Communication-efficient Massively Distributed Connected Components

IPDPS 2022 · May 31, 2022
Sebastian Lamm and Peter Sanders
Distributed Connected Components

- \( G = (V, E) \) undirected, \( V = 1..n, E \subseteq \binom{V}{2} \)
- Vertices \( u, v \in V \) connected \( \iff \exists \text{ path } \langle u, \ldots, v \rangle \) in \( G \)
- Connected component (CC) is maximal \( C \subseteq V \) of connected vertices
- Label \( l(u) = l(v) \iff u, v \in V \) in same CC
- \( G \) is distributed over \( p \) processing elements (PEs) 1..\( p \)

![Diagram showing distributed connected components with labels for PE 1 and PE 2.](image)

- \( l(\text{orange}) = 0 \)
- \( l(\text{blue}) = 1 \)
- \( l(\text{red}) = 2 \)
Applications

- Key building block for processing massive graphs
- Important part of data mining and graph clustering

- Few highly engineered scalable graph algorithms (e.g. Graph500)
- Phenomena like contention and latency increasingly important

Aim to reduce bottleneck communication volume
Communication-Efficiency

Communication Model

- **Point-to-point** communication
- Cost for message of length $\ell$ is $\alpha + \beta \cdot \ell$
- **Startup overhead**
- **Proportionality factor (Bandwidth)**
- **Rounds** of computation and communication

Communication-efficiency

- Bottleneck communication volume $\propto$ cut size $C = \sum_{1..p} C_i$

![Diagram showing communication between PE 1 and PE 2 with $C_1 = 3$, $C_2 = 3$, and $C = C_1 + C_2 = 6$.]
Linear Work Decomposition

- **Low-Diameter Decomposition** [Miller et al., 2013]
  - Partition vertices such that partitions have low diameter
  - Minimize number of edges between partitions
  - Diameter $O(\log n/\beta)$ and $O(\beta m)$ cut edges ($0 < \beta < 1$)

- CCs via recursive decomposition [Blelloch et al., 2014]

Work $O(m)$ and depth $O(\log^2 n/\beta)^*$

* on CRCW PRAM
Algorithm Overview (KaCC)

**Input Distribution:** Assign $|E|/p$ edges to each PE

**Output:** Label $\ell$ for each vertices $v \in V$
Algorithm Overview (KaCC)

**Input Distribution:** Assign $|E|/p$ edges to each PE

**Output:** Label $\ell$ for each vertices $v \in V$

1. Two rounds of **local contraction** with local BFS
Algorithm Overview (KaCC)

**Input Distribution:** Assign $|E|/p$ edges to each PE

**Output:** Label $\ell$ for each vertices $v \in V$

1. Two rounds of **local contraction** with local BFS
Algorithm Overview (KaCC)

**Input Distribution:** Assign $|E|/p$ edges to each PE

**Output:** Label $\ell$ for each vertices $v \in V$

1. Distribute high degree vertices
Algorithm Overview (KaCC)

Input Distribution: Assign $|E|/p$ edges to each PE
Output: Label $\ell$ for each vertices $v \in V$

2. Distribute high degree vertices
Algorithm Overview (KaCC)

**Input Distribution:** Assign $|E|/p$ edges to each PE
**Output:** Label $\ell$ for each vertices $v \in V$

3. Exponential decomposition using LP
Algorithm Overview (KaCC)

**Input Distribution:** Assign $|E|/p$ edges to each PE

**Output:** Label $\ell$ for each vertices $v \in V$

4. **Contract partitions**
Algorithm Overview (KaCC)

Input Distribution: Assign $|E|/p$ edges to each PE
Output: Label $\ell$ for each vertices $v \in V$

4. Contract partitions
Algorithm Overview (KaCC)

**Input Distribution:** Assign $|E|/p$ edges to each PE

**Output:** Label $\ell$ for each vertices $v \in V$

5. **Gather on root** and uncontract
Algorithm Overview (KaCC)

Input Distribution: Assign $|E|/p$ edges to each PE
Output: Label $\ell$ for each vertices $v \in V$

5. Gather on root and uncontract
Communication-Efficient?

- Empirically engineered algorithm design
- Powerful communication-efficient initial contractions
- However: Decomposition not in the worst case

Remedy (Outline)

- Use BFS (bounded to logarithmic depth) instead of LP
- Residual graph shrinks geometrically (in number of edges)
- Redistribute graph after decomposition
Experimental Setup

Competitors

- LACC by Azad and Buluç (IPDPS 2019)
- ParConnect by Flick et al. (SC 2015)
- KaCC

Instances

- Synthetic instances from KaGen generator (weak scaling)
  - 2D grids, RGG, RHG, RMAT
- Large real-world instances (strong scaling)

Hardware

- Up to 16384 cores of SuperMUC-NG thin nodes
Weak Scaling

Grid($10^3, 10^3, 0.51$), RGG($2^{16}, 0.55 \sqrt{\log(n)/n}$), RHG($2^{16}, 8, 3.0$), RMAT($2^{16}, 2^{19}$)

Edges per second

Number of elements

Cut edges (%)

ParConnect, LACC, KaCC

Max. sends, Max. receives, Total cut
Strong Scaling

- orkut
- twitter
- friendster
- wiki-links-en
- uk-2002
- it-2004
- europe
- USA-road

Running time (s)

#PEs $p$

ParConnect, LACC, KaCC
Conclusion

- Case-study for massively parallel graph algorithms
- Exploit locality as much as possible
- Multiple efficient contraction procedures
- Orders of magnitude faster for graphs with (good) locality

Future Work

- Extend with shared-memory parallelization
- More scalable sparse all-to-all
- Practical fully communication-efficient implementation

Thank you!

Code available at https://github.com/sebalamm/components
Weak Scaling