Queen's University Belfast | GLOBAL RESEARCH INSTITUTES | asa | ECIT THE INSTITUTE OF ELECTRONICS COMMUNICATIONS AND INFORMATION TECHNOLOGY

# Taming Irregular Memory Accesses in Graph Analytics Workloads


Hans Vandierendonck h.vandierendonck@qub.ac.uk

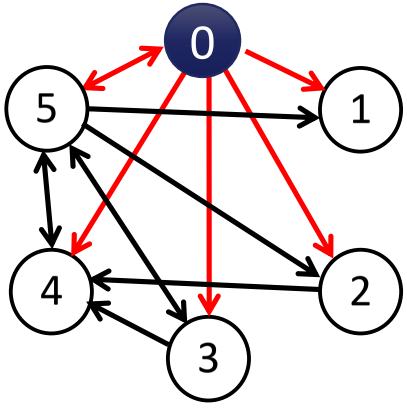
Joint work with:

Jiawen Sun jsun03@qub.ac.uk

Dimitrios S. Nikolopoulos d.nikolopoulos@qub.ac.uk

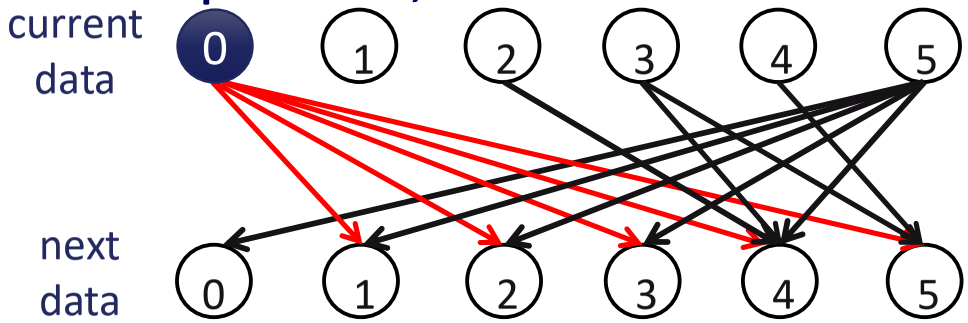
# Graph Algorithms

- Iteratively calculate a property of each vertex in a graph
- E.g.: PageRank values, label of connected components , etc.




- Vertex model


- Driven by frontier: set of active vertices
- Many updates in parallel, with conflicts

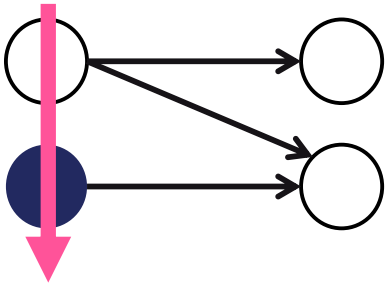


# Challenges
# Shared-Memory Graph Analytics

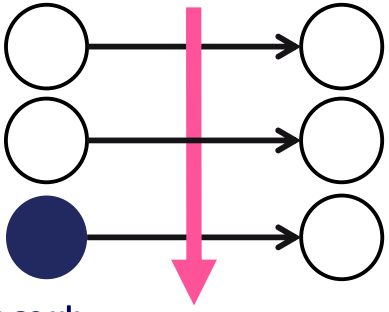
- Memory intensive
  - Irregular memory accesses
  - Poor memory locality
  - Non-Uniform Memory Access (NUMA)
- Little work per edge
  - ~10-20 assembly instructions per edge
  - Loop control causes major overhead
- Key: graph partitioning
  - Fast, creating parallelism and locality
  - Unaware of graph topology
- Degrees of freedom:
  - Iteration order over edges
- Restrictions:
  - Parallelism
  - Data races

# State-of-the-Art Graph Data Structures

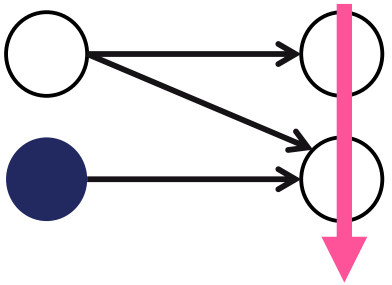
- Compressed Sparse Rows (CSR)
  - List outgoing edges for each vertex
  - "Forward" traversal (push)



- Coordinate list (COO)
  - A list of edges



- Compressed Sparse Columns (CSC)
  - List incoming edges for each vertex
  - "Backward" traversal (pull)

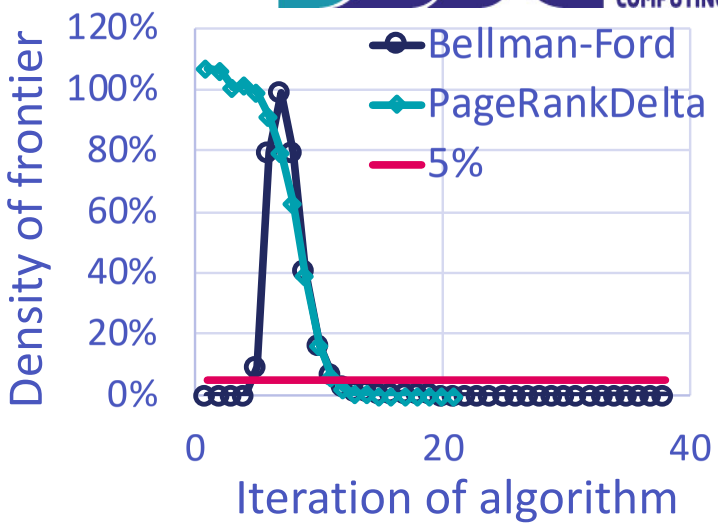


CSC and COO require that all edges are traversed

# State-of-the-Art Graph Traversal

**Graph traversal [SC'12] [SPAA'13]**

```
d = (#active vertices +
#active edges) / #edges

if d > 5% then
    # dense frontier
    if algorithm prefers
       forward then
       traverse CSR
    else
       traverse CSC
    endif
else # d <= 5%
    # sparse frontier
    traverse CSR
endif
```




Programmer's choice
Little is known to guide this

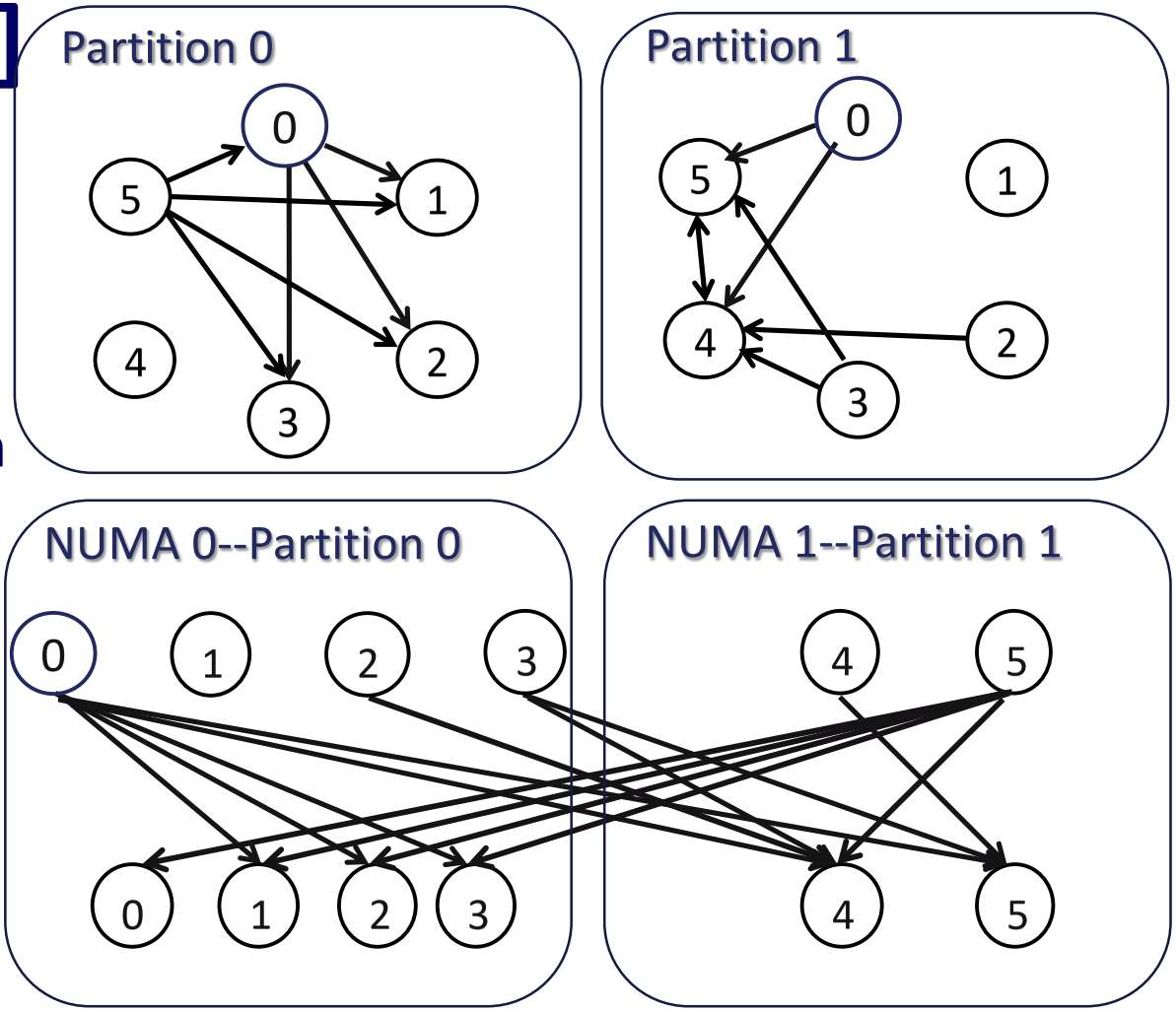
We will shed some light on this
… work in progress

Requires storage of both
CSC and CSR

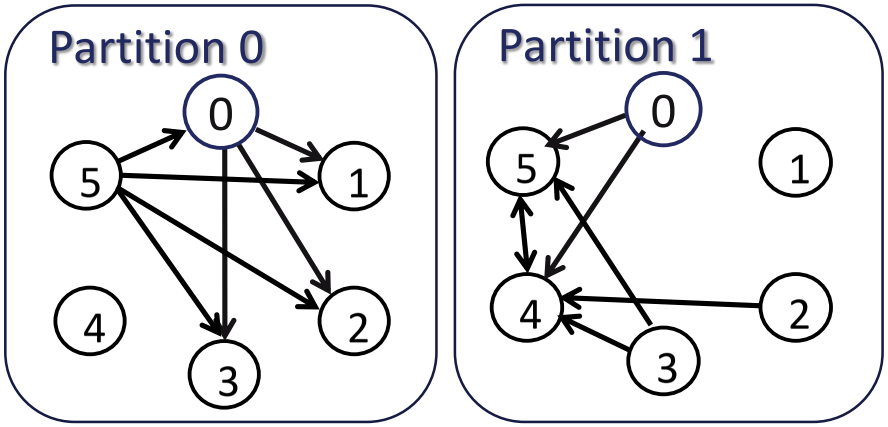
# Partitioning Real-World Graphs

# Graph Partitioning [PPoPP 2015]

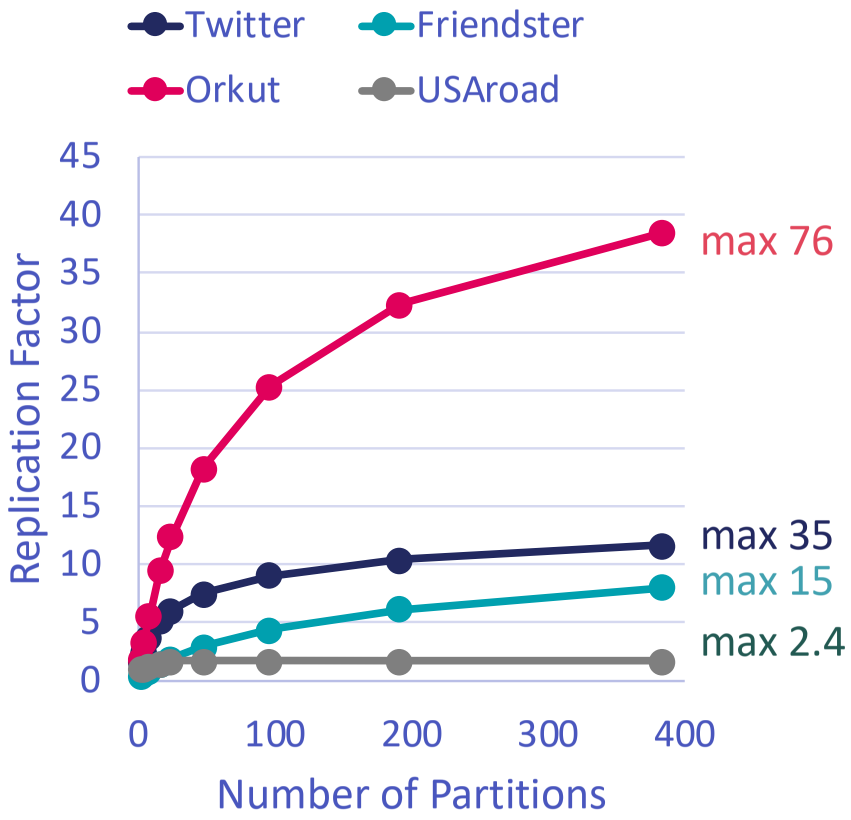
- Simple rules:
- Place edges with same destination in same partition
- Place vertices with consecutive labels in same partition
- Balance number of edges in each partition
- Causes imbalance in vertices



# Vertex Replication

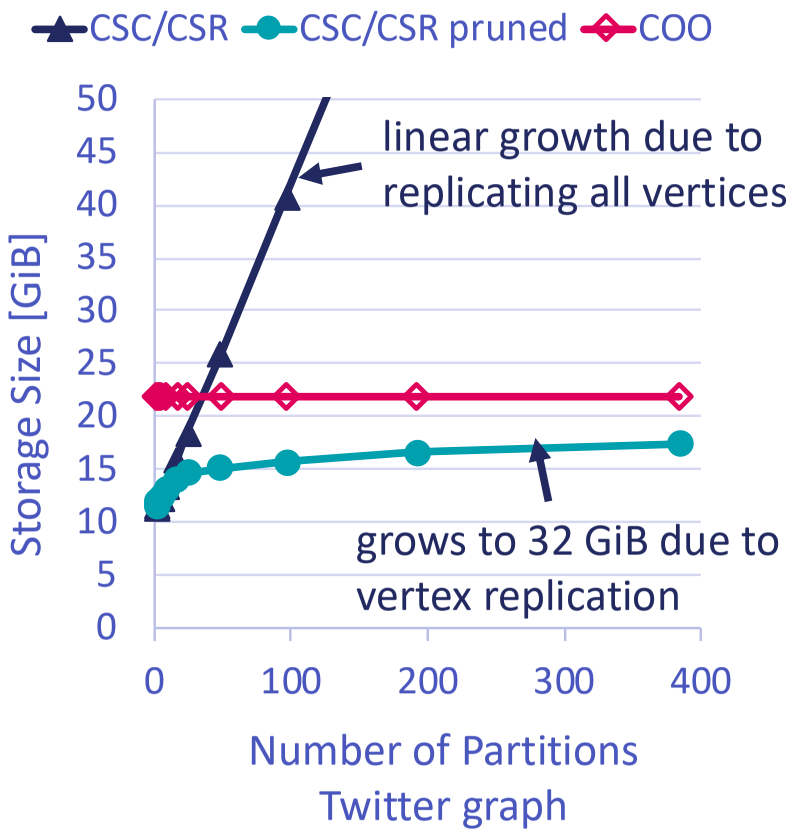



- When partitioning the edge set, a vertex may appear in multiple partitions
- Relevant terms: *vertex cut, shadow vertices, ghost vertices*
- Replication factor = #repeated vertices / #unique vertices




Max replication factor is #edges / #vertices

# Implications of Vertex Replication

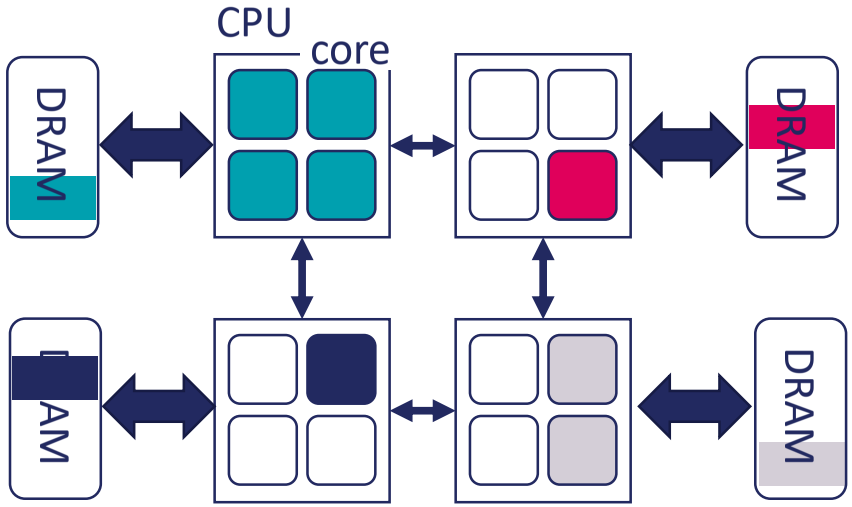



- CSC and CSR are not scalable formats
  - Increased storage
  - Increased execution time to "visit" vertices
- Pruned CSC/CSR
  - Omits zero-degree vertices
- COO is scalable to any number of partitions
  - But inefficient for sparse frontiers

# Optimising NUMA Layout

# NUMA-Aware Scheduling

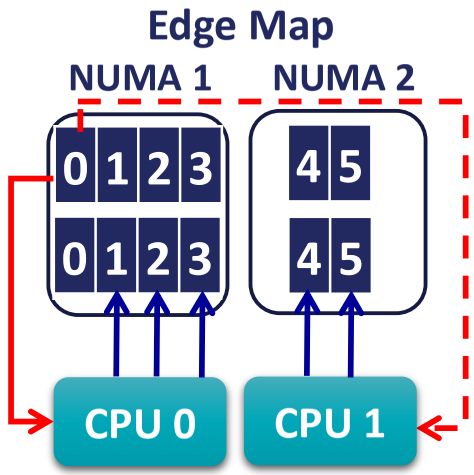
- Two-pronged approach:
  - Allocating data on specific NUMA nodes
  - Mapping code to CPU cores located “close” to data
- Data allocation:
  - malloc/new
  - mmap + mbind




- New annotation for code scheduling:
  #pragma cilk numa(strict)
  cilk_for( n=0; n < NUMA; ++n ) { … }
  - Loop range restricted to number of NUMA nodes
  - Composes with other Cilk constructs

# NUMA-Awareness in GraphGrind: Edge Traversal

- Partition CSC and CSR
  - 1 partition per NUMA node
  - All stores are NUMA-local

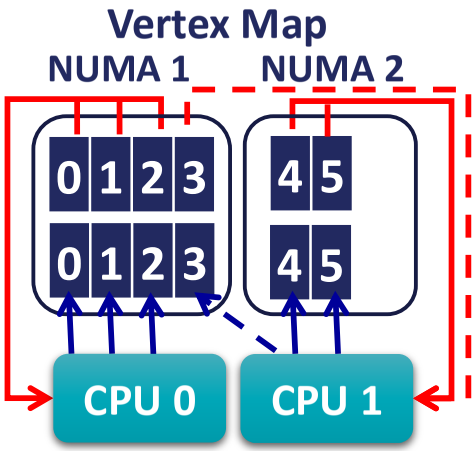



- Exception: sparse frontiers
  - Very few active vertices
  - Vertex replication incurs major overhead
  - Use unpartitioned CSR
  - NUMA: interleaved allocation

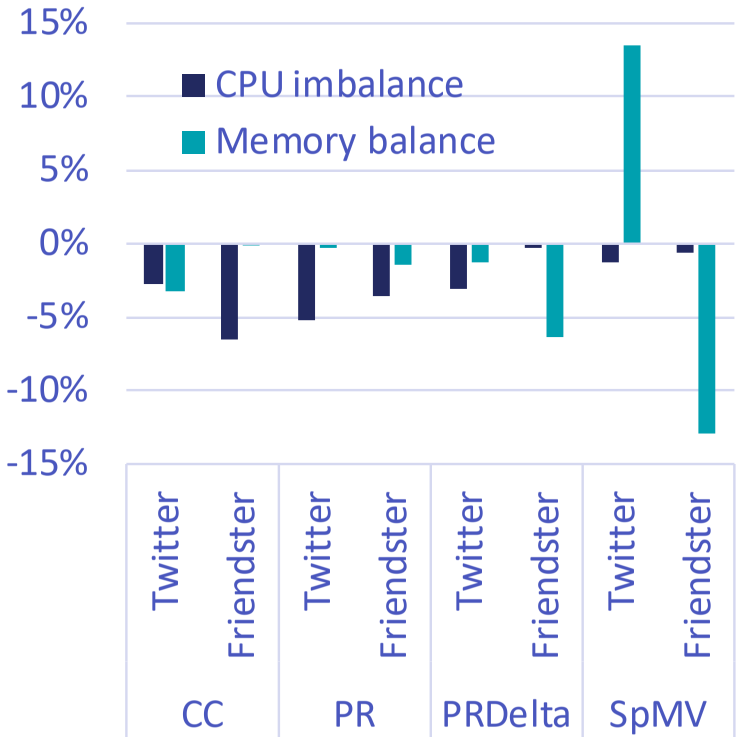
Local Read  Remote Read  Local Write  Remote Write

# NUMA-Awareness in GraphGrind: Vertex Traversal

- Edge traversal dictates layout of vertex data
  - Prefer CPU load balancing
  - Incur remote loads and stores

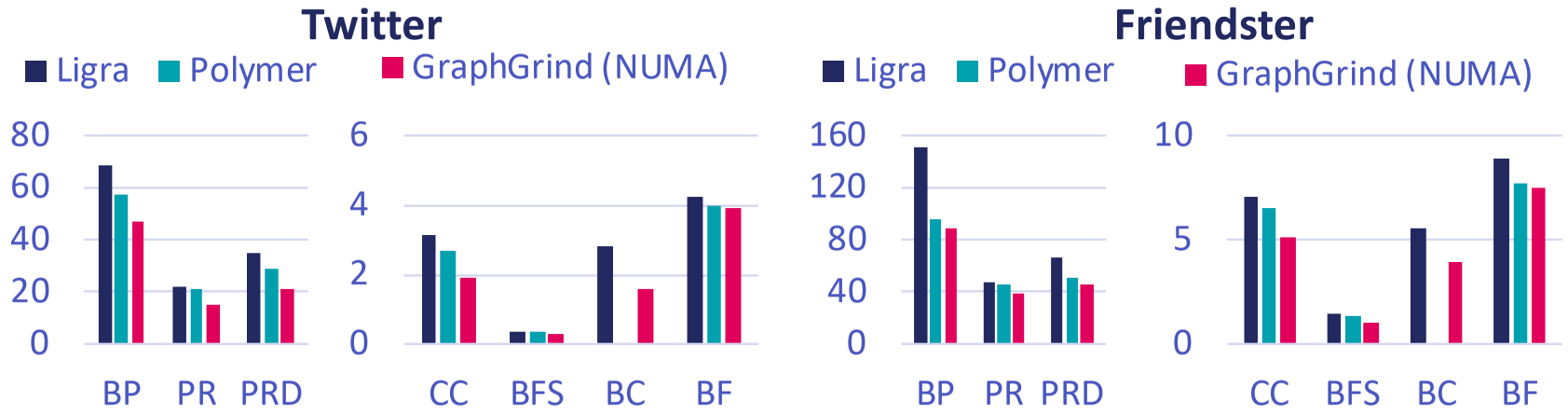



- Analysis of Alternatives




Local Read  Remote Read  Local Write  Remote Write

# Performance Evaluation NUMA Placement

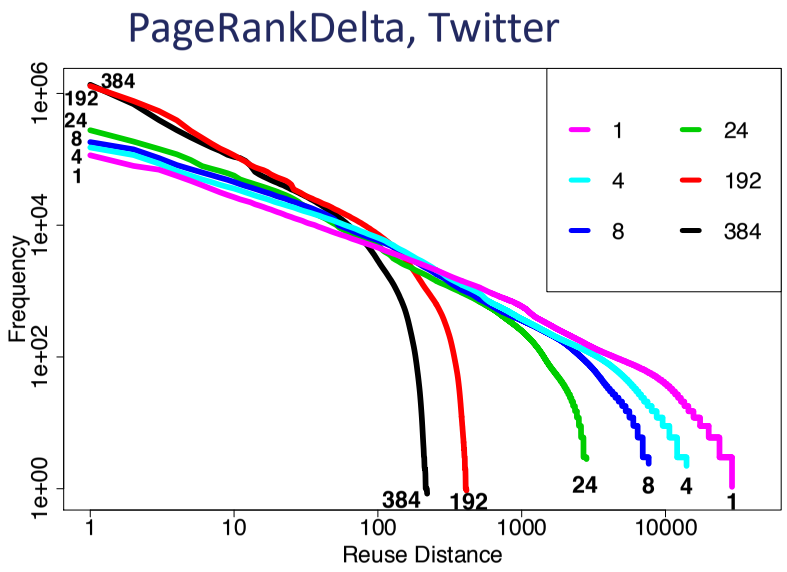



- Combination of optimisations
  - Pruned CSC/CSR representation
  - Tune partitioning to edge/vertex algorithms
  - NUMA-aware layout of vertex arrays
  - CSC traversal: “caching” intermediate values to minimise load/stores
  - Full frontier: specialised version of code that omits frontier check
  - Sparse CSR traversal: no partitioning applied

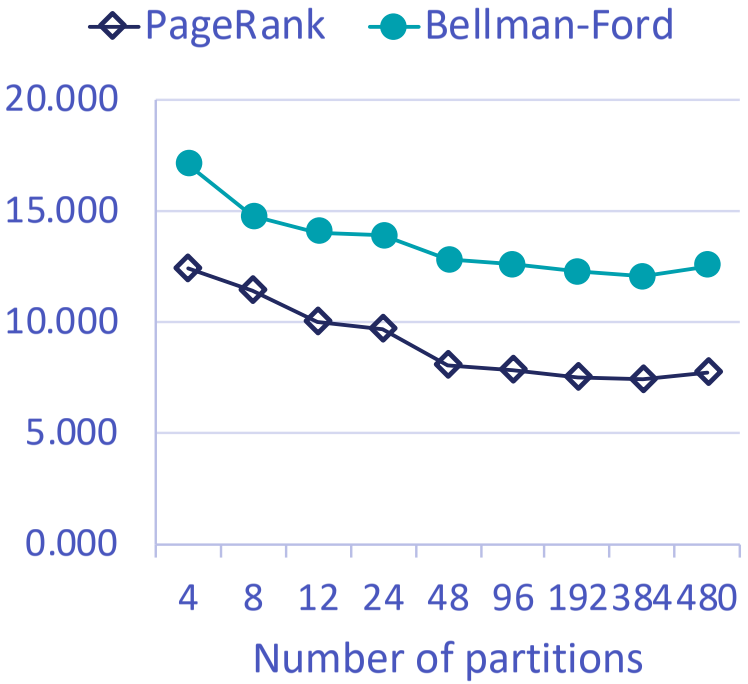
# Optimising Memory Locality

# Memory Locality

## Reuse Distance Distribution



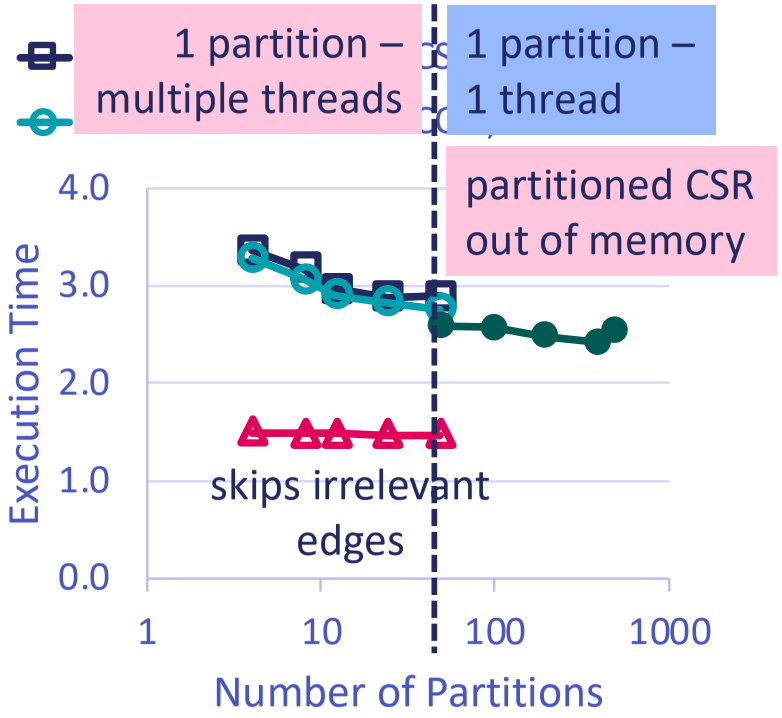

## Misses Per Kilo-Instruction (MPKI) – Twitter graph



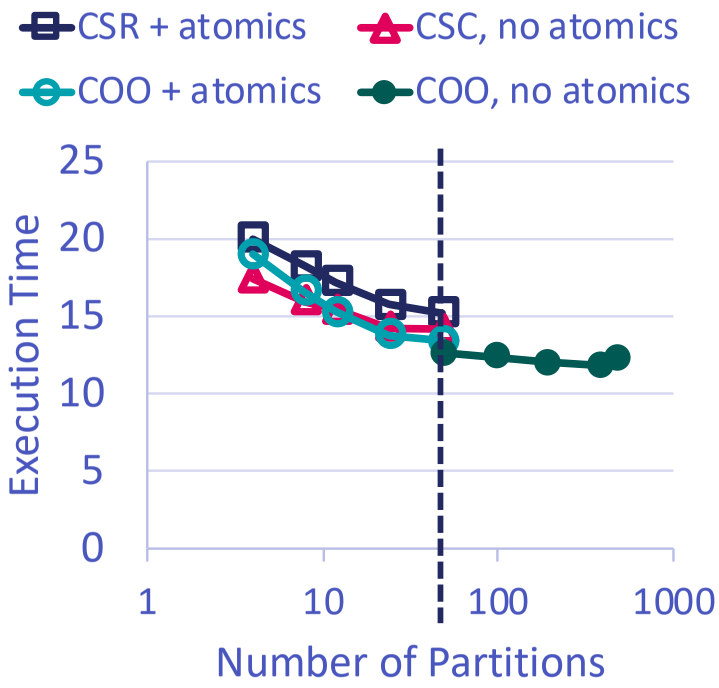
# Performance Benefits of Graph Partitioning

CENTRE FOR DATA SCIENCE AND SCALABLE COMPUTING

## Betweenness-Centrality, Twitter




## PageRank, Twitter 10 iterations



# Graph Traversal Revisited

CENTRE FOR DATA SCIENCE AND SCALABLE COMPUTING

**Graph traversal [SC'12] [SPAA'13]**

```
d = (#active vertices +
#active edges) / #edges

if d > 5% then
   # dense frontier
   if algorithm prefers
      forward then
      traverse CSR
   else
      traverse CSC
   endif
else # d <= 5%
   # sparse frontier
   traverse CSR
endif
```

**GraphGrind: locality optimised**

```
d = (#active vertices +
#active edges) / #edges

if d > 50% then
   # dense frontier
   traverse partitioned COO
else if d > 5% then
   # medium-dense case
   # dense frontier
   traverse CSC
else # d <= 5%
   # sparse frontier
   traverse CSR
endif
```

# Gra
# Rev

Preferred direction aligns to most common frontier density

CENTRE FOR DATA SCIENCE AND SCALABLE COMPUTING

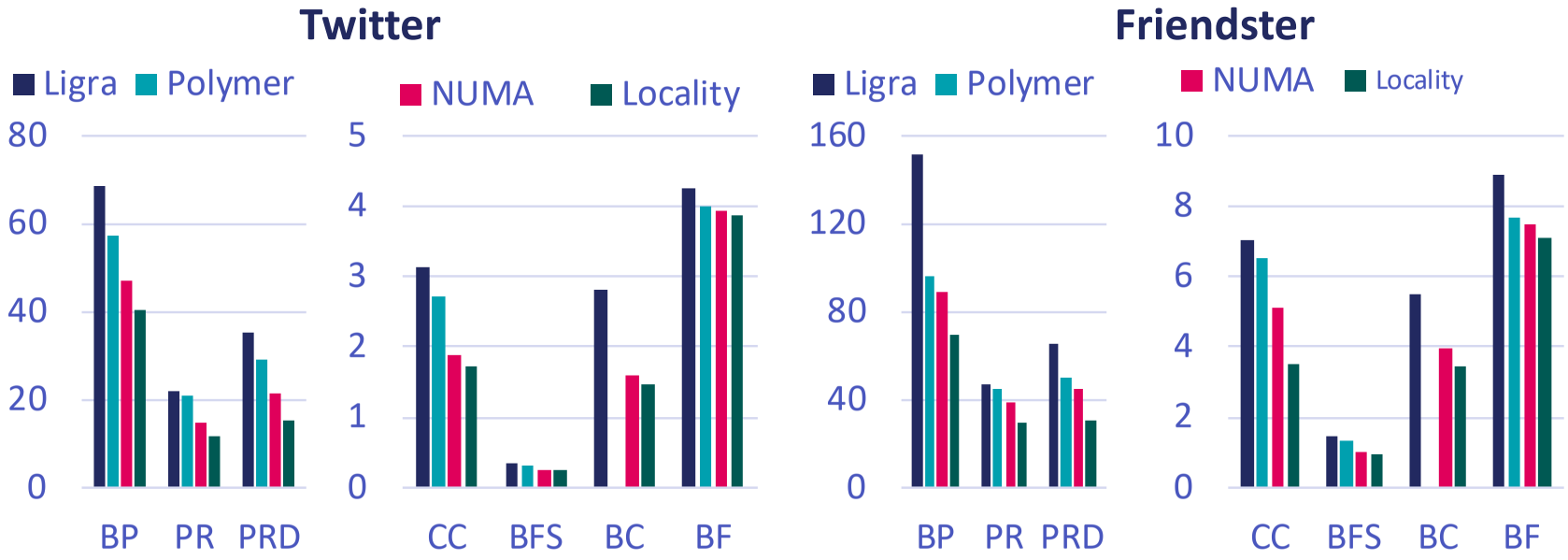
**Graph traversal [SC'12] [SPAA'13]**

```
d = (#active vertices +
#active edges) / #edges

if d > 5% then
   # dense frontier
   if algorithm prefers
      forward then
      traverse CSR
   else
      traverse CSC
   endif
else # d <= 5%
   # sparse frontier
   traverse CSR
endif
```

**GraphGrind: locality optimised**

```
d = (#active vertices +
#active edges) / #edges

if d > 50% then
   # dense frontier
   traverse partitioned COO
else if d > 5% then
   # medium-dense case
   # dense frontier
   traverse CSC
else # d <= 5%
   # sparse frontier
   traverse CSR
endif
```

# Performance Evaluation
# Locality Optimisation (384 partitions)




- Partitioning improves memory locality
- Partitions are unit of parallelism to resolve data races

# Conclusion

- Graph partitioning drives NUMA-awareness; memory locality optimisation and conflict-free parallelisation
- Three phases in edge-map: sparse (CSR), medium-dense (CSC), dense (COO)
- Significant performance improvement over state-of-the-art for shared-memory graph analytic
- Details:
  - Sun *et al.* "GraphGrind: Addressing Load Imbalance of Graph Partitioning", ICS'17
  - Sun *et al.* "Accelerating Graph Analytics by Utilising the Memory Locality of Graph Partitioning", To appear in ICPP'17
- Ongoing Work:
  - Compact graph representations and efficient graph ingress … leveraging graph partitioning