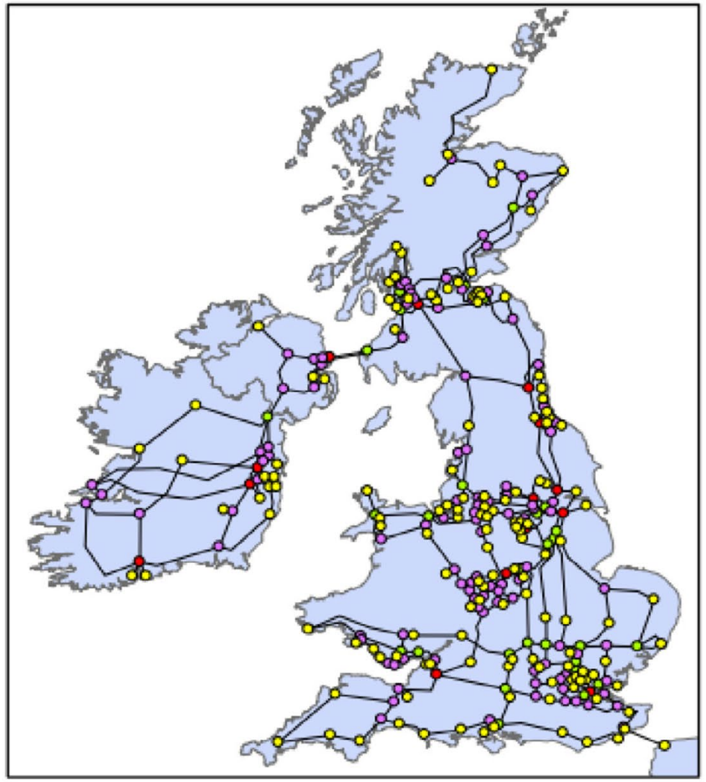
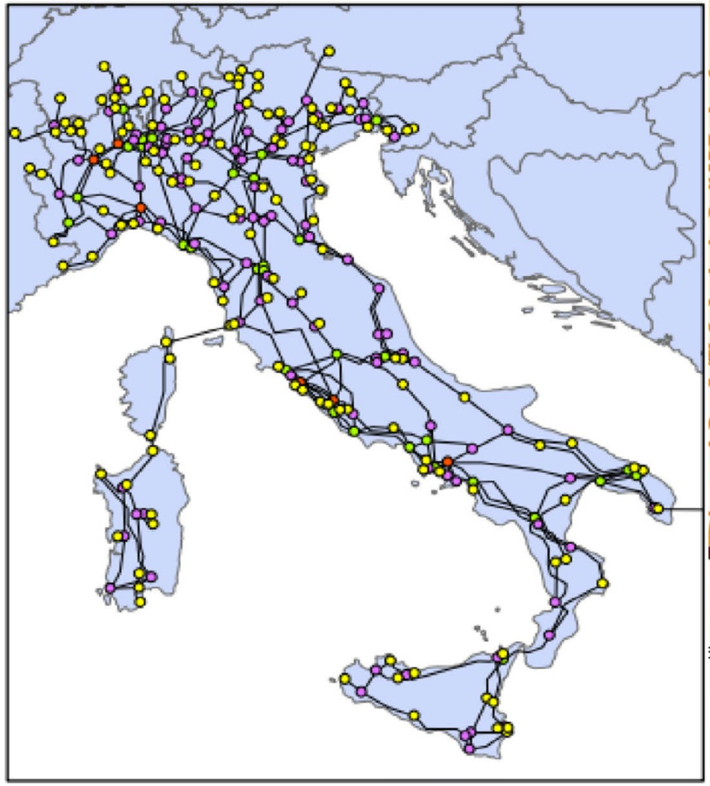
ANDRÁS STRAUSZ, FLAVIO VELLA (UNIVERSITY OF TRENTO), SALVATORE DI GIROLAMO,
MACIEJ BESTA, TORSTEN HOEFLER

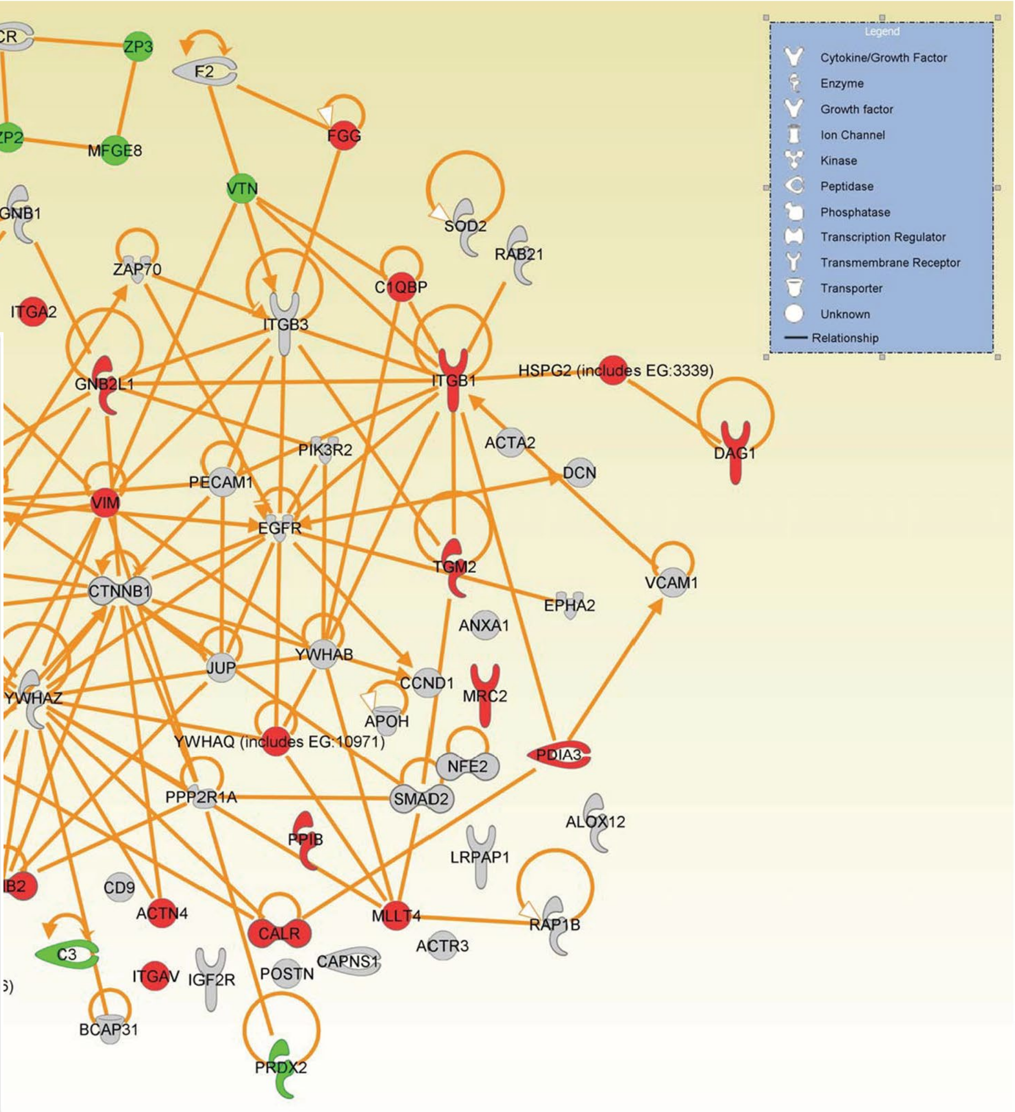
Asynchronous Distributed-Memory Triangle Counting and LCC with RMA Caching

Warm-up: Local Clustering Coefficient

- **Graphs represent relational data very well**



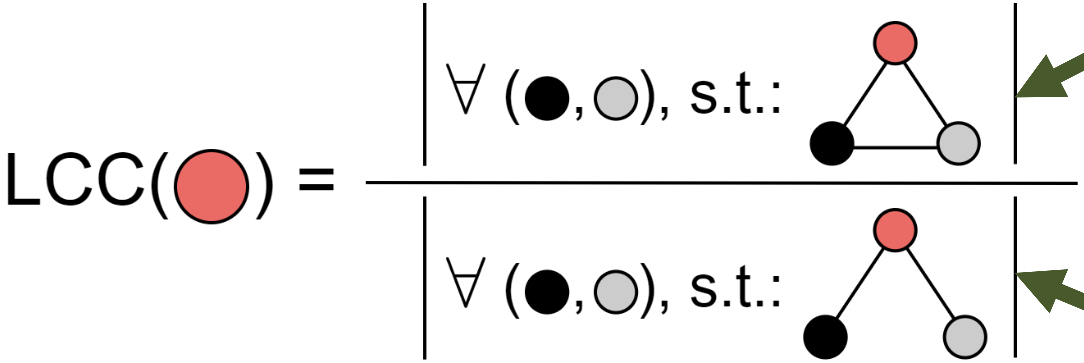




Right: Peddinti, Divyaswetha & Memili, Erdoğan & Burgess: Proteomics-Based Systems Biology Modeling of Bovine Germinal Vesicle Stage Oocyte and Cumulus Cell Interaction
Left: Alireza Shahpari, Mohammad Khansari, Ali Moeini: Vulnerability analysis of power grid with the network science approach based on actual grid characteristics: A case study in Iran

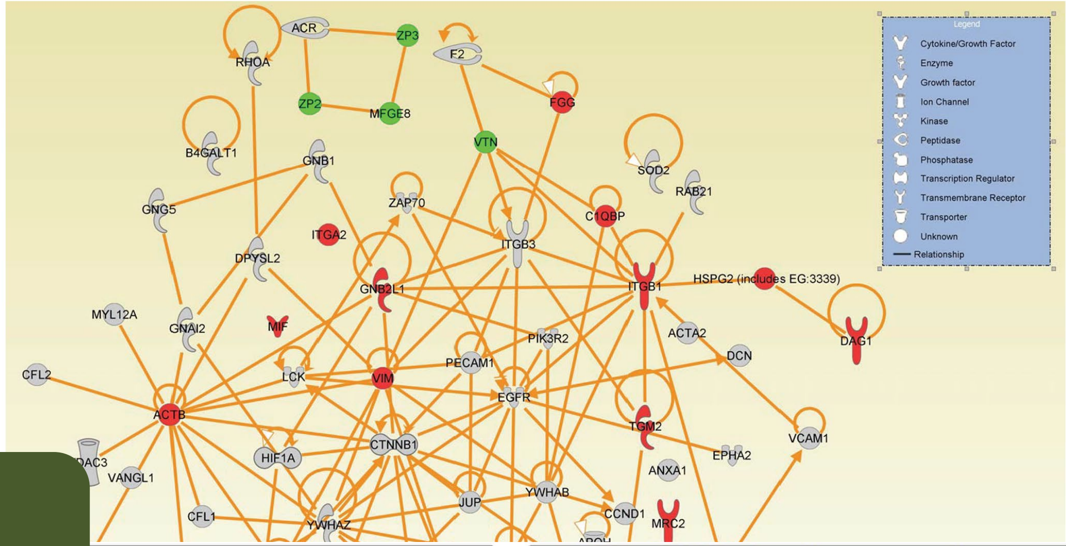
Warm-up: Local Clustering Coefficient

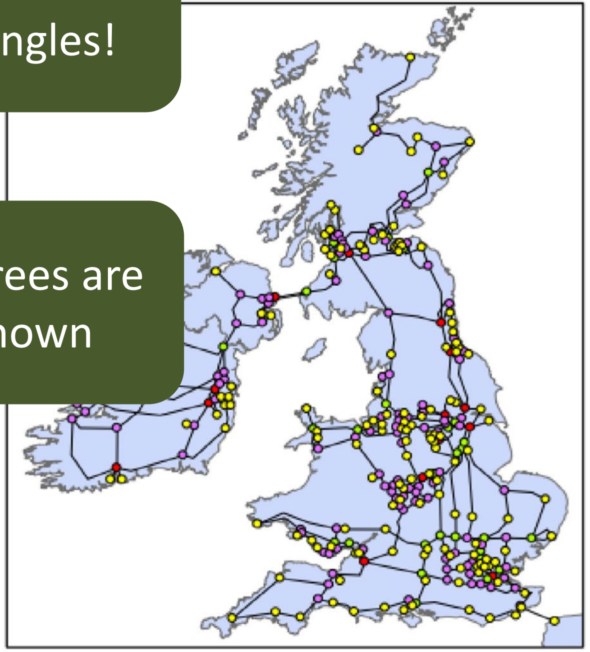
- **Graphs represent relational data very well**
- **LCC: likelihood that neighbors of a vertex are connected**

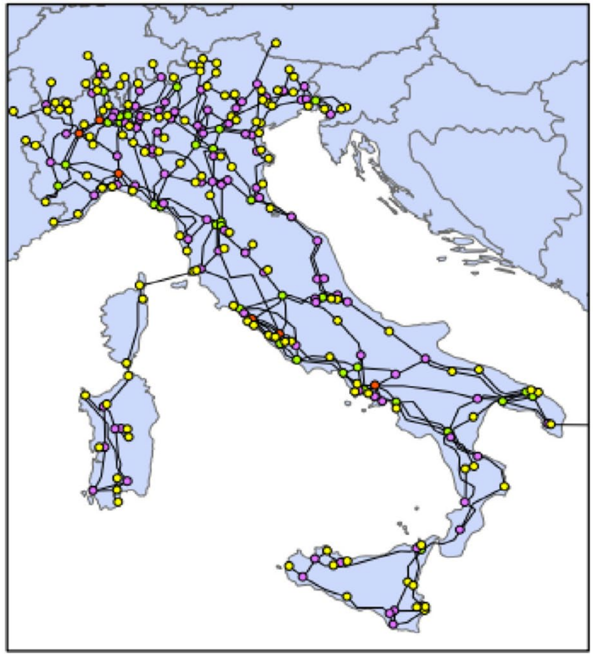


Count triangles!

Degrees are known

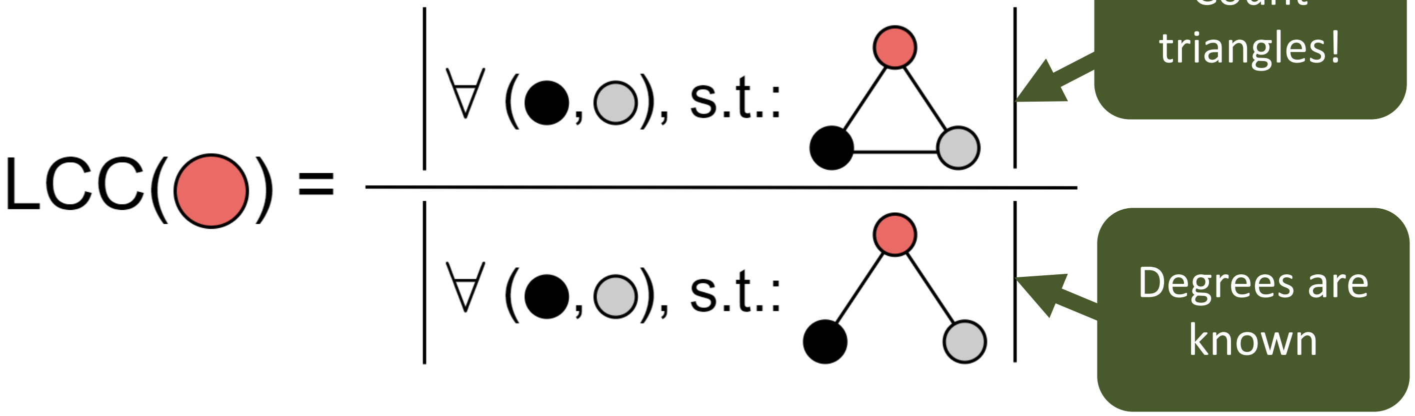


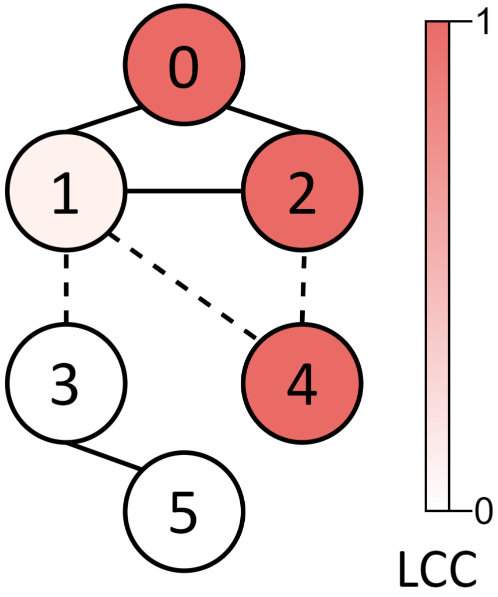




Warm-up: Local Clustering Coefficient

- **Graphs represent relational data very well**
- **LCC: likelihood that neighbors of a vertex are connected**

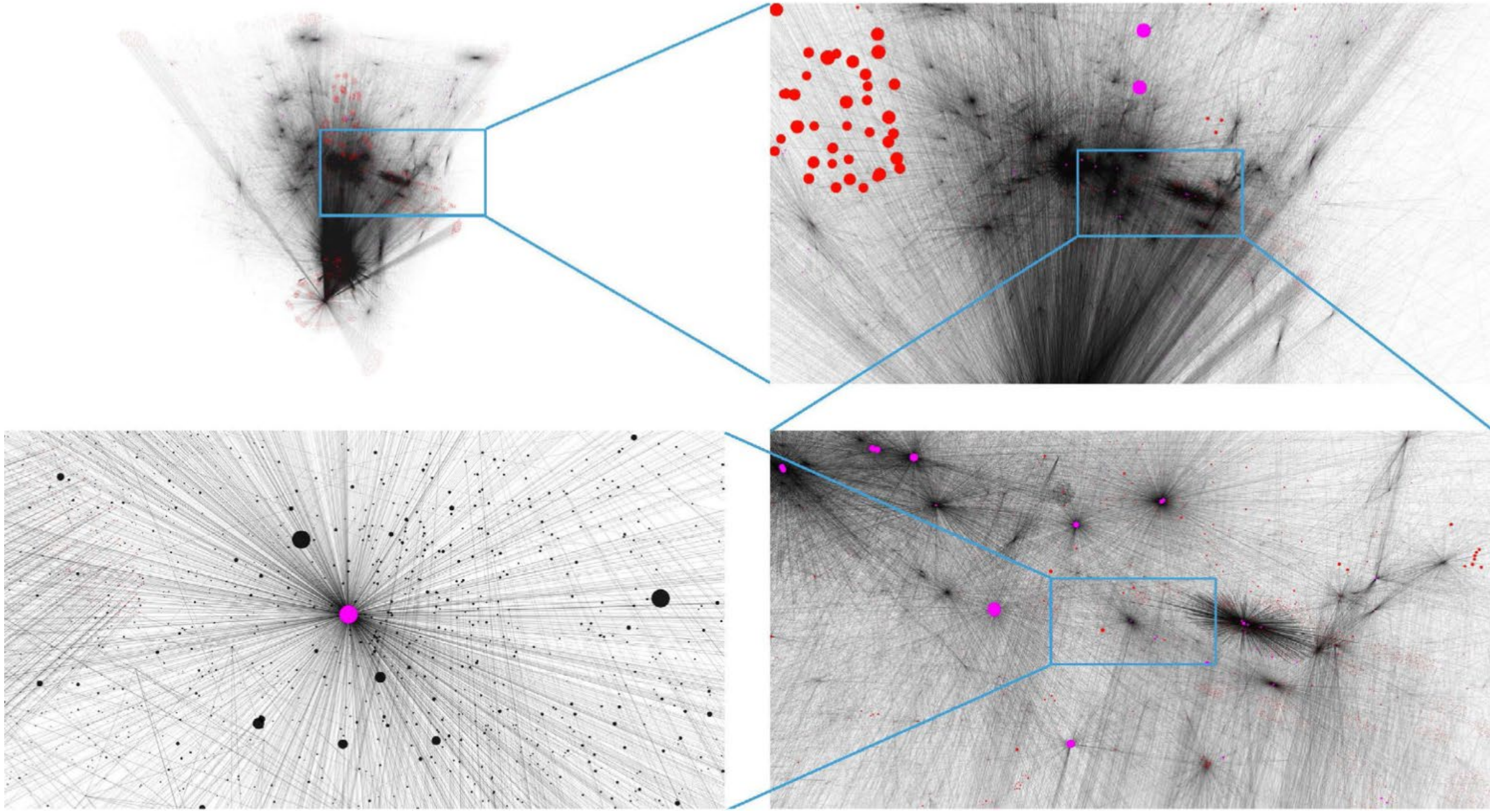




- **Many useful applications in link prediction problems**
  - community detection, link recommendation

Challenges: Graphs are huge and skewed

- Billions of vertices and hundreds of billions of edges
- Scale-free degree distribution

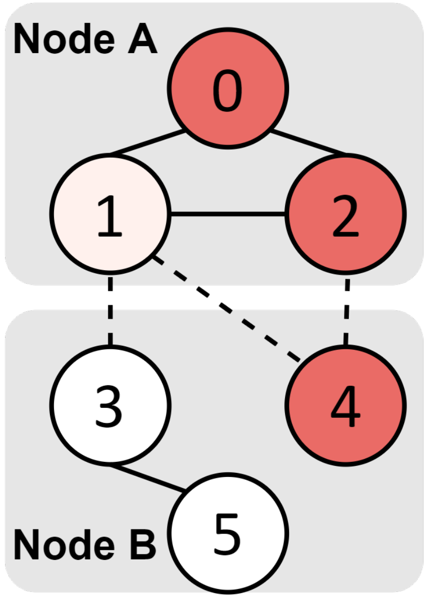


Right: Albert-László Barabási: Network Science (Chapter 4)

Distributed-memory TC & LCC computing

Current state-of-the-art:

1. **Synchronized computation**
   - Bulk Synchronous Parallel
   - MapReduce
2. **Frontier intersection**
3. **Graph partitioning**
   - Static vertex delegation



Our work proposes:

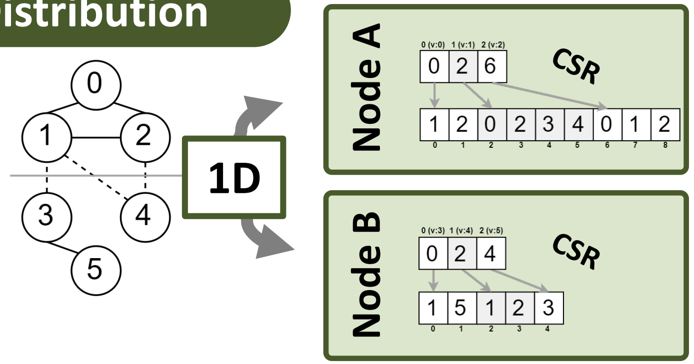
1. **Fully asynchronous algorithm based on MPI-RMA**
2. **Hybrid strategy for local TC**
3. **Exploiting data reuse with caching**

   Application-specific eviction policy

In general, 4-12x faster results for scale free graphs compared to TriC
Best results show up to 100x speedup

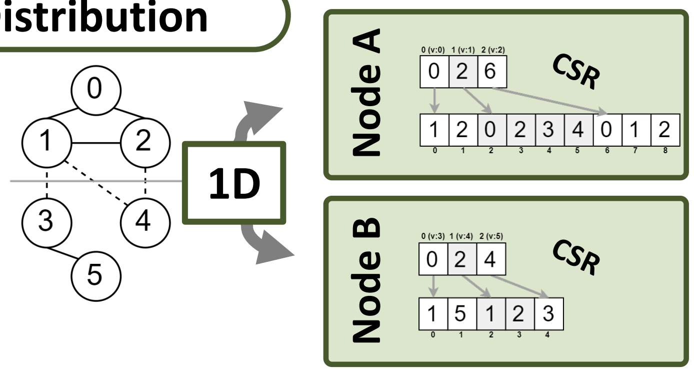
Algorithm overview: Distribution

1 Distribution

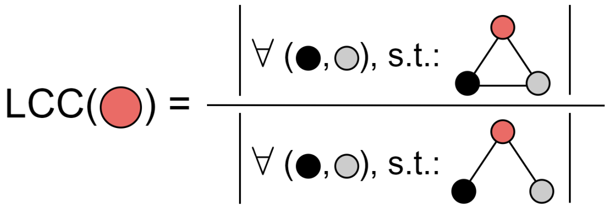


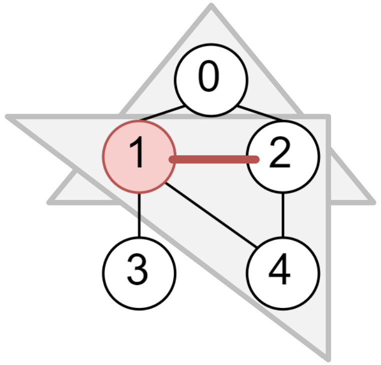
Algorithm overview: Shared memory computation

1 Distribution



2 Local TC computation





$\#\text{triangles} = |adj(v) \cap adj(w)|$

Binary search

$O(|adj(v)| \log(|adj(w)|))$

VS

$\frac{|adj(v)|}{|adj(w)|} > B$

Sorted Set Intersection

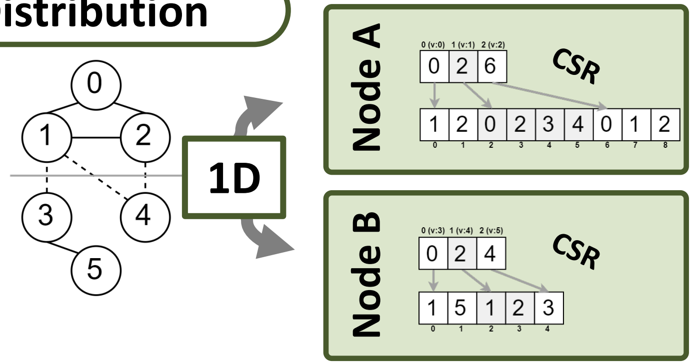
$O(|adj(v)| + |adj(w)|)$

Hybrid method decreases running time by up to 8%

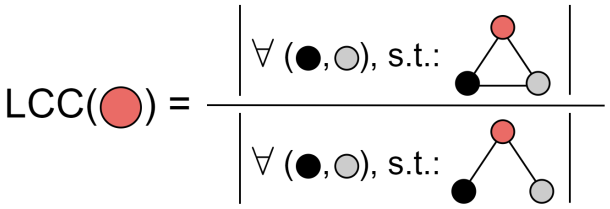
On shared memory 2.7x speedup using 16 threads

Algorithm overview: Shared memory computation

1 Distribution



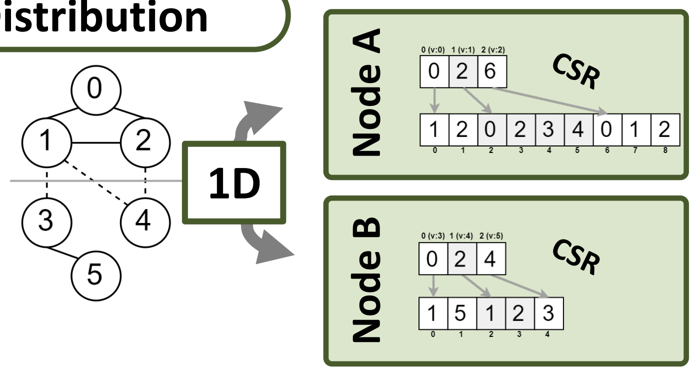
2 Local TC computation



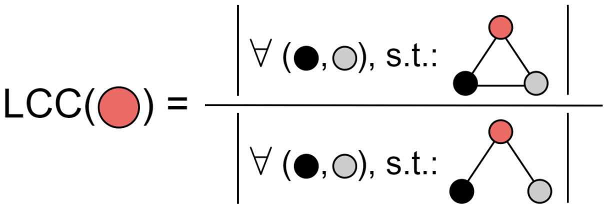
- **Shared memory parallel**
- **Hybrid method**

Algorithm overview: Distributed memory algorithm

1 Distribution



2 Local TC computation

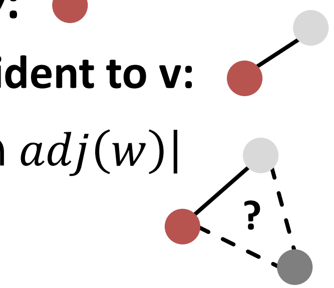


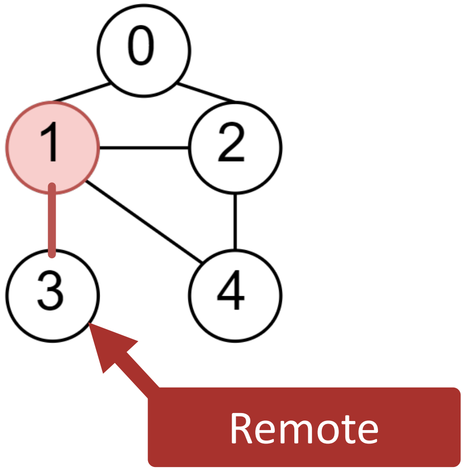
- **Shared memory parallel**
- **Hybrid method**

3 Asynchronous distributed memory algorithm

1. **For all local vertices *v*:**
   2. **For all vertices w incident to v:**

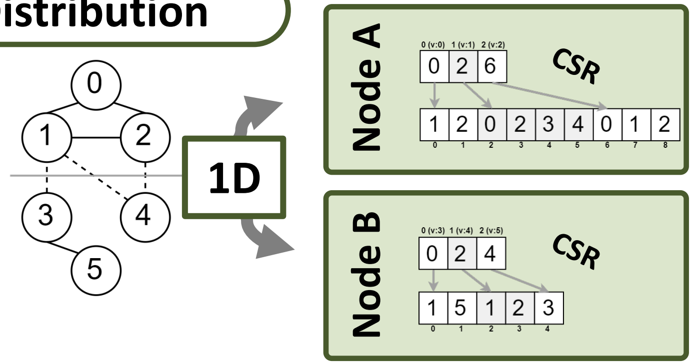
$\#\text{triangles} \mathrel{+}= |\, adj(v) \cap adj(w) |$



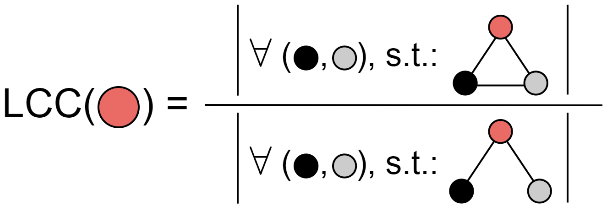


Algorithm overview: Distributed memory algorithm

1 Distribution



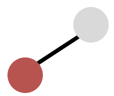
2 Local TC computation



- **Shared memory parallel**
- **Hybrid method**

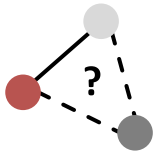
3 Asynchronous distributed memory algorithm

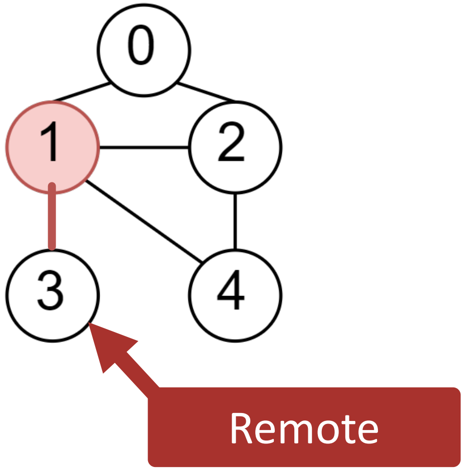
1. **For all local vertices *v*:**
   2. **For all vertices w incident to v:**



If w is remote: Get $adj(w)$

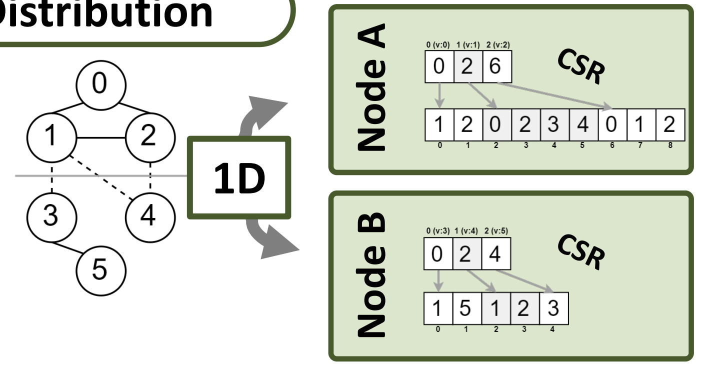
$\#\text{triangles} \mathrel{+}= |\, adj(v) \cap adj(w)|$



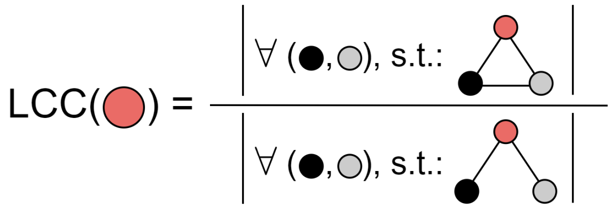


Algorithm overview: MPI-RMA

1 Distribution



2 Local TC computation



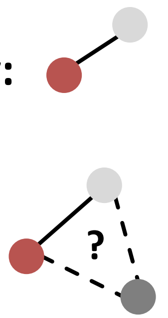
- **Shared memory parallel**
- **Hybrid method**

3 Asynchronous distributed memory algorithm

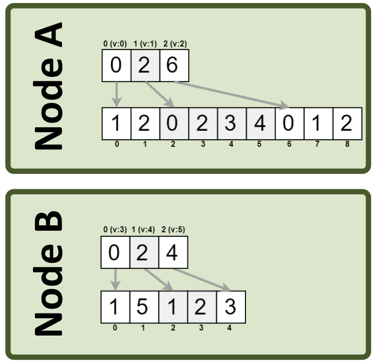
1. **For all local vertices *v*:**
   2. **For all vertices w incident to v:**

      If w is remote: Get $adj(w)$

   $\#\text{triangles} \mathrel{+}= |\,adj(v) \cap adj(w)|$

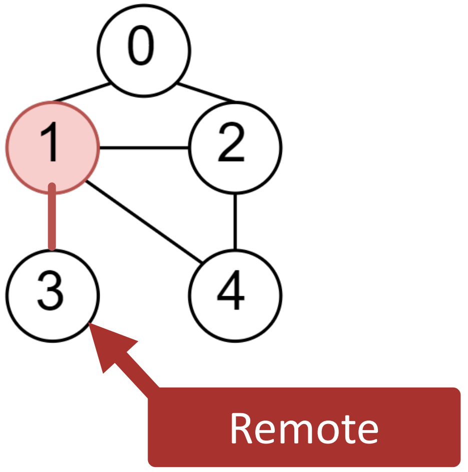


4 Communication



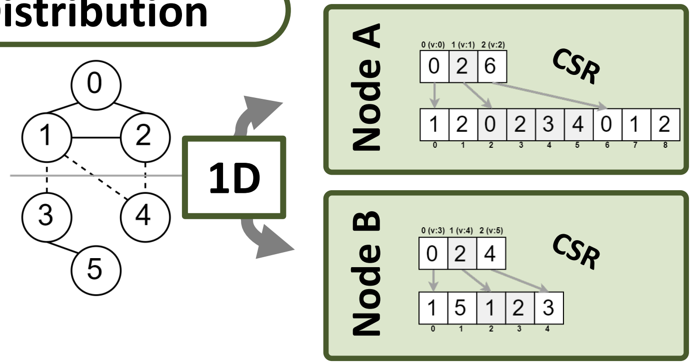
MPI-RMA

- **Non-blocking communication**
- **Target process is not involved**
- **Hardware support**
- **Core elements:**
  - MPI Window
  - MPI_Get(window, target, offset, size);

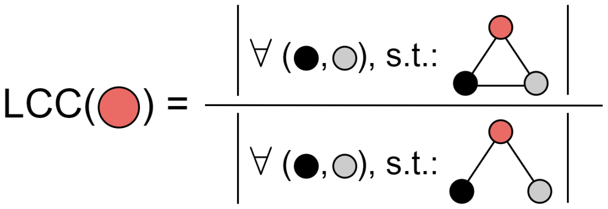


Algorithm overview: MPI-RMA

1 Distribution



2 Local TC computation

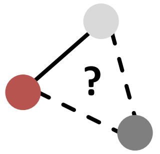


- **Shared memory parallel**
- **Hybrid method**

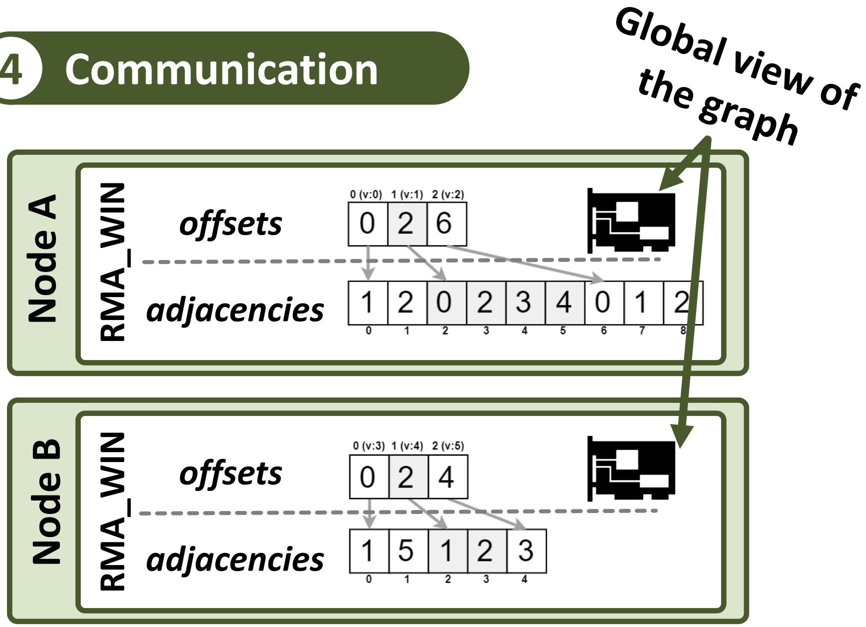
3 Asynchronous distributed memory algorithm

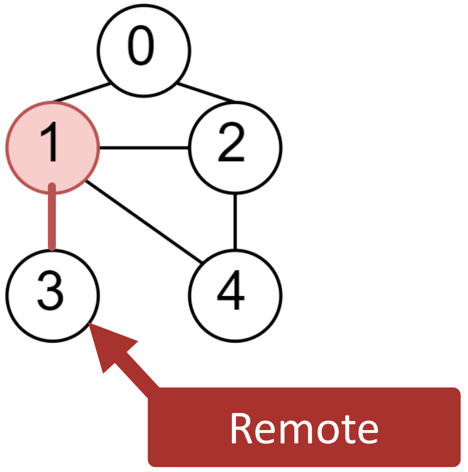
1. **For all local vertices *v*:**
   2. **For all vertices w incident to v:**
      If w is remote: Get $adj(w)$

   $\#\text{triangles} \mathrel{+}= |\, adj(v) \cap adj(w)|$



4 Communication

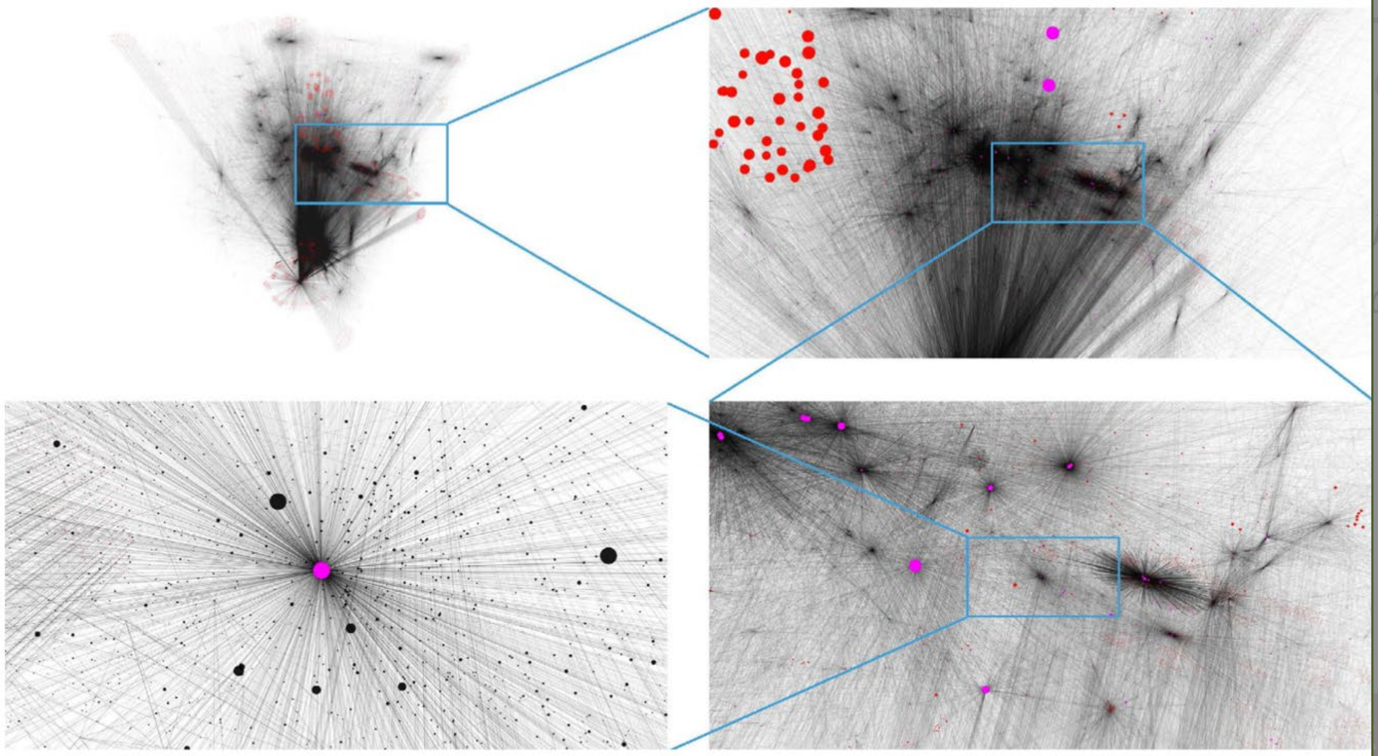




SPCL spcl.ethz.ch @spcl_eth DINFK

Challenges: Graphs are huge and skewed

- Billions of vertices and hundreds of billions of edges
- Scale-free degree distribution



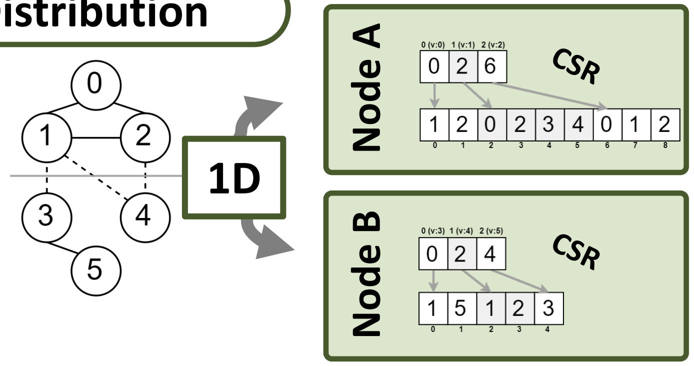
Right: Albert-László Barabási: Network Science (Chapter 4)

5

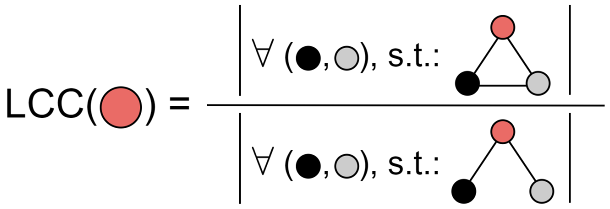
Exploit temporal locality by caching RMA reads

Algorithm overview: Caching with CLaMPI

1 Distribution



2 Local TC computation

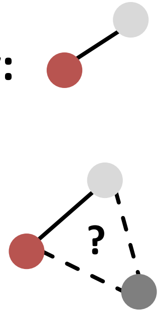


- Shared memory parallel
- Hybrid method

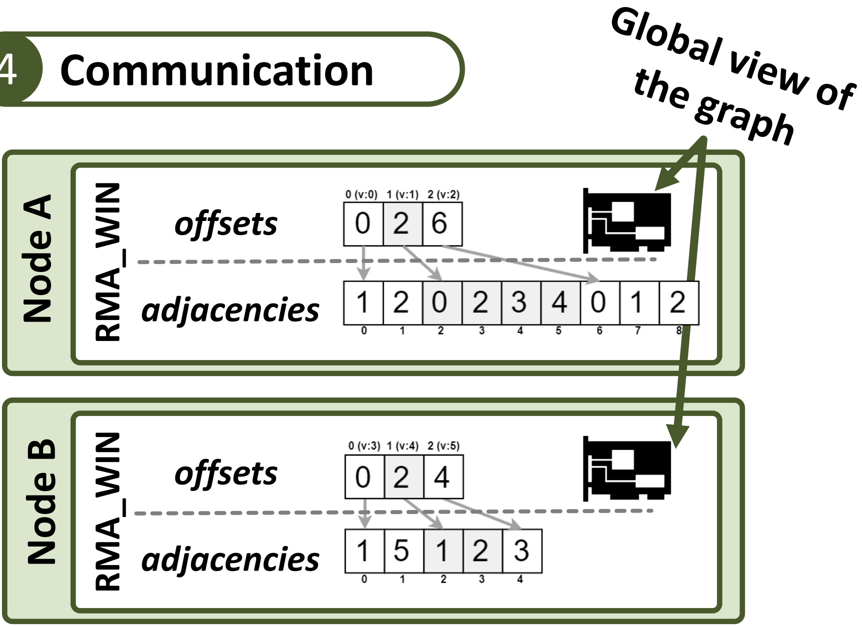
3 Asynchronous distributed memory algorithm

1. For all local vertices *v*:
   2. For all vertices w incident to v:
      If w is remote: Get $adj(w)$

   #triangles += $|\, adj(v) \cap adj(w) |$



4 Communication



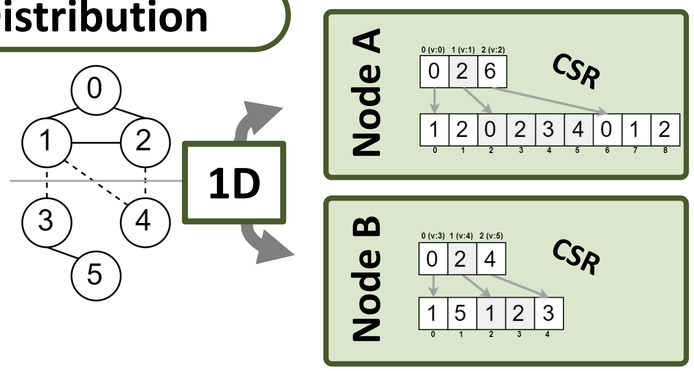
5 Data reuse

CLaMPI – RMA cache

- Transparent caching layer
- Supports variable size reads

Algorithm overview: Caching with CLaMPI

1 Distribution



2 Local TC computation

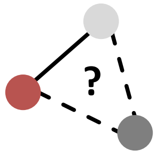
LCC(●) = |∀ (●,●), s.t.: [triangle]| / |∀ (●,●), s.t.: [wedge]|

- **Shared memory parallel**
- **Hybrid method**

3 Asynchronous distributed memory algorithm

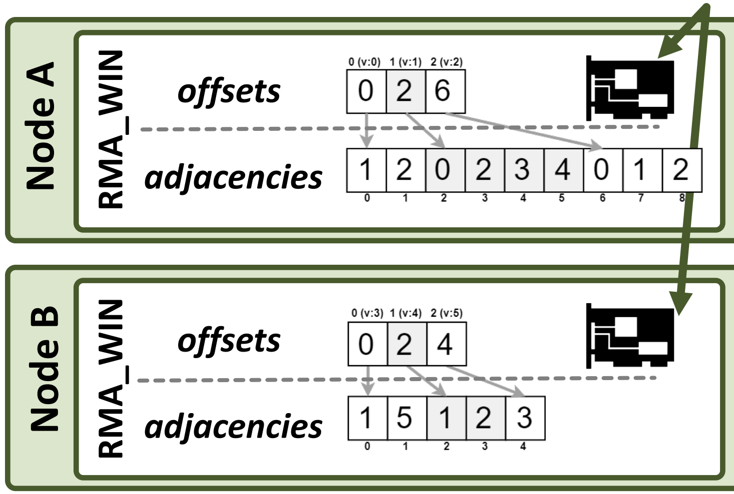
1. **For all local vertices *v*:**
   2. **For all vertices w incident to v:**
      If w is remote: Get $adj(w)$

#triangles += $|\, adj(v) \cap adj(w) \,|$



4 Communication

Global view of the graph



5 Data reuse

Frequently accessed subgraph (redundant)

Node A — **CLaMPI cache**

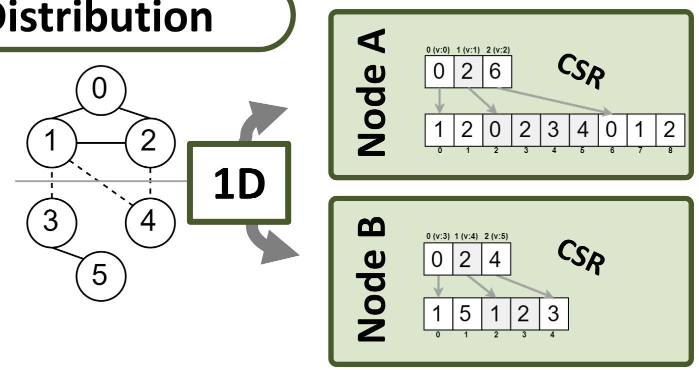
window	node	offset	size	data
offsets	B	0	2	0 2
adjacencies	B	0	2	1 5
...	...	...	...	

Node B — **CLaMPI cache**

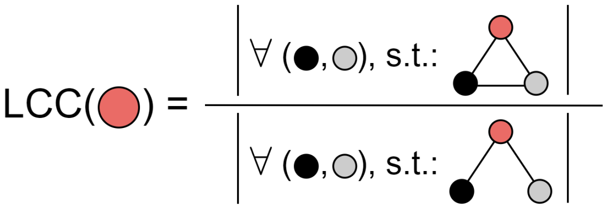
window	node	offset	size	data
offsets	A	0	2	2 6
adjacencies	A	0	2	0 2 3 4
...	...	...	...	

Algorithm overview: Caching with CLaMPI

1 Distribution



2 Local TC computation



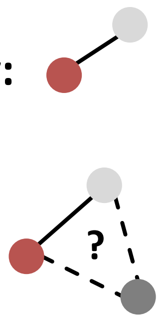
- **Shared memory parallel**
- **Hybrid method**

3 Asynchronous distributed memory algorithm

1. **For all local vertices *v*:**
   2. **For all vertices w incident to v:**

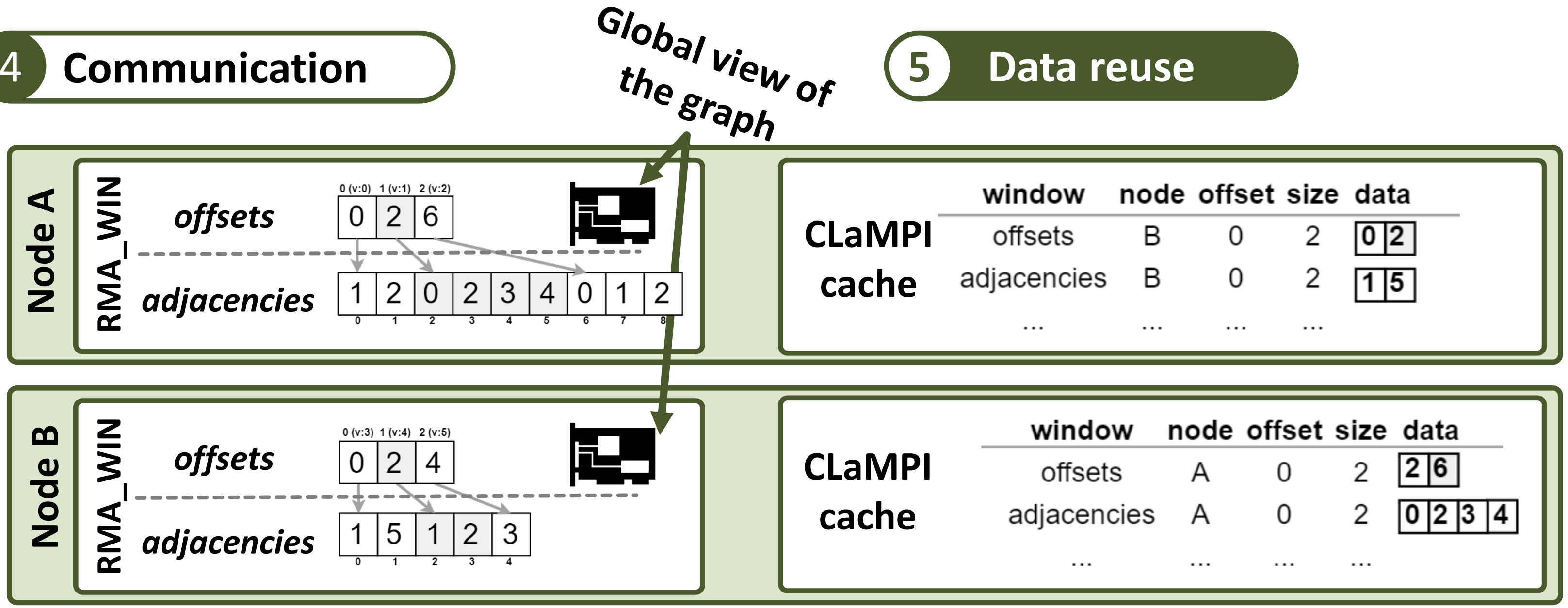
      If w is remote: Get $adj(w)$

   $\#\text{triangles} \mathrel{+}= |\, adj(v) \cap adj(w) |$



4 Communication

5 Data reuse



Node A — CLaMPI cache

window	node	offset	size	data
offsets	B	0	2	0 2
adjacencies	B	0	2	1 5
...	...	...	...	

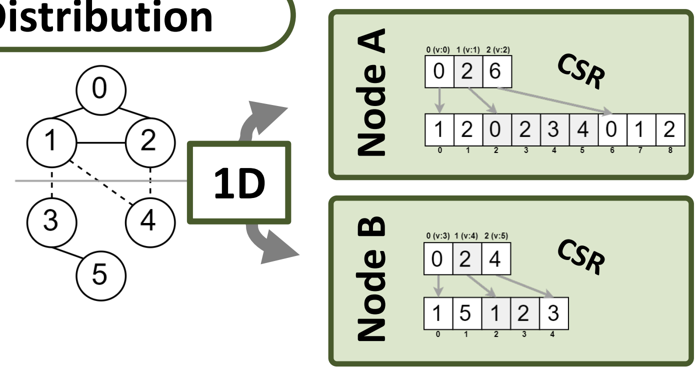
Node B — CLaMPI cache

window	node	offset	size	data
offsets	A	0	2	2 6
adjacencies	A	0	2	0 2 3 4
...	...	...	...	

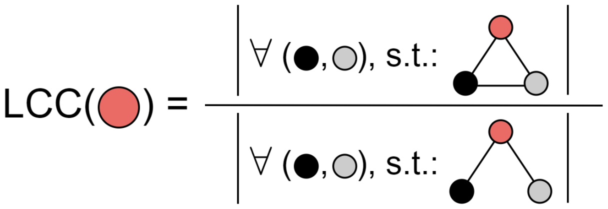
- **User defined score**
  - **Improvement between 14.4% and 35.6%**

Algorithm overview

1 Distribution



2 Local TC computation

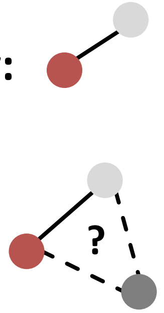


- **Shared memory parallel**
- **Hybrid method**

3 Asynchronous distributed memory algorithm

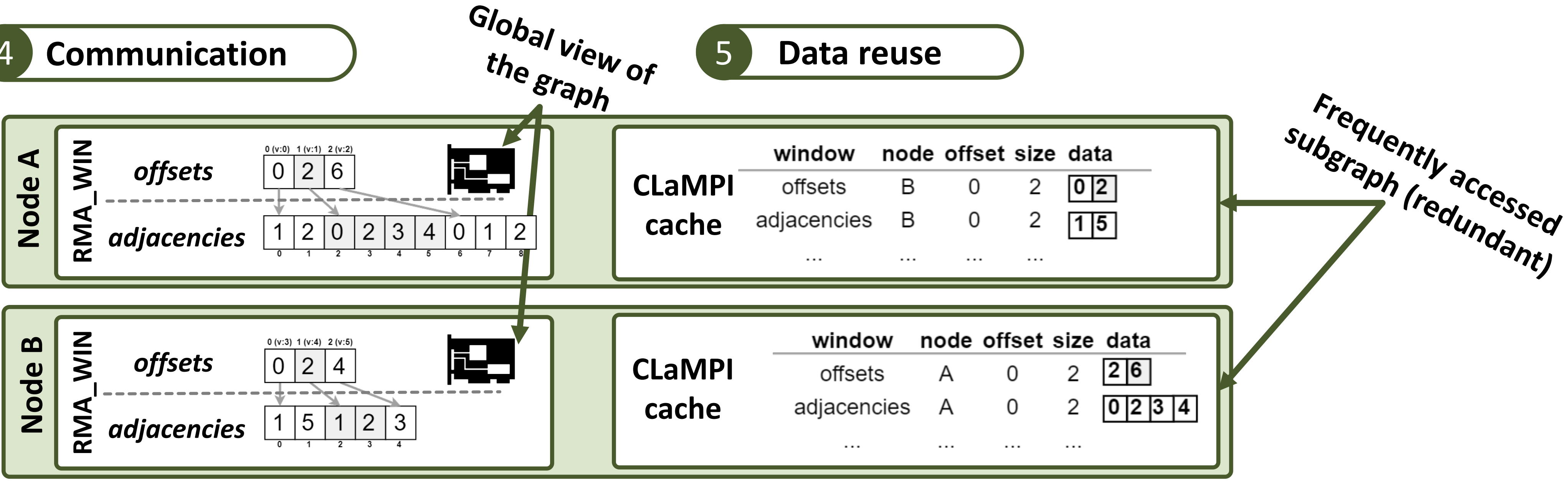
1. **For all local vertices *v*:**
   2. **For all vertices w incident to v:**
      If w is remote: Get $adj(w)$

   $\#\text{triangles} \mathrel{+}= |\, adj(v) \cap adj(w)|$



4 Communication

5 Data reuse



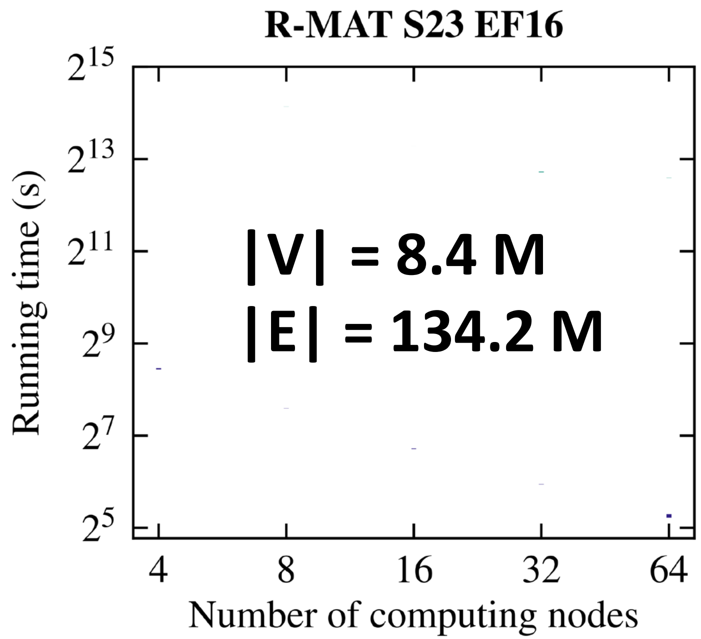
CLaMPI cache (Node A)

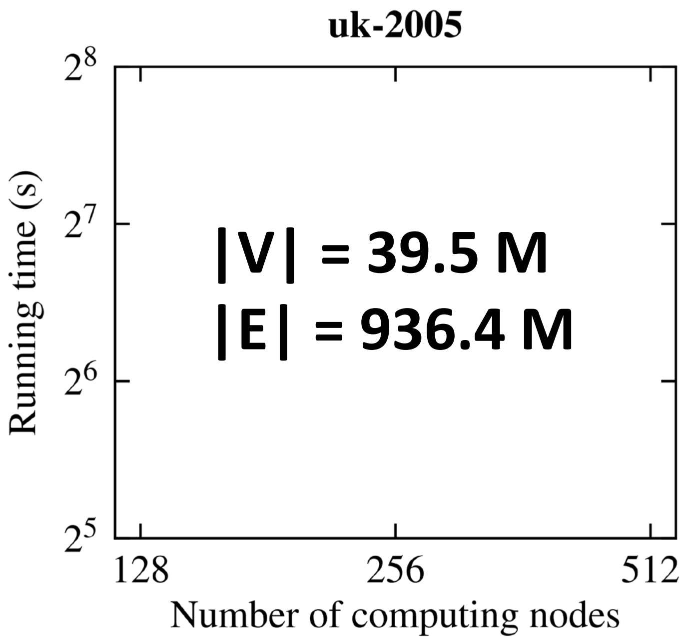
window	node	offset	size	data
offsets	B	0	2	0 2
adjacencies	B	0	2	1 5
…	…	…	…	

CLaMPI cache (Node B)

window	node	offset	size	data
offsets	A	0	2	2 6
adjacencies	A	0	2	0 2 3 4
…	…	…	…	

Results





System specification

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

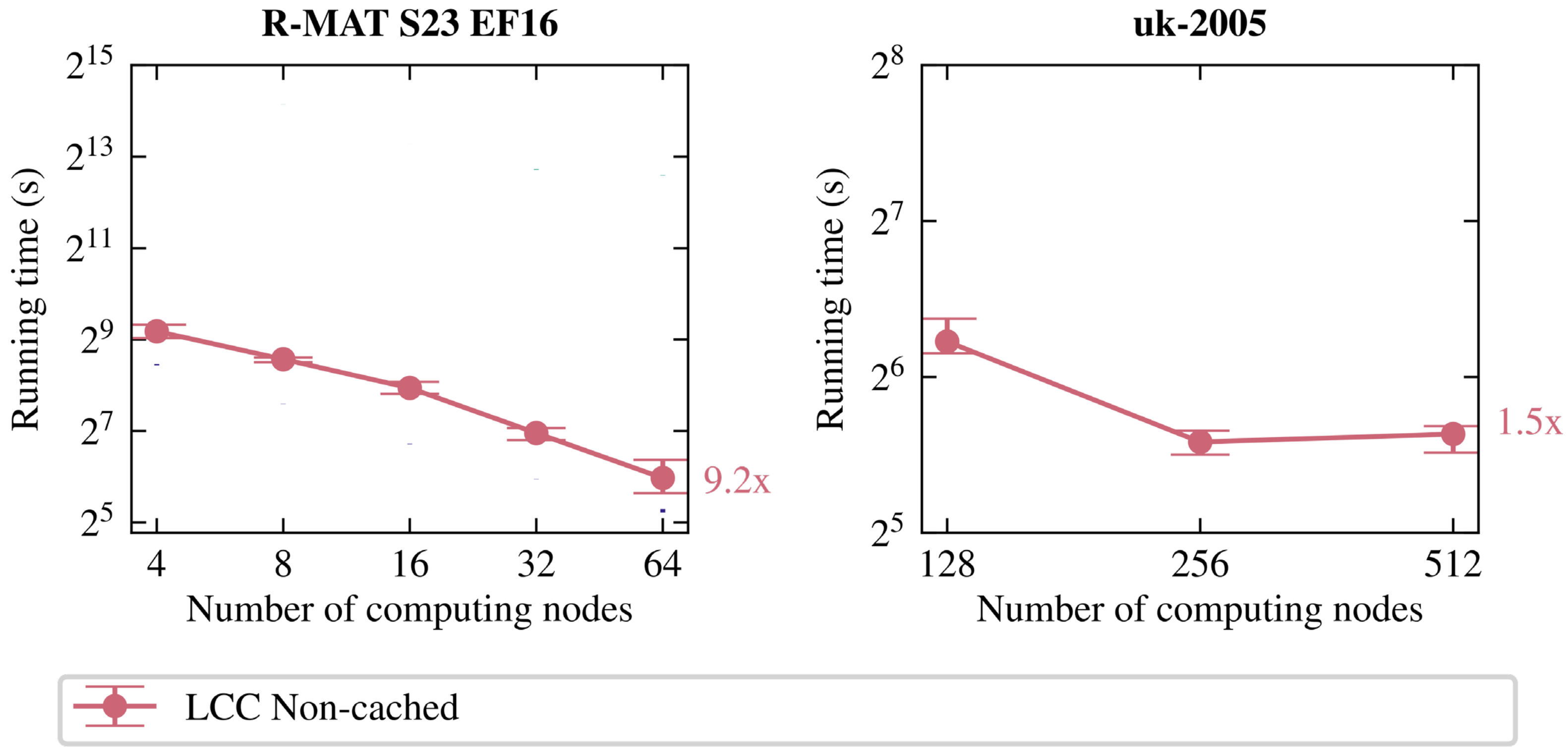
Intel® Xeon® E5-2690 v3
2.60GHz (12 cores)

64 GBs
per node

Cray's Aries interconnect
(dragonfly topology)

ICC 19.1 with -O3
cray-mpich 7.7.16 MPI

Results



System specification

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

Intel® Xeon® E5-2690 v3
2.60GHz (12 cores)

64 GBs
per node

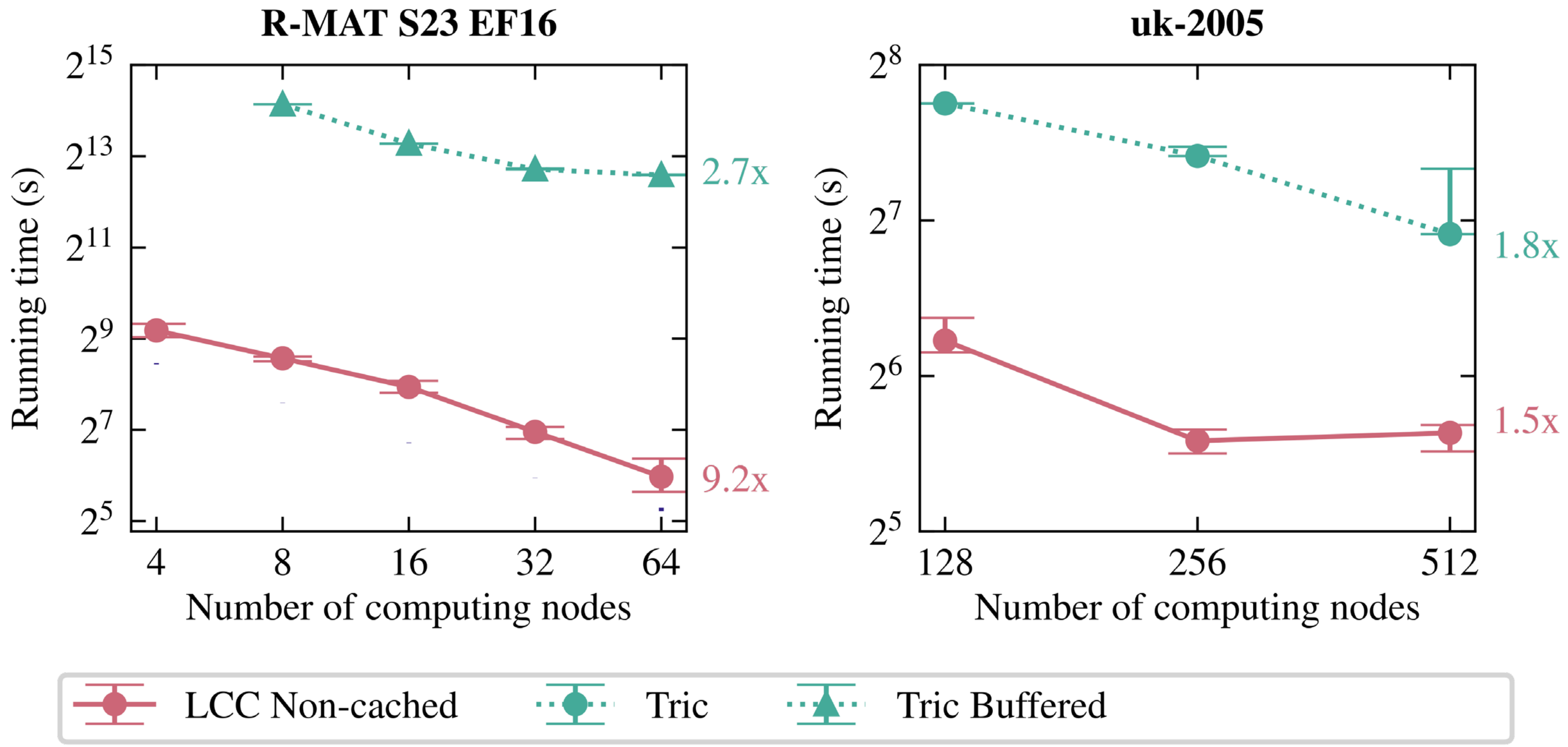
Cray's Aries interconnect
(dragonfly topology)

ICC 19.1 with -O3
cray-mpich 7.7.16 MPI

Results

Better scaling, especially for highly skewed graphs

In general, **4-12x faster** results, best results show **up to 100x speedup**



Eliminating synchronization overheads

System specification

CSCS Centro Svizzero di Calcolo Scientifico Swiss National Supercomputing Centre

Intel® Xeon® E5-2690 v3 2.60GHz (12 cores)

64 GBs per node

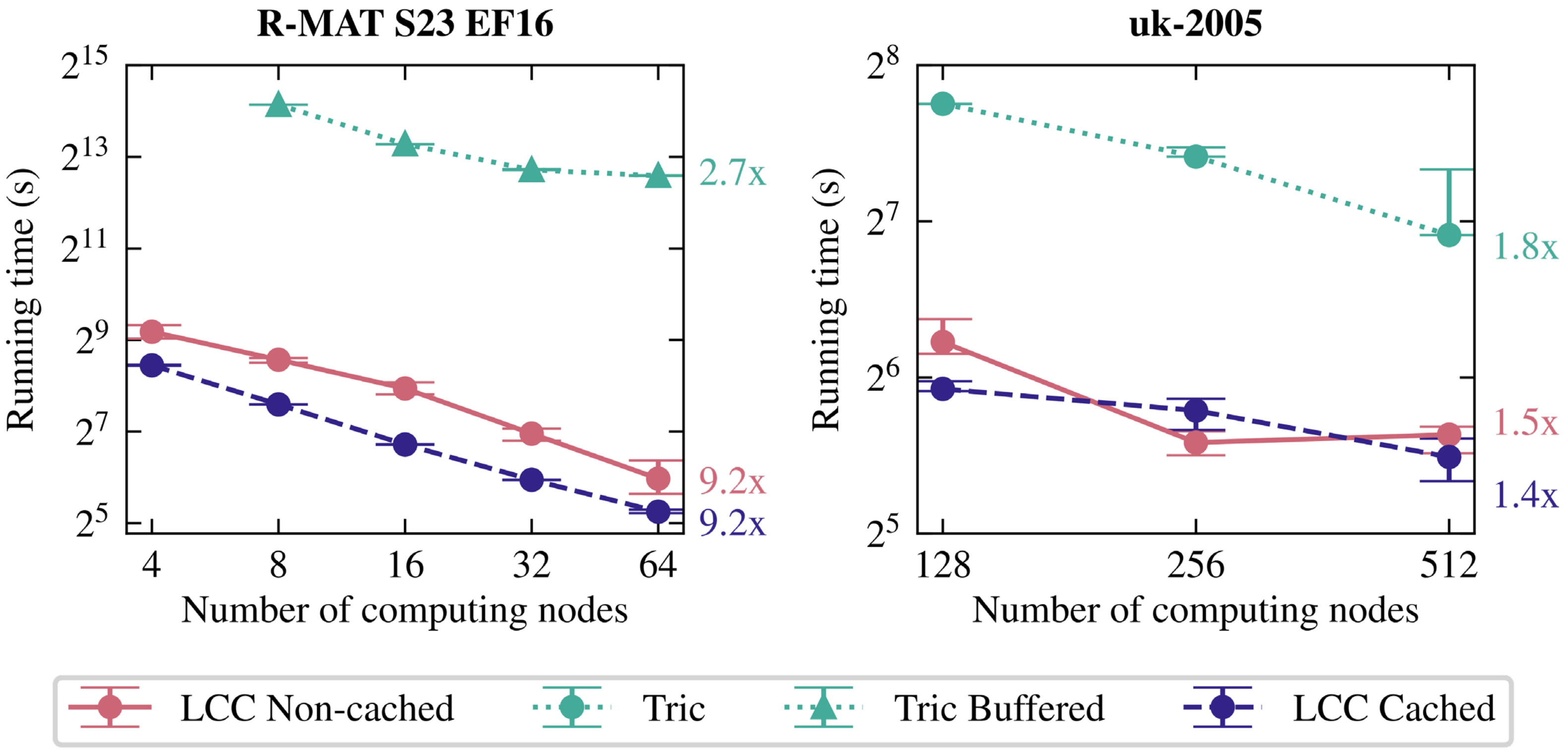
Cray's Aries interconnect (dragonfly topology)

ICC 19.1 with -O3 cray-mpich 7.7.16 MPI

Results

Better scaling, especially for highly skewed graphs

In general, **4-12x faster** results, best results show **up to 100x speedup**



Caching reduces runtime with up to **73%**

✓ **Eliminating synchronization overheads**

✓ **Vertex delegation by caching**

System specification

CSCS Centro Svizzero di Calcolo Scientifico Swiss National Supercomputing Centre

Intel® Xeon® E5-2690 v3 2.60GHz (12 cores)

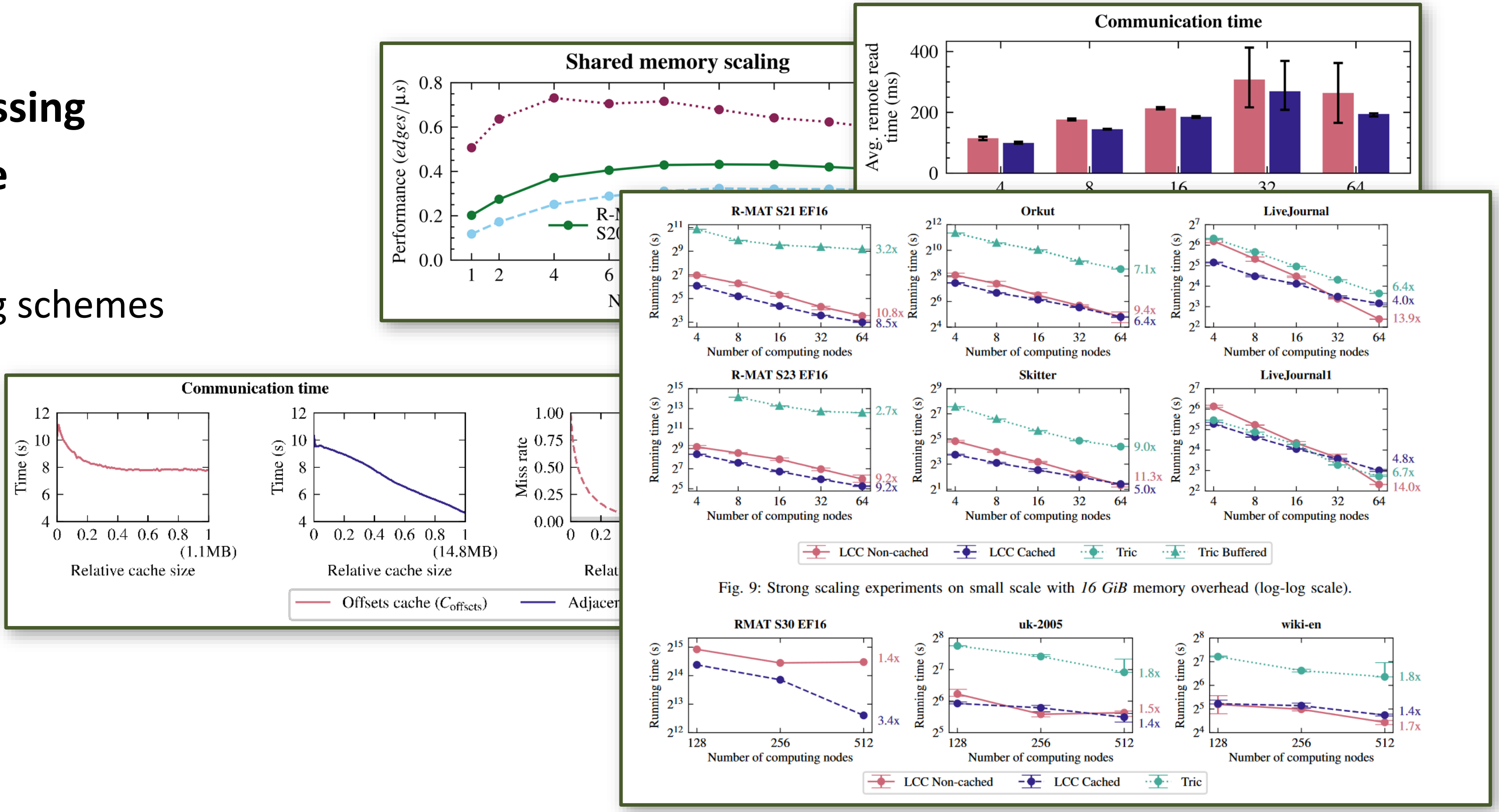
64 GBs per node

Cray's Aries interconnect (dragonfly topology)

ICC 19.1 with -O3 cray-mpich 7.7.16 MPI

Results

- **Asynchronous processing**
- **Caching performance**
- **Outlook**
  - Different partitioning schemes
  - Caching in other graph computations
  - Application specific eviction procedure



Further results and analysis in the paper!

System specification

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

Intel® Xeon® E5-2690 v3
2.60GHz (12 cores)

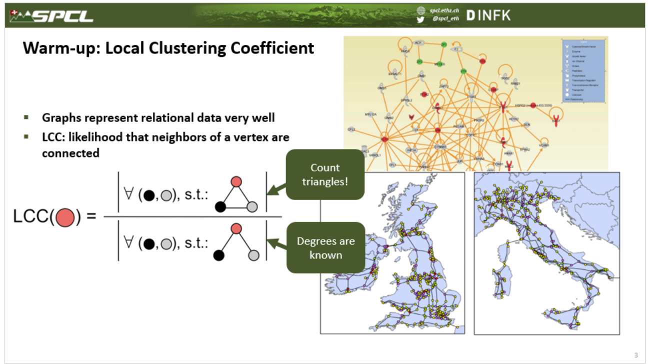
64 GBs
per node

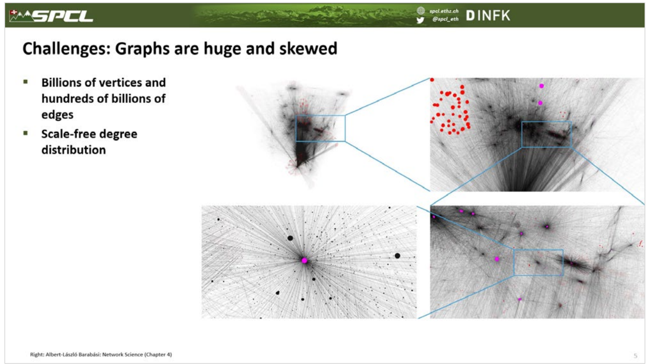
Cray's Aries interconnect
(dragonfly topology)

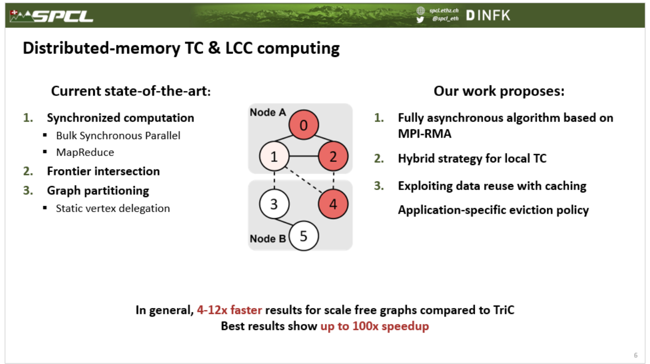
ICC 19.1 with -O3
cray-mpich 7.7.16 MPI

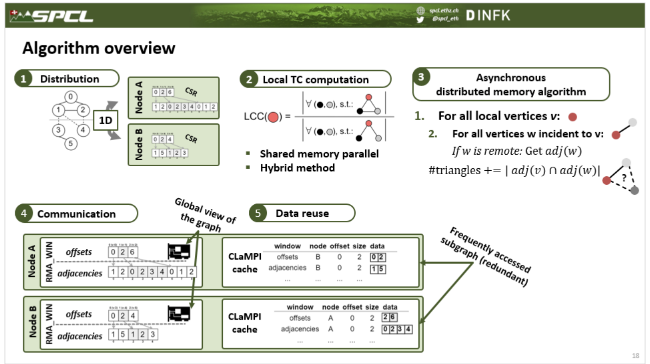
Asynchronous Distributed-Memory Triangle Counting and LCC with RMA Caching

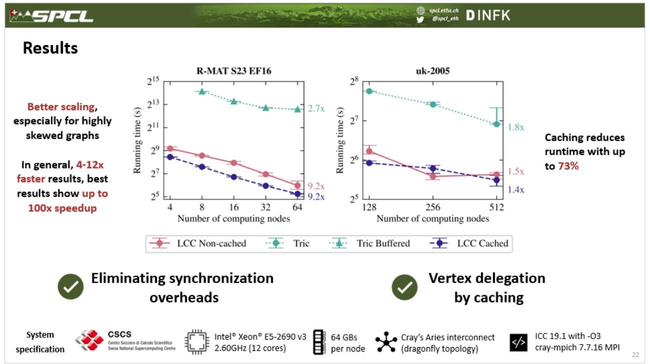
András Strausz*, Flavio Vella†, Salvatore Di Girolamo‡, Maciej Besta‡ and Torsten Hoefler‡
*‡Dept. of Computer Science, ETH Zürich, Zürich, Switzerland
†Dept. of Engineering and Computer Science, University of Trento, Trento, Italy
*strausza@student.ethz.ch, †flavio.vella@unitn.it, ‡firstname.lastname@inf.ethz.ch











Thank you for your attention!