Combining Data Duplication and Graph Reordering to Accelerate Parallel Graph Processing

Vignesh Balaji and Brandon Lucia
CMU

Published at HPDC'19

Presented by Victor Ying

6.886 – May 6, 2021

The problem

- Consider algorithms that traverse (some or all) edges on each round to propagate values from source to destination vertices.

Algorithm 1 Typical graph processing kernel

```
1: par_for src in Frontier do
2:     for dst in out_neigh(src) do
3:         AtomicUpd (vtxData[dst]), auxData[src])
```

- We want to use the cache hierarchy effectively.
- We want to reduce contention (cache-line ping-ponging) and overheads associated with atomic memory operations.

Prior work: Switching direction of edge traversal (see Feb. 23 lecture)

- Traversing edges from sources to destinations is called push (a.k.a. top-down).
- Switching to pull (a.k.a. bottom-up) solves the problem of atomics & contention, but is **not work efficient**.

Algorithm 2 Pull version of graph kernel

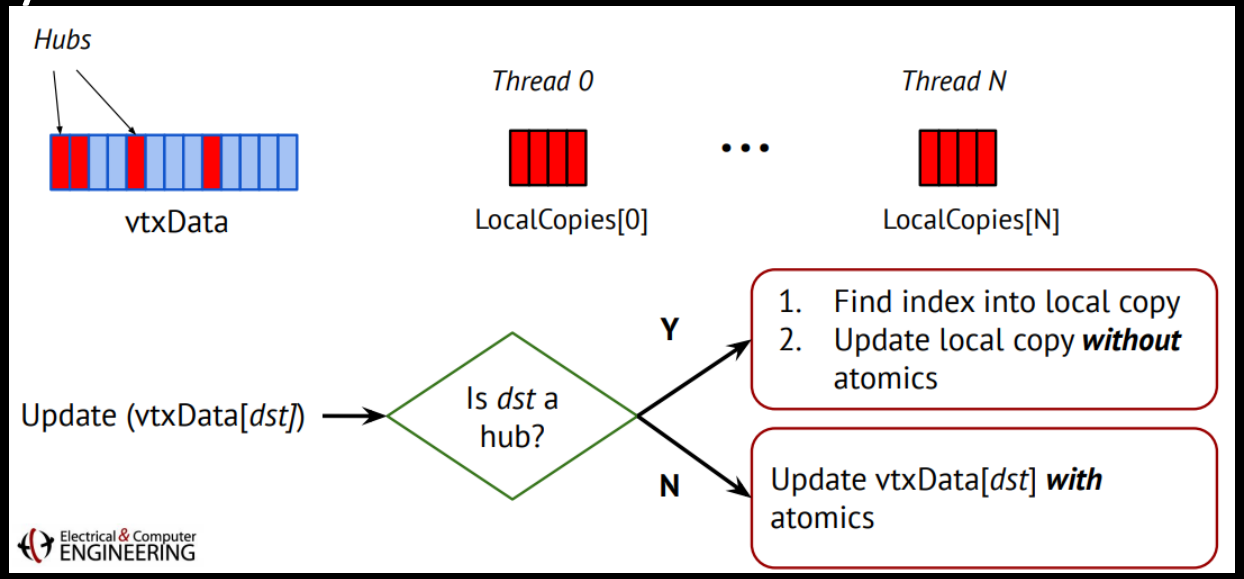
```
1: par_for dst in G do
2:     for src in in_neigh(dst) do
3:         if src in Frontier then
4:             Upd (vtxData[dst]), auxData[src])
```

Solution attempt #1: Naïve Duplication

- Assumes that the updates are associative and commutative. Think of doing a parallel reduction.
- Duplicate all destination vertex data so that each thread performs updates on its local copy. No need for atomics!
- Also known as “privatization”.
- Problem: Enormous memory overhead

Solution attempt #2: Selective Duplication (HubDup)

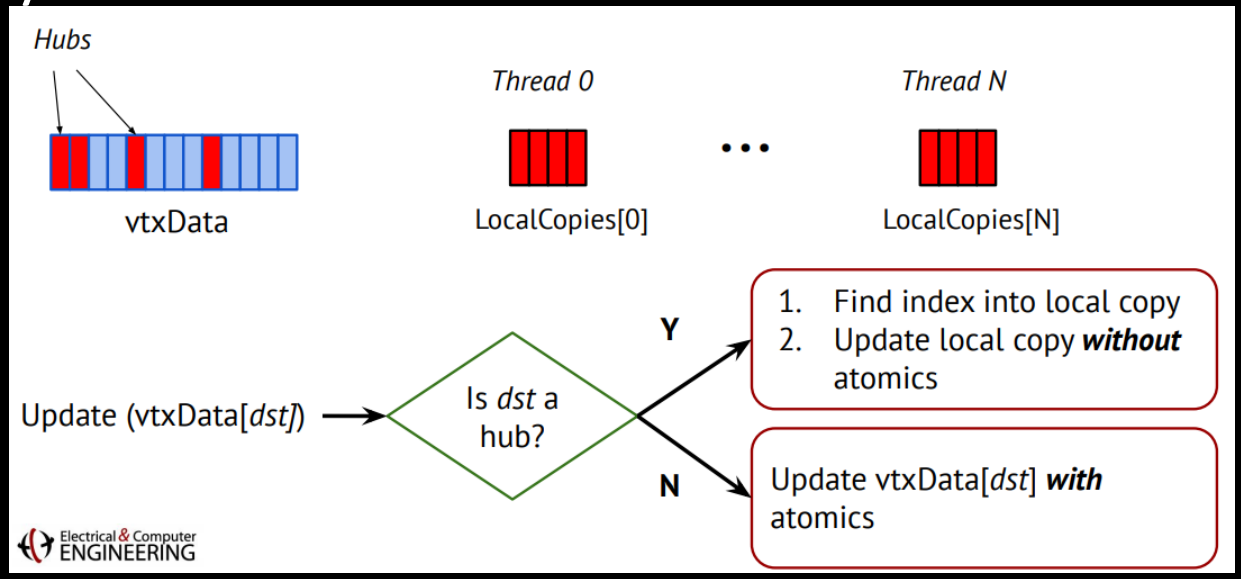
- Observation for some real-world graphs:
  - Most nodes have low degree, negligible contention.
  - A small fraction of nodes are “hubs”: these are updated many times.
- So privatize only the vertex data associated with hubs!



- Expensive overheads: lookup in a an extra data structure when visiting each vertex to figure out if it’s a “hub” to index into the tread-local copies.

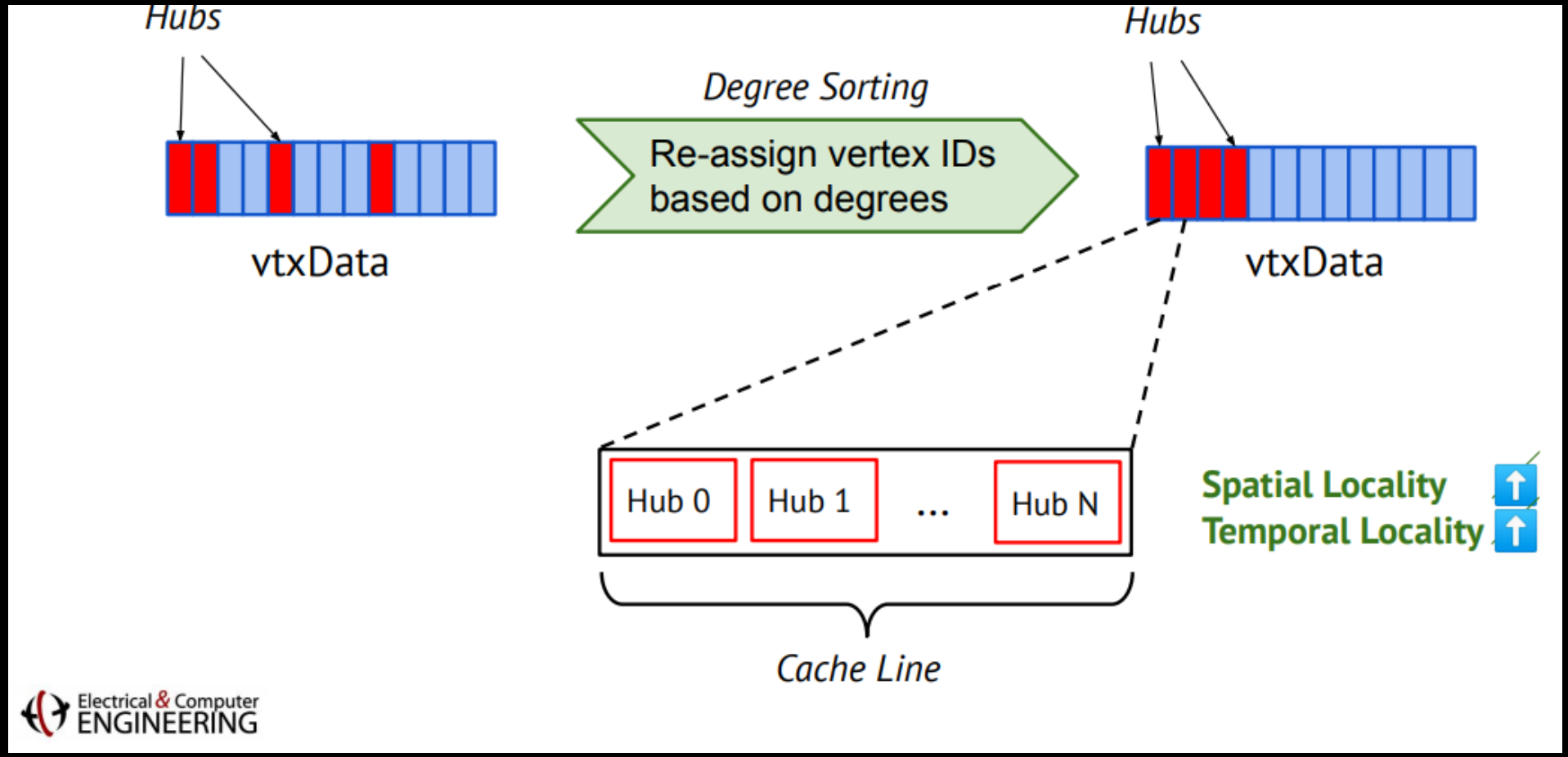
Ultimate solution: RADAR = HubDup + degree sorting

- Observation for some real-world graphs:
  - Most nodes have low degree, negligible contention.
  - A small fraction of nodes are “hubs”: these are updated many times.
- So privatize only the vertex data associated with hubs!

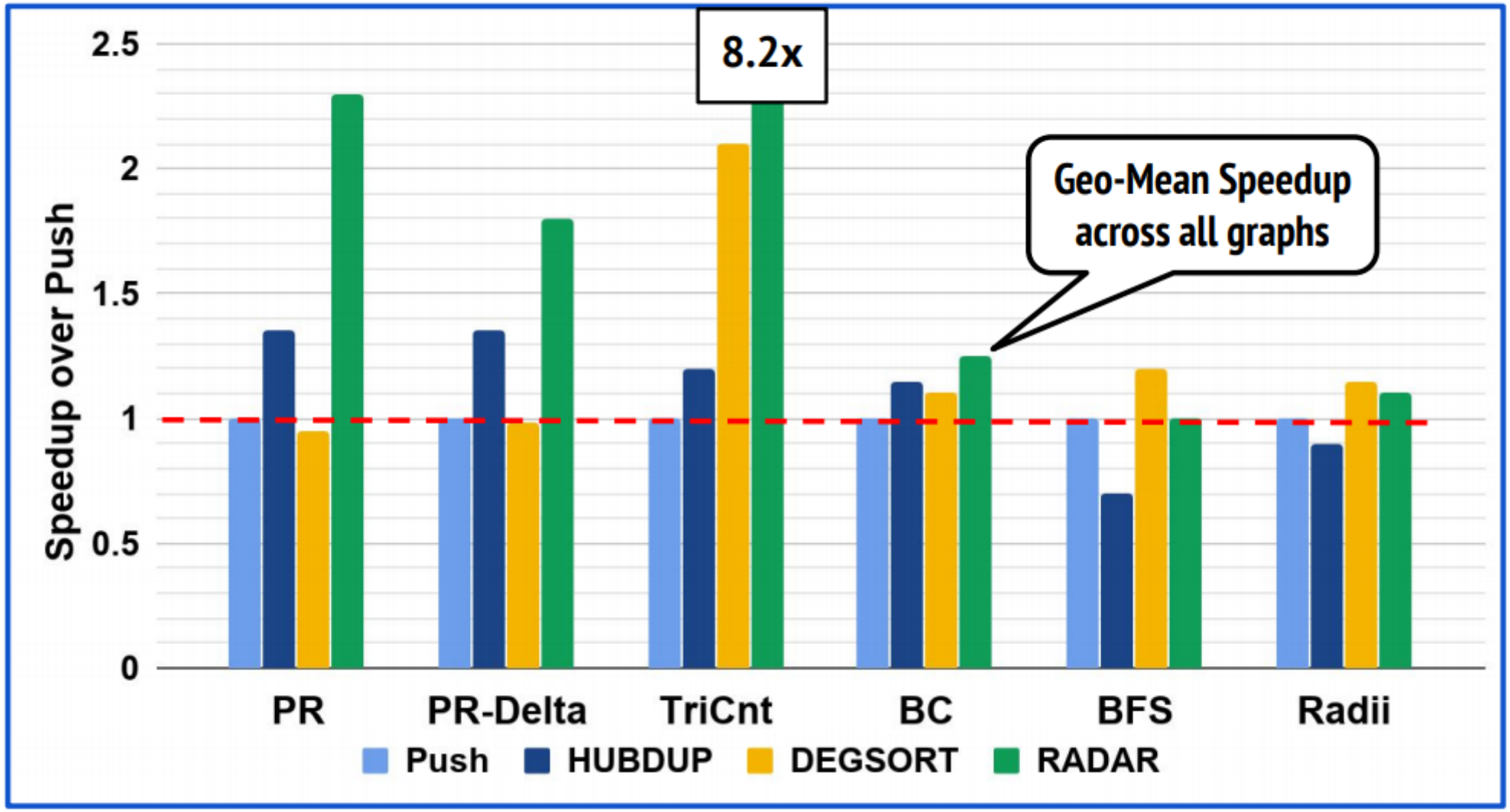


- Preprocess the graph with X hubs such that:
  - Hubs have vertex IDs 0 … X-1. Non hubs have IDs X, …, N-1

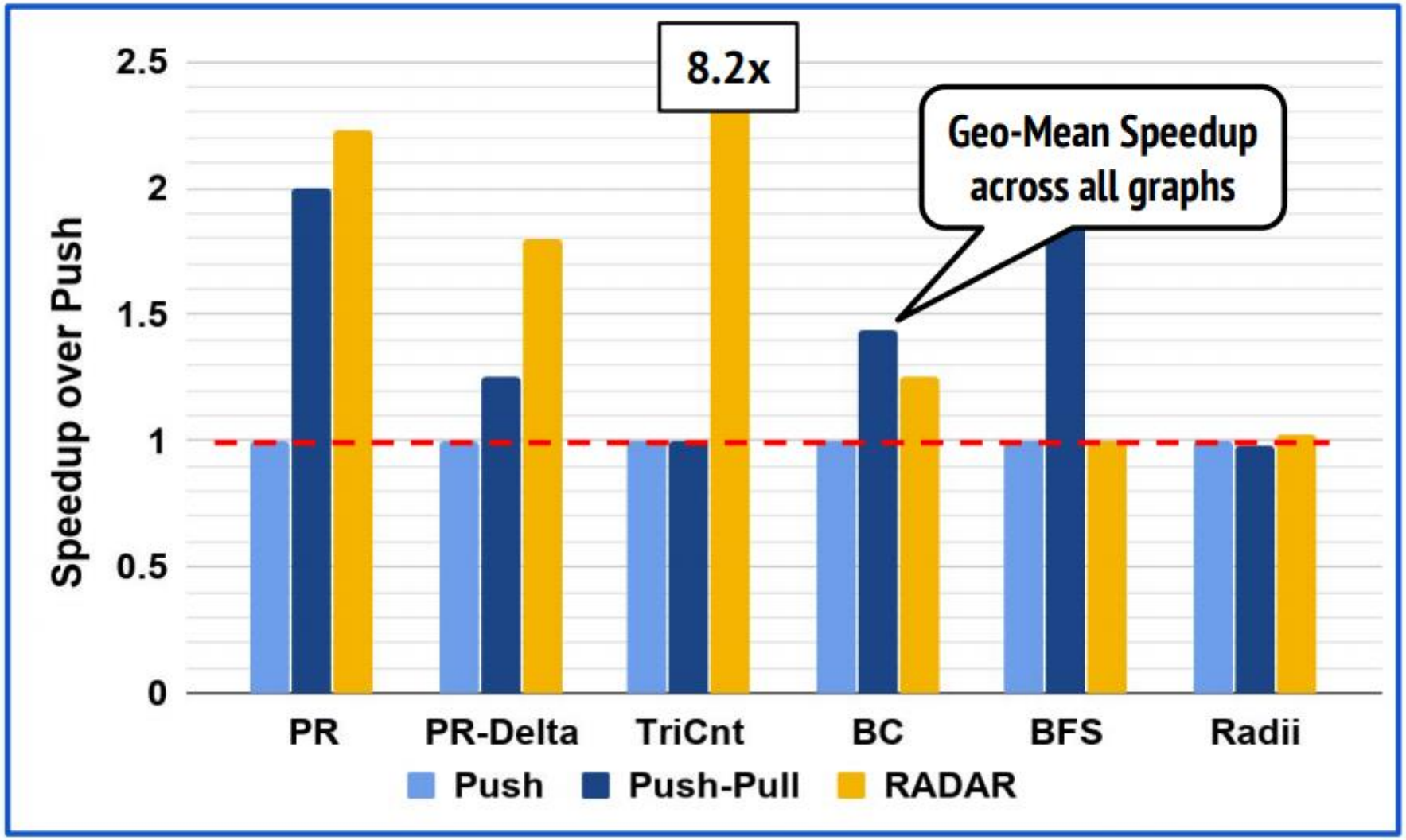
RADAR's degree sorting improves cache capacity efficiency too!



RADAR Outperforms Both HUBDUP And Degree Sorting



Performance Of RADAR Compared To Push-Pull



Conclusions

- RADAR (hub duplication + degree-based vertex partitioning) reduces the overheads of cache-line ping-ponging and atomics for power-law graphs
- This is an alternative to direction-switching (push-pull), sometimes one is better than the other.
- Note: the graph must be preprocessed to do the degree-based sorting/partitioning: this overhead is non-trivial if you're not going to run many rounds of computation on the same graph.
- Note: vertices need not be strictly degree-sorted. After partitioning into hubs and non-hubs, you could use another reordering heuristic to reorder the hubs among themselves.
- See also on hardware support to do an even better version of this, without needing degree-based sorting/partitioning:
  - Anurag Mukkara, Nathan Beckmann, Daniel Sanchez. "PHI: Architectural Support for Synchronization- and Bandwidth-Efficient Commutative Scatter Updates," in MICRO-52, October 2019.