Karlsruhe Institute of Technology

Engineering a Distributed-Memory Triangle Counting Algorithm

IPDPS 2023

Peter Sanders and Tim Niklas Uhl | May 18, 2023

Motivation

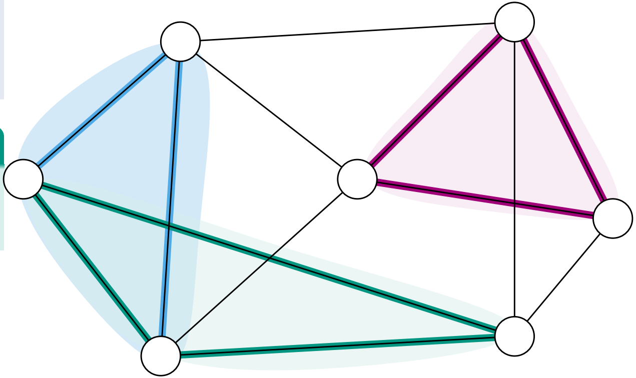
Problem Definition

- **Given:** undirected graph $G = (V, E)$
- **Output:** all **triangles** $\{u, v, w\} \subseteq V$ with $\{u, v\}, \{v, w\}, \{u, w\} \in E$.

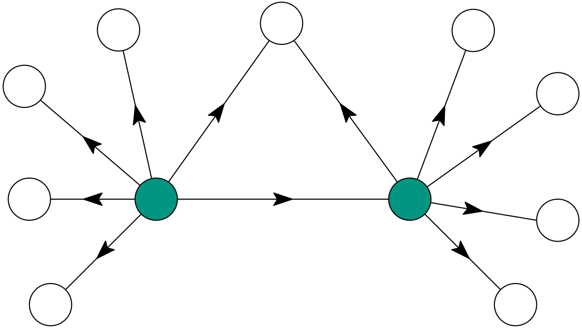
Applications

- graph analysis $\rightarrow$ **local clustering coefficient**
- spam detection, link recommendation, . . .

Goal: distributed algorithm scaling to billions of edges



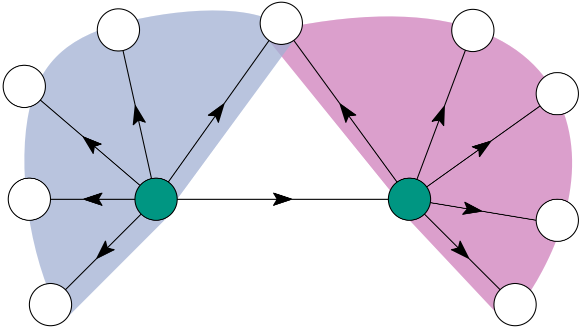
(Distributed) Triangle Counting



Sequential (Latapy 2006)

- **orient** edges
- iterate over all edges and **intersect** outgoing neighborhoods

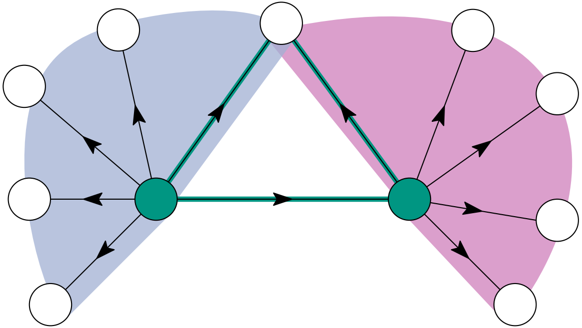
(Distributed) Triangle Counting



Sequential (Latapy 2006)

- **orient** edges
- iterate over all edges and **intersect** outgoing neighborhoods

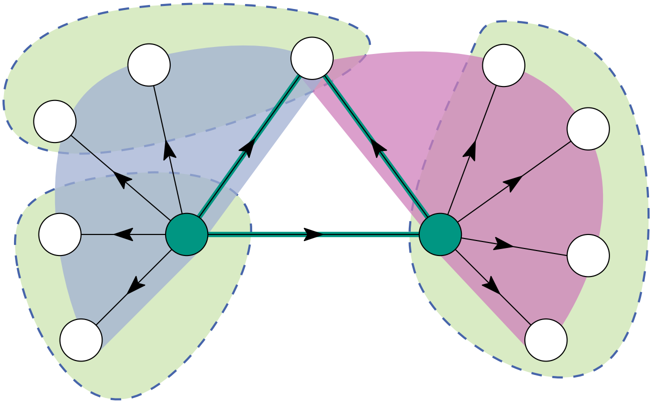
(Distributed) Triangle Counting



Sequential (Latapy 2006)

- **orient** edges
- iterate over all edges and **intersect** outgoing neighborhoods

(Distributed) Triangle Counting



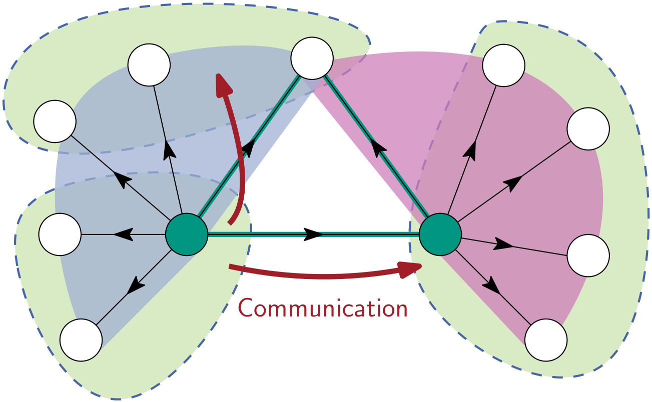
Sequential (Latapy 2006)

- **orient** edges
- iterate over all edges and **intersect** outgoing neighborhoods

Distributed Memory (Arifuzzaman et al. 2015)

- **1D partitioning** of the vertex set
- send neighborhoods to adjacents PEs

(Distributed) Triangle Counting



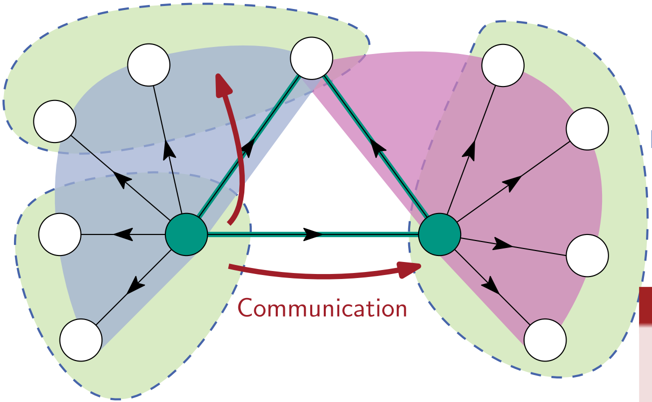
Sequential (Latapy 2006)

- **orient** edges
- iterate over all edges and **intersect** outgoing neighborhoods

Distributed Memory (Arifuzzaman et al. 2015)

- **1D partitioning** of the vertex set
- send neighborhoods to adjacents PEs

(Distributed) Triangle Counting



Sequential (Latapy 2006)

- **orient** edges
- iterate over all edges and **intersect** outgoing neighborhoods

Distributed Memory (Arifuzzaman et al. 2015)

- **1D partitioning** of the vertex set
- send neighborhoods to adjacents PEs

But

- superlinear communication volume
- volume dependent on neighborhood size
- irregular communication pattern

Algorithmic Building Blocks

Karlsruhe Institute of Technology

Point-to-point model

$$\alpha + \beta\ell$$

Algorithmic Building Blocks

Point-to-point model

Bandwidth

$$\alpha + \beta \ell$$

Startup overhead

Message length

Algorithmic Building Blocks

Bandwidth

Point-to-point model

$$\alpha + \beta \ell$$

Startup overhead

message buffering

indirect messaging

Message length

exploiting locality

Algorithmic Building Blocks

Bandwidth

Point-to-point model

$$\alpha + \beta \ell$$

Startup overhead

- limit buffer to **local input size**
- flush via **non-blocking** send on overflow

message buffering

indirect messaging

Message length

exploiting locality

Algorithmic Building Blocks

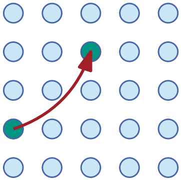
Bandwidth

Point-to-point model

$\alpha + \beta\ell$

Startup overhead

- limit buffer to **local input size**
- flush via **non-blocking** send on overflow



message buffering

indirect messaging

Message length

exploiting locality

Algorithmic Building Blocks

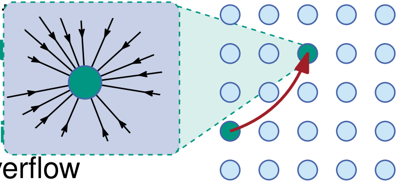
Point-to-point model

$\alpha + \beta\ell$

Bandwidth

Startup overhead

- limit buffer
  local input
- flush via
  non-block
  send on overflow



message buffering

indirect messaging

Message length

exploiting locality

Algorithmic Building Blocks

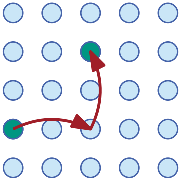
Bandwidth

Point-to-point model

$$\alpha + \beta\ell$$

Startup overhead

- limit buffer to **local input size**
- flush via **non-blocking** send on overflow



message buffering   indirect messaging

Message length

exploiting locality

Algorithmic Building Blocks

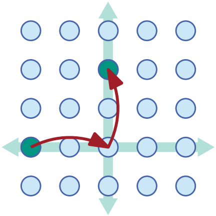
Point-to-point model

Bandwidth

$$\alpha + \beta\ell$$

Startup overhead

- limit buffer to **local input size**
- flush via **non-blocking** send on overflow



message buffering

indirect messaging

Message length

exploiting locality

Algorithmic Building Blocks

Point-to-point model

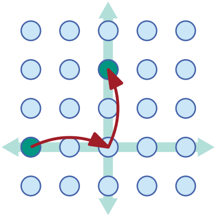
Bandwidth

$$\alpha + \beta \ell$$

Startup overhead

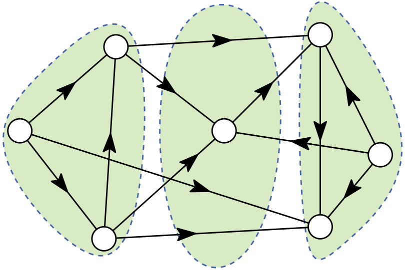
- limit buffer to **local input size**
- flush via **non-blocking** send on overflow

message buffering



indirect messaging

Message length



exploiting locality

Algorithmic Building Blocks

Karlsruhe Institute of Technology

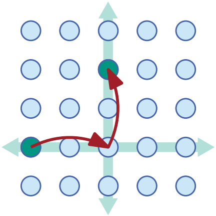
Point-to-point model

Bandwidth

$$\alpha + \beta\ell$$

Startup overhead

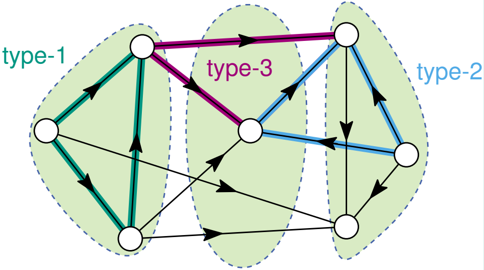
- limit buffer to **local input size**
- flush via **non-blocking** send on overflow



message buffering

indirect messaging

Message length



exploiting locality

Algorithmic Building Blocks

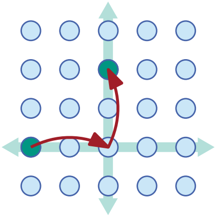
Point-to-point model

$\alpha + \beta\ell$

Bandwidth

Startup overhead

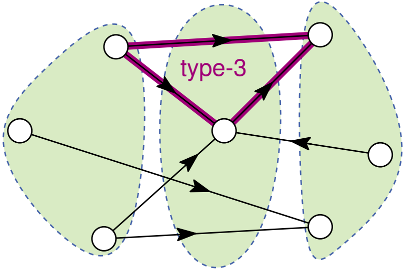
- limit buffer to **local input size**
- flush via **non-blocking** send on overflow



message buffering

indirect messaging

Message length



exploiting locality

Experimental Setup

Karlsruhe Institute of Technology

Competitors

based on MPI

- HavoqGT by Pearce et al. (HPEC 2019)
- TriC by Ghosh and Halappanavar (HPEC 2020)
- **ours** (four variants)

Instances

- Synthetic instances from KaGen generator (weak scaling)
  - RGG-2D, RHG, GNM, RMAT
  - up to 2^{37} edges
- Large real-world instances (strong scaling)

Hardware

- evaluated using up to 32 768 ($= 2^{15}$) cores of SuperMUC-NG thin nodes

Evaluation

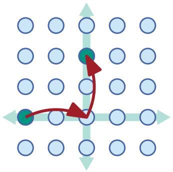
$\boxed{\alpha} + \beta\ell$

Startup overhead

message buffering

- limit buffer to **local input size**
- flush via **non-blocking** send on overflow

indirect messaging



Evaluation

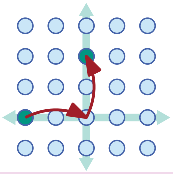
$$\boxed{\alpha} + \beta\ell$$

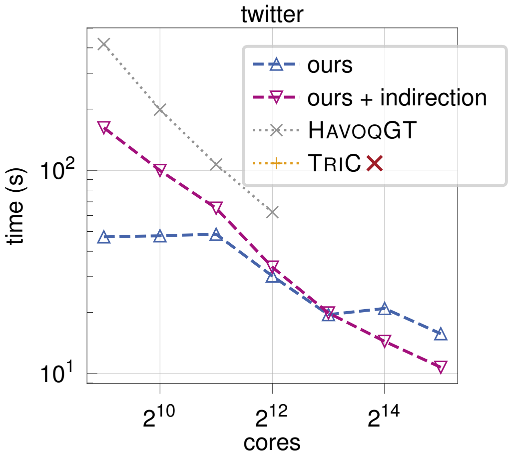
Startup overhead

message buffering

- limit buffer to **local input size**
- flush via **non-blocking** send on overflow

indirect messaging





Evaluation

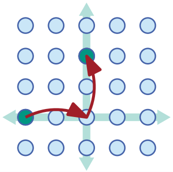
$\boxed{\alpha} + \beta\ell$

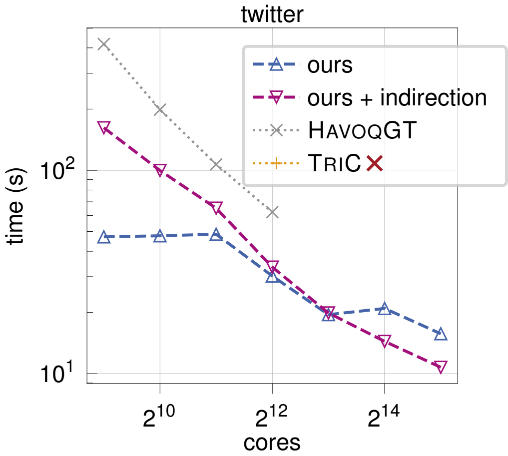
Startup overhead

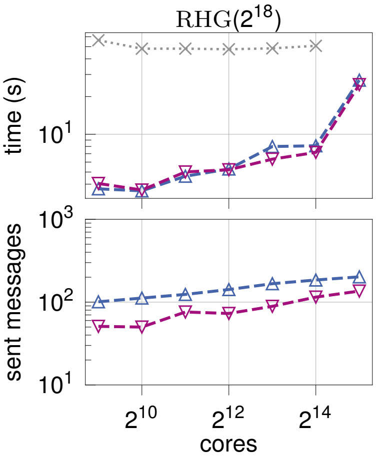
message buffering

- limit buffer to **local input size**
- flush via **non-blocking** send on overflow

indirect messaging

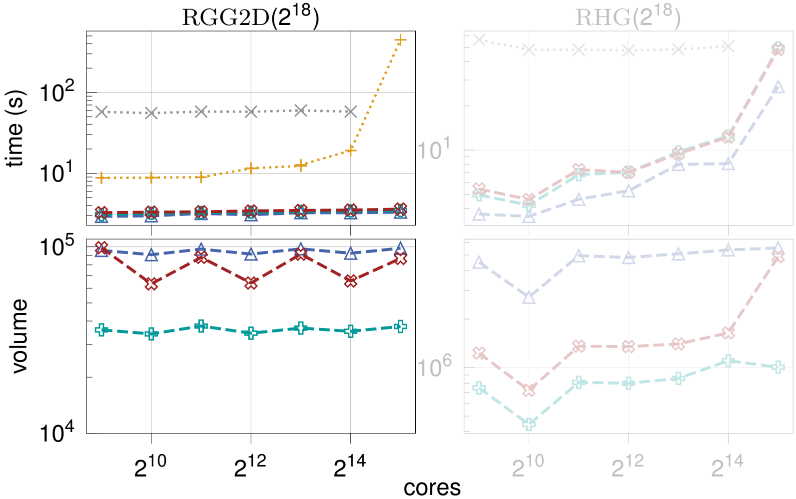


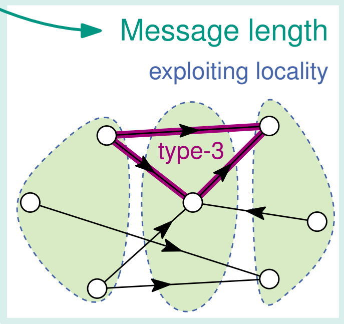


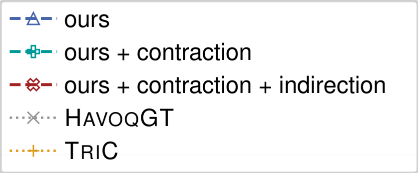


Evaluation

$\alpha + \beta \ell$

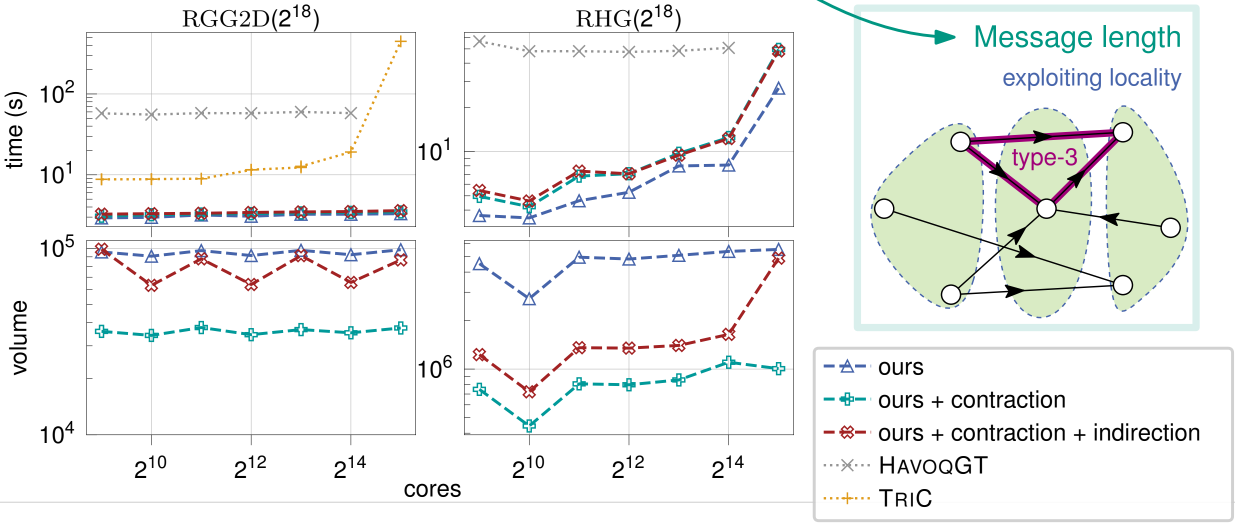






Evaluation

$\alpha + \beta\ell$



Conclusion

Karlsruhe Institute of Technology

Our contributions:

Massively Scalable Distributed Triangle Counting Algorithms

- **linear memory requirements** using non-blocking sparse all-to-all algorithm
- reduce **startup overhead** by message aggregation and indirect communication
- reduce **communication** by local computations
- up to **18**$\times$ better performance
- scalable to up to at least **32 768 cores**

niklas-uhl/katric

Future work:

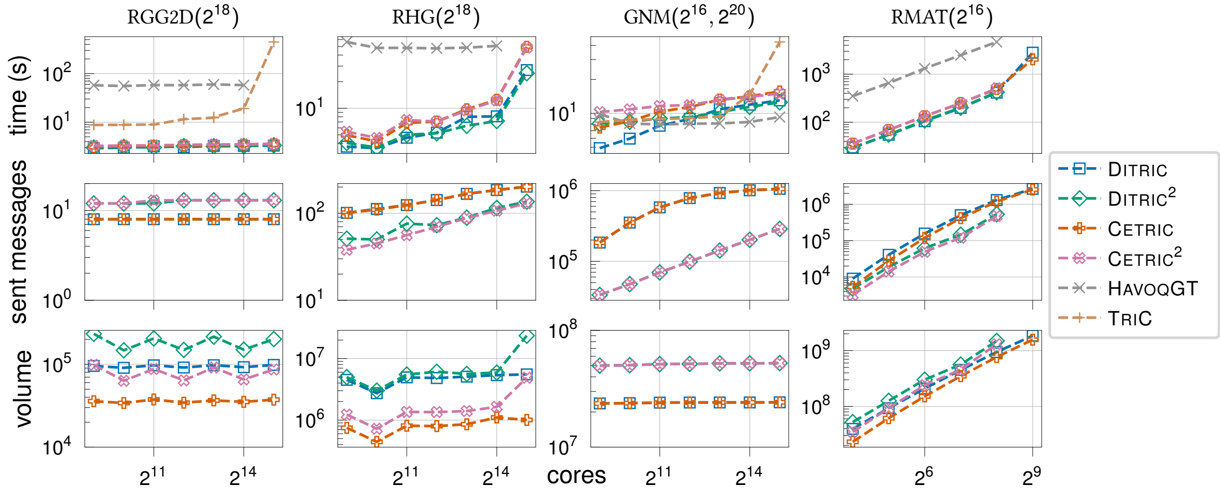
- engineered **hybrid** implementation
- use **communication primitives** as foundation for general purpose graph processing framework

European Research Council
Established by the European Commission

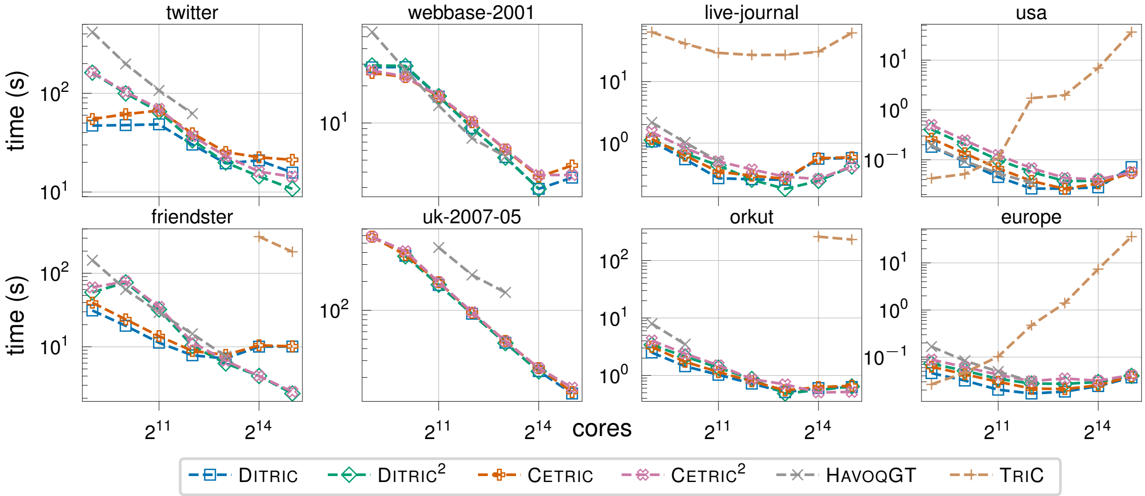
Appendix

Full Results

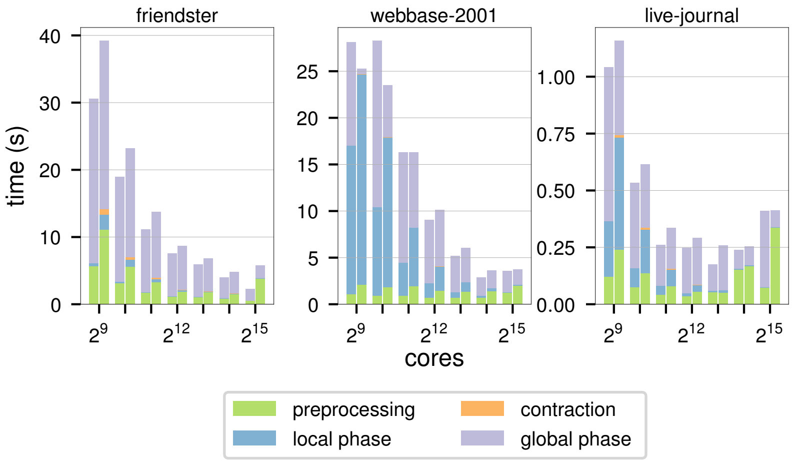
Full Weak Scaling Experiments



Full Strong Scaling Experiments



Phase Times


friendster
webbase-2001
live-journal
time (s)
cores
40
30
20
10
0
25
20
15
10
5
0
1.00
0.75
0.50
0.25
0.00
2^9
2^{12}
2^{15}
preprocessing
local phase
contraction
global phase

Hybrid Implementation