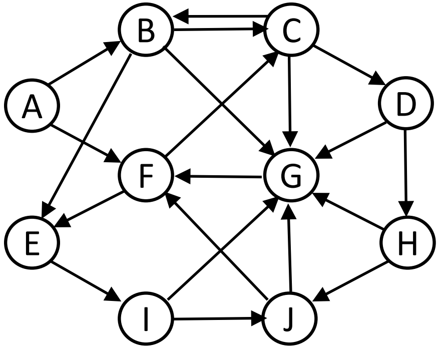
Original graph



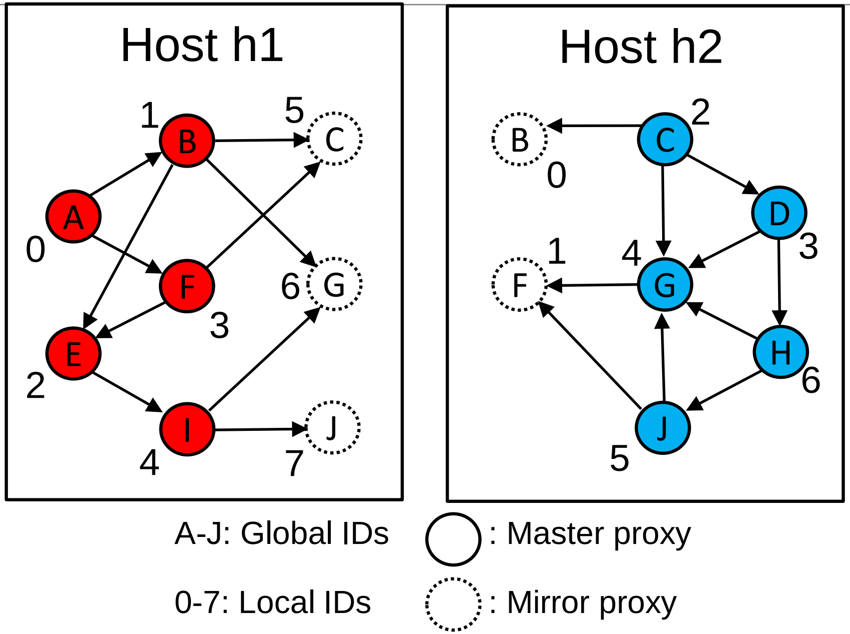
Partitions of the graph

- 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

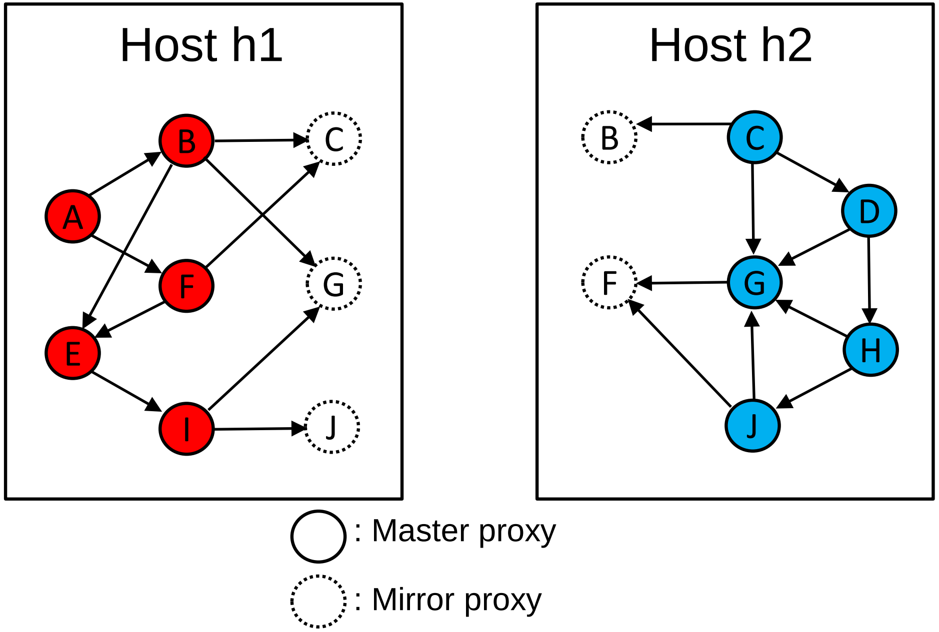


Partitions of the graph

- 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

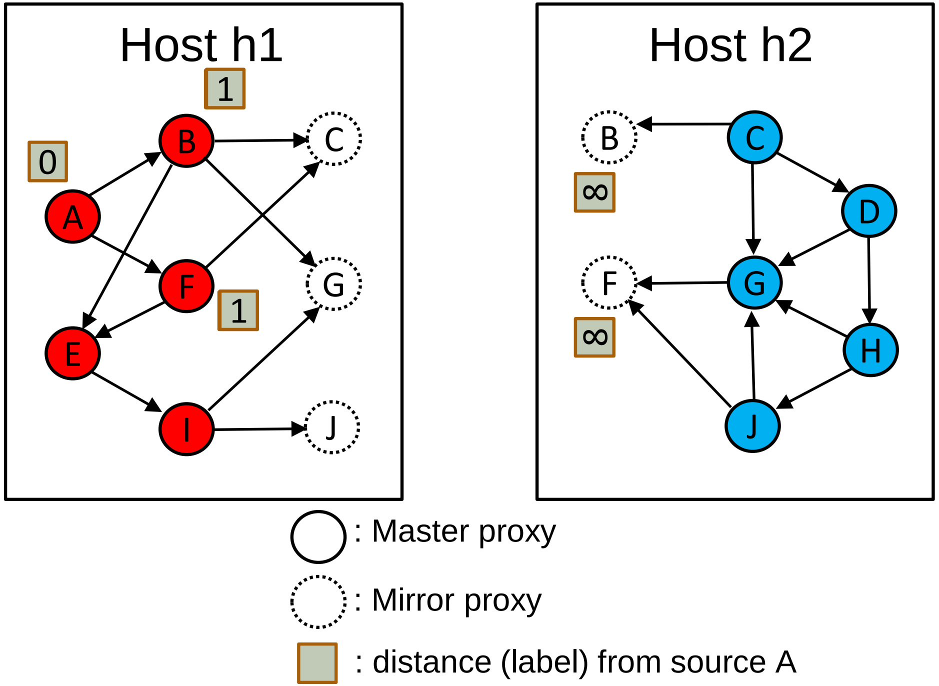
How to synchronize the proxies?

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



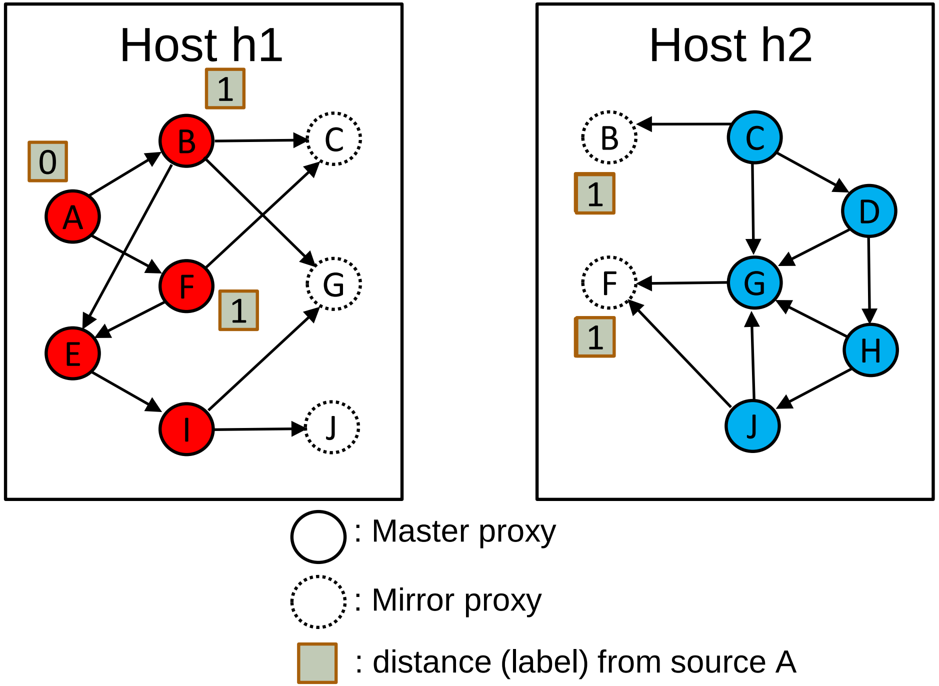
How does Gluon synchronize the proxies?

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

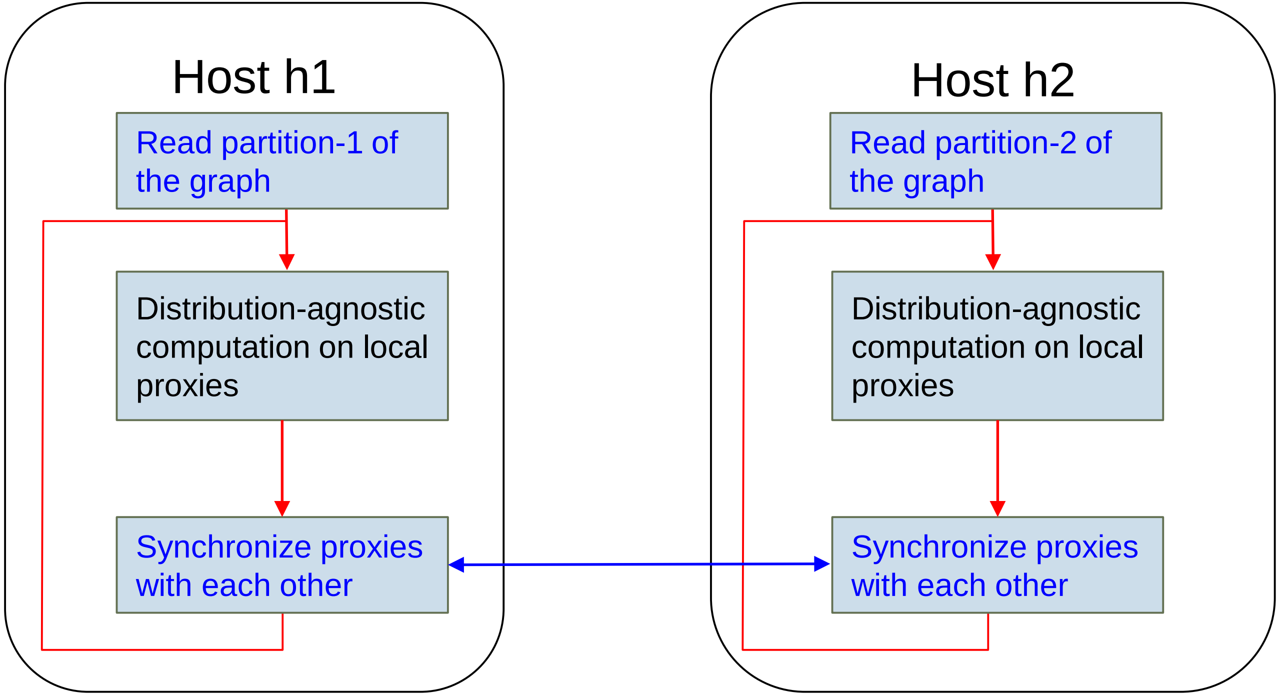


How does Gluon synchronize the proxies?

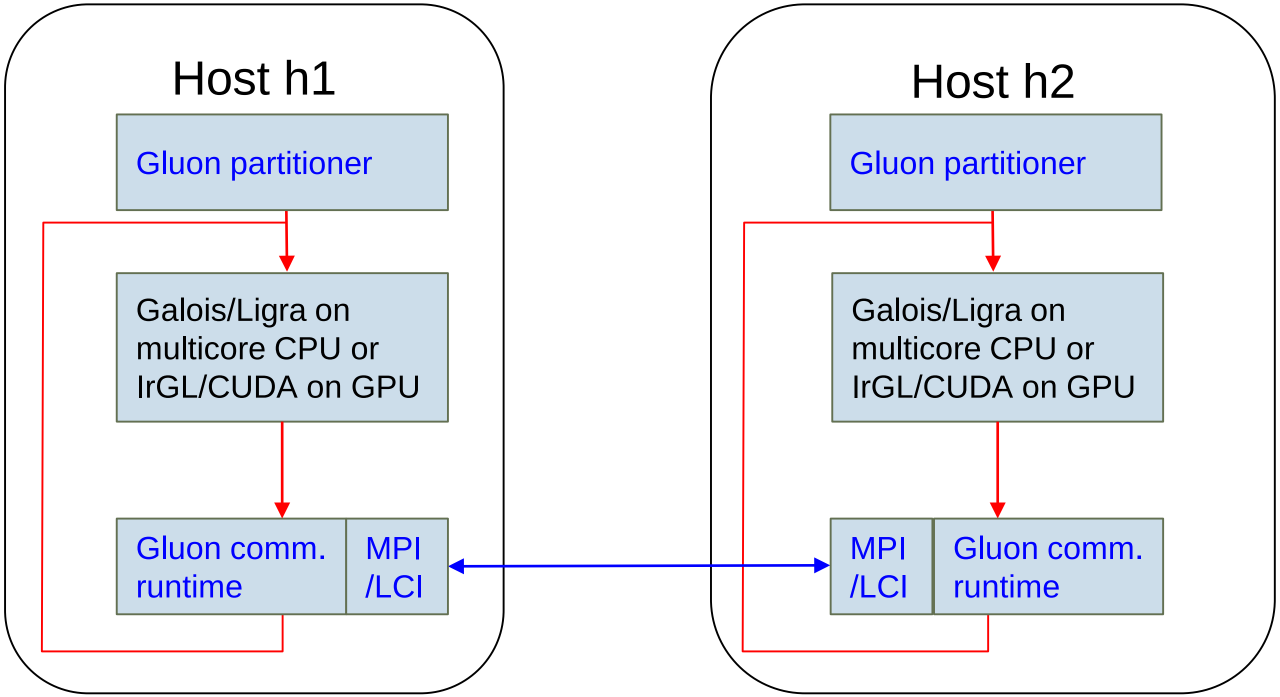
- Exploit domain knowledge
  - Cached copies can be stale as long as they are eventually synchronized
- Use all-reduce:
  - Reduce from mirror proxies to master proxy
  - Broadcast from master proxy to mirror proxies



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:** 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

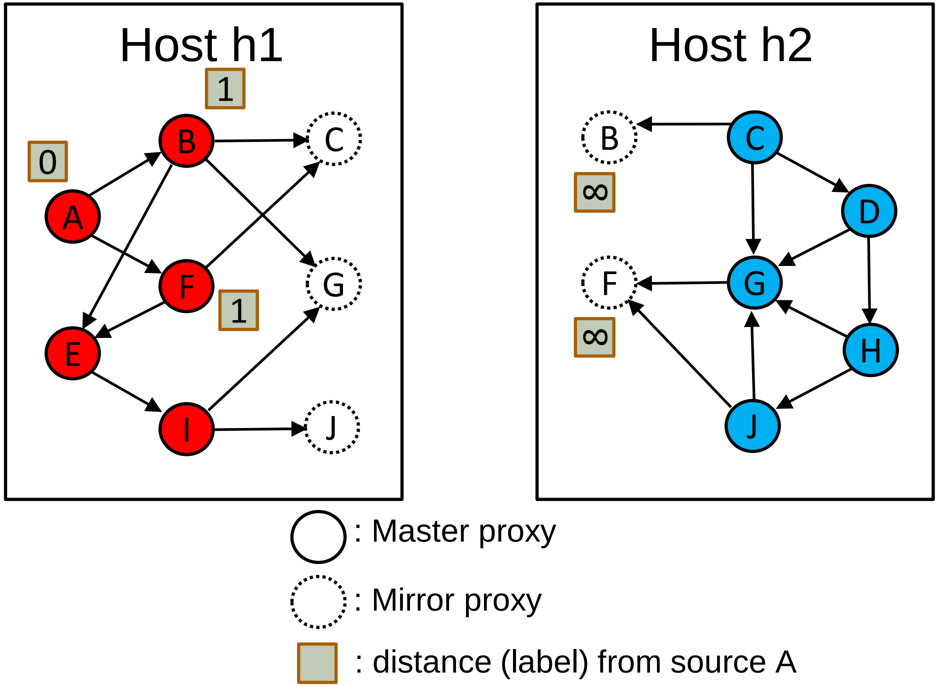
Structural invariants in the partitioning

Structural invariants in this partitioning:

- Mirror proxies do not have outgoing edges

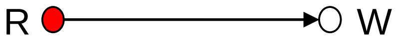
As a consequence, for sssp:

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



Partitioning: strategies, constraints, invariants

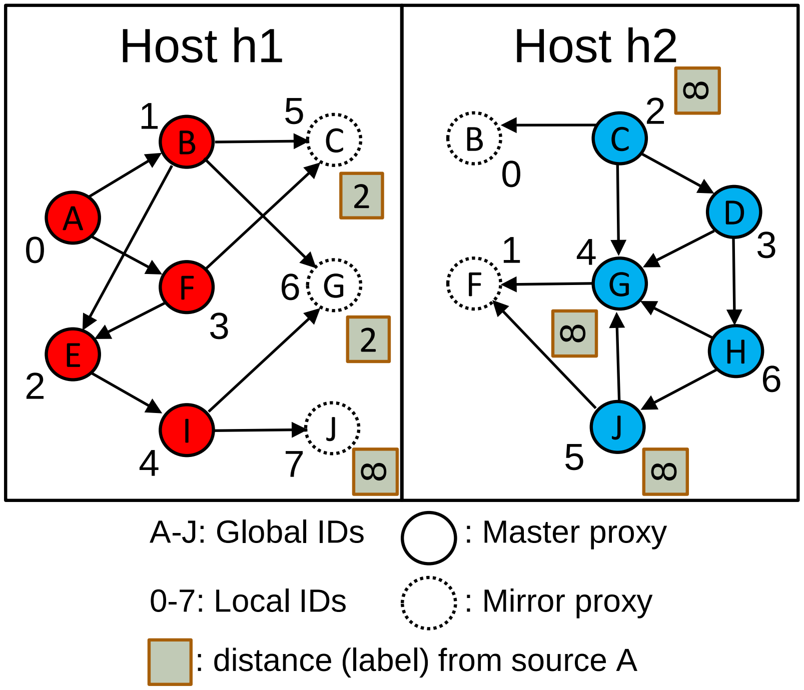
- Algorithm invariant in SSSP:



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*

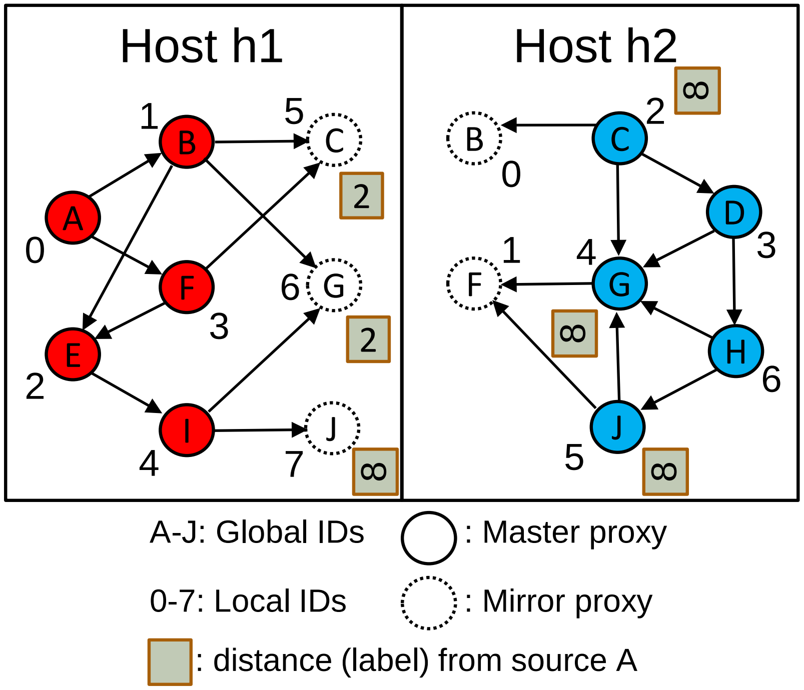
Exploiting Temporal Invariance to Optimize Communication

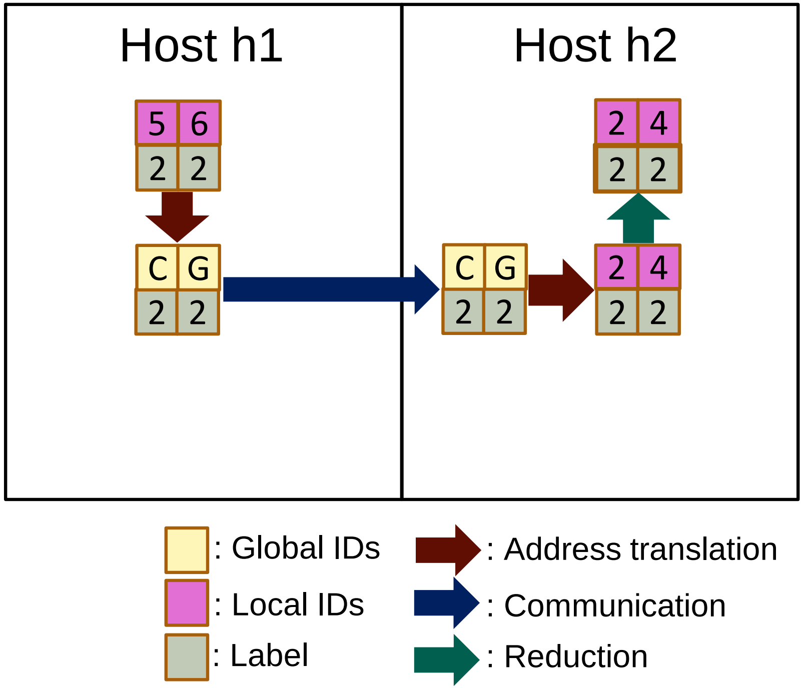
Bulk-communication



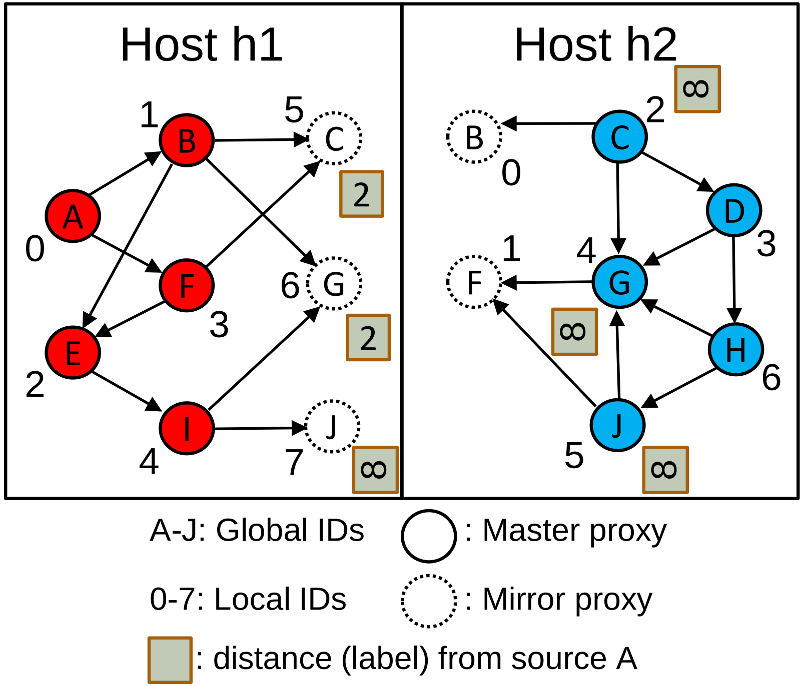
- Proxies of millions of nodes need to be synchronized in a round
  - Not every node is updated in every round
- Address spaces (local-IDs) of different hosts are different
- Existing systems: use address translation and communicate global-IDs along with updated values

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

Experimental Results

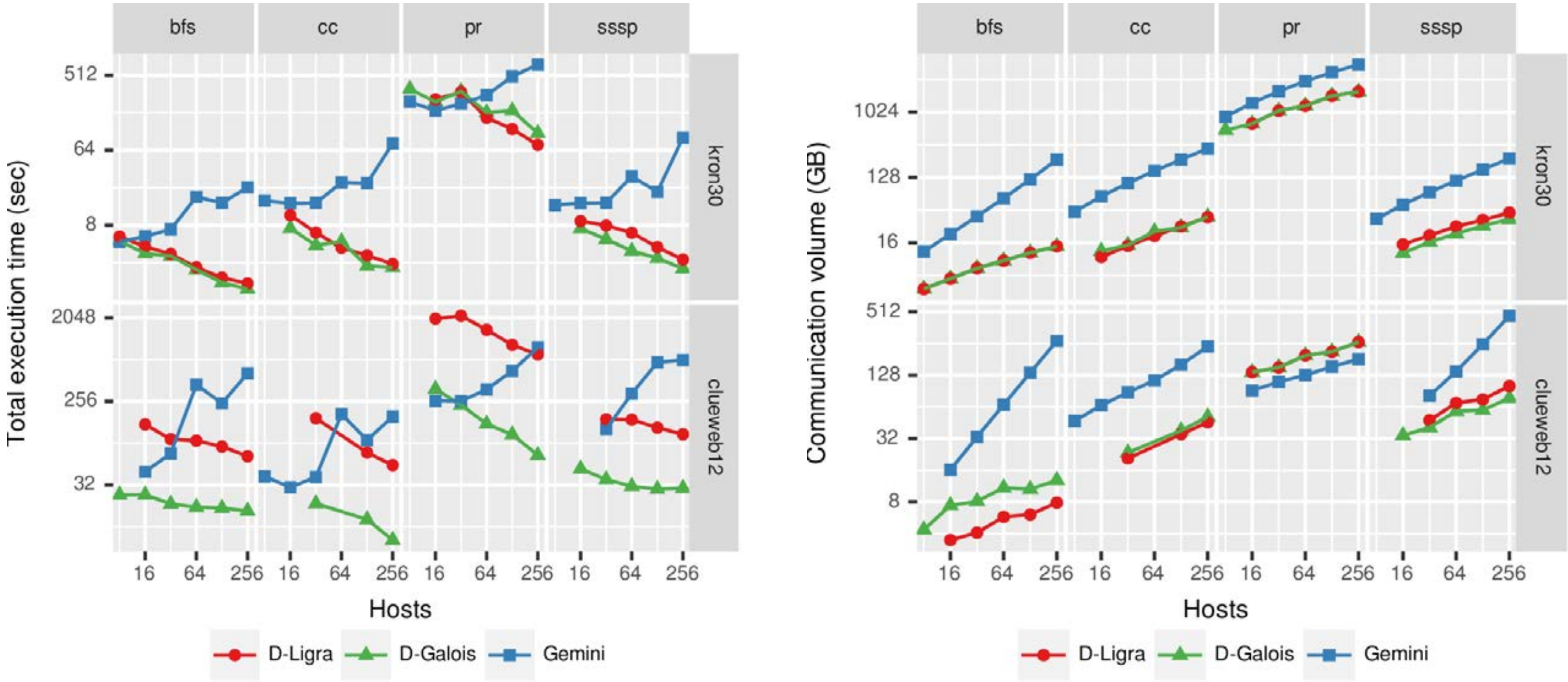
Experimental setup

- **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)

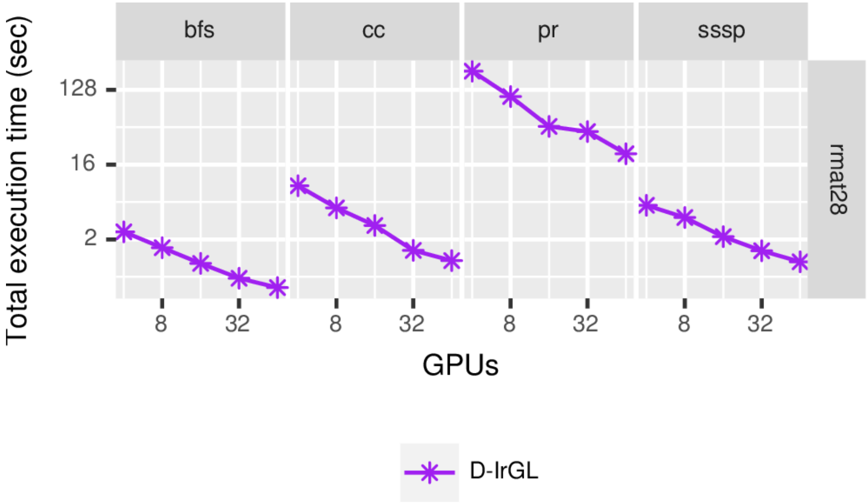
Inputs	rmat28	kron30	clueweb12	wdc12
|V|	268M	1073M	978M	3,563M
|E|	4B	11B	42B	129B
|E|/|V|	16	16	44	36
Size (CSR)	35GB	136GB	325GB	**986GB**

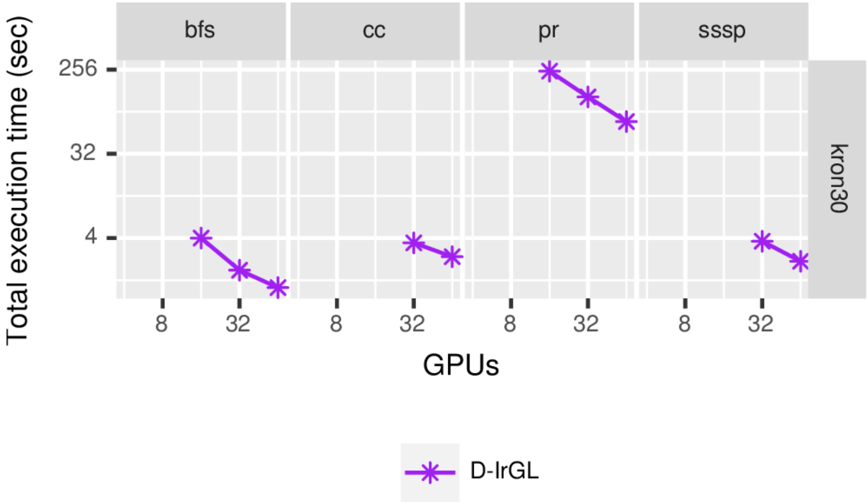
Clusters	Stampede (CPU)	Bridges (GPU)
Max. hosts	**256**	**64**
Machine	Intel Xeon Phi KNL	4 NVIDIA Tesla K80s
Each host	272 threads of KNL	1 Tesla K80
Memory	96GB DDR3	12GB DDR5

Strong scaling on Stampede (68 cores on each host)



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

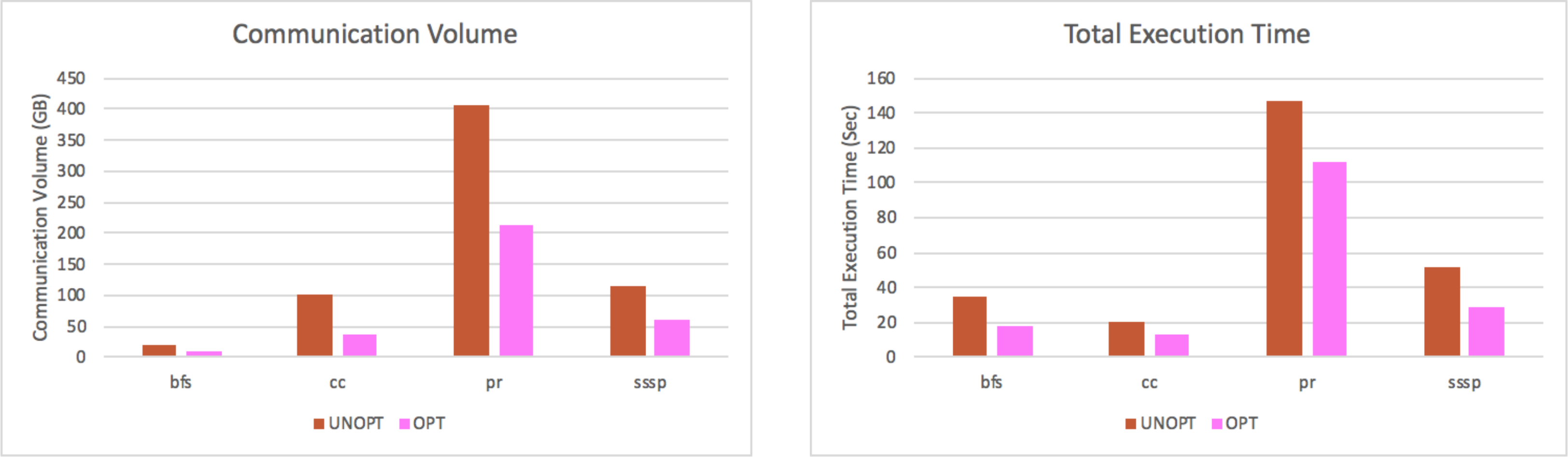




D-IrGL scales well

Impact of Gluon's communication optimizations

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

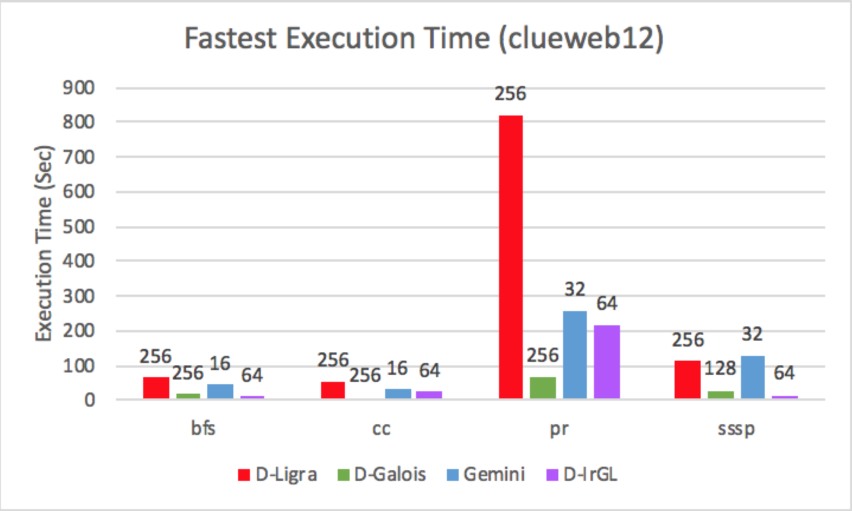


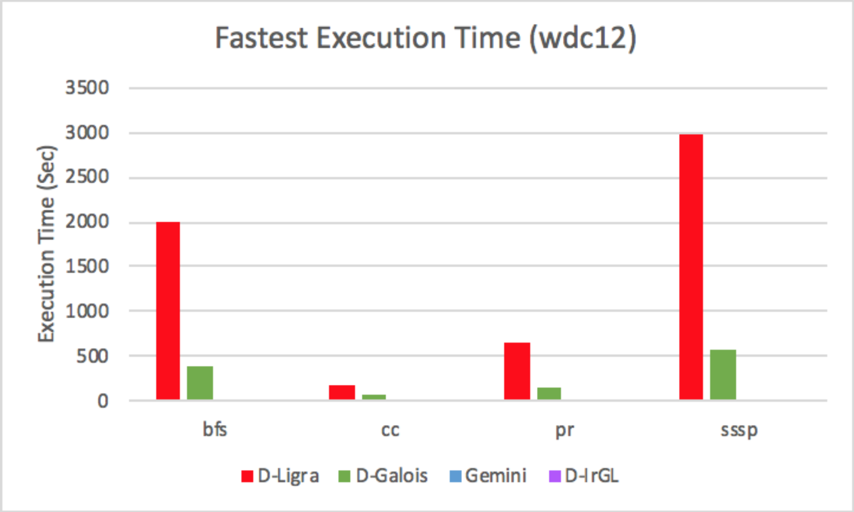
Improvement (geometric mean):

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