KTrussExplorer: Exploring the Design Space of K-truss Decomposition Optimizations on GPUs

Safaa Diab, Mhd Ghaith Olabi, **Izzat El Hajj**

American University of Beirut

HPEC Graph Challenge

September 23, 2020

Overview

KTrussExplorer is a highly parameterized framework for exploring different combinations of k-truss decomposition optimizations on GPUs

Supported features:

- Edge-centric parallelization
- Undirected or directed graphs
  - Directed by index or by degree
- Tiling the adjacency matrix
- Parallelizing intersections
- Removing or marking weak edges
- Recomputing for all or affected edges

Contributions:

- A survey of optimizations
- A framework for exploring the design space

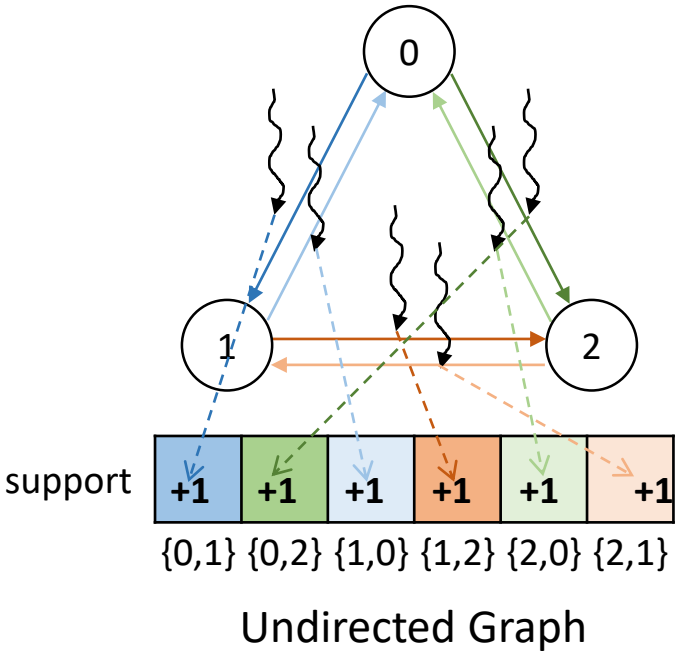
github.com/ielhajj/ktruss-explorer

- A view of the design space
- Unexplored combinations faster than prior champions

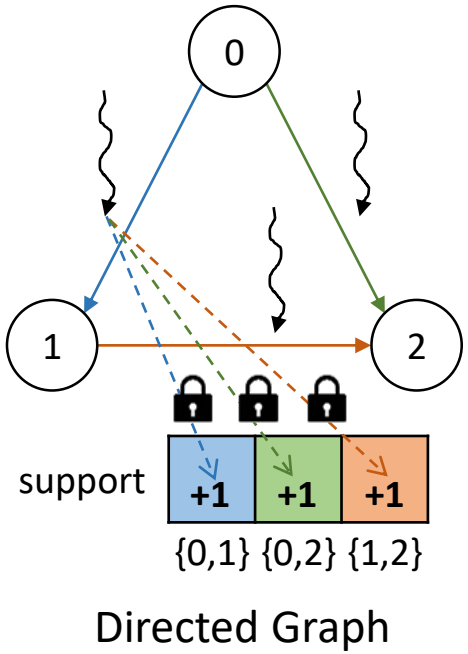
Methodology

- **Software:** KtrussExplorer kernels are implemented in CUDA
- **System:** Evaluation is on one Volta V100 GPU with 16GB of memory
- **Datasets:** We evaluate with all graphs in the graph challenge collection
  - Except: Friendster, graph500-scale24-ef16, and graph500-scale25-ef16 due to limited device memory capacity.
- **Search space:** Design space is searched exhaustively
  - Except: very large graphs

Graph Directedness

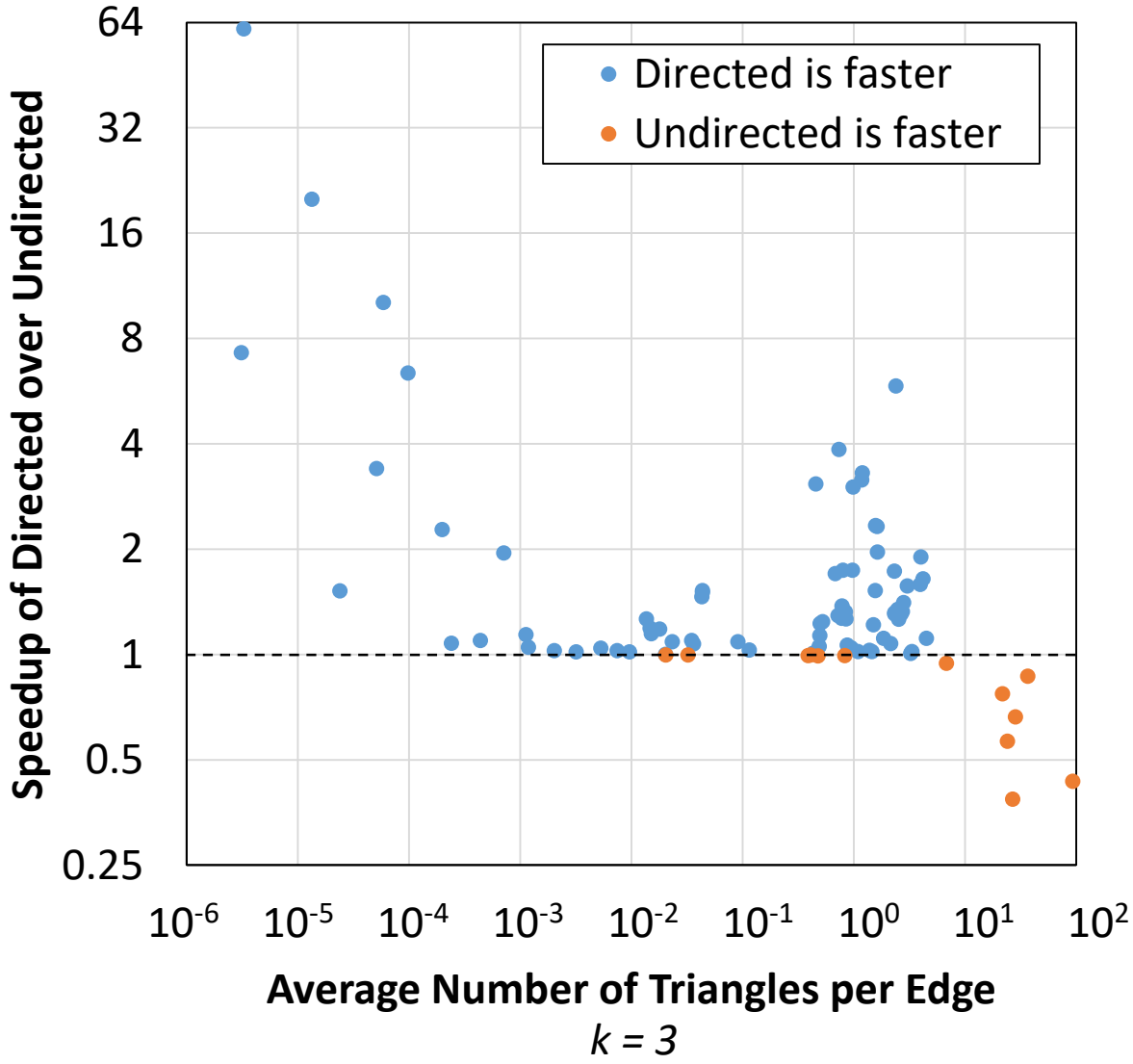


Undirected Graph



Directed Graph

- ✓ Less synchronization (no atomics)
- ✓ Stop counting early
- ✓ Less redundancy



Directing Edges by Degree

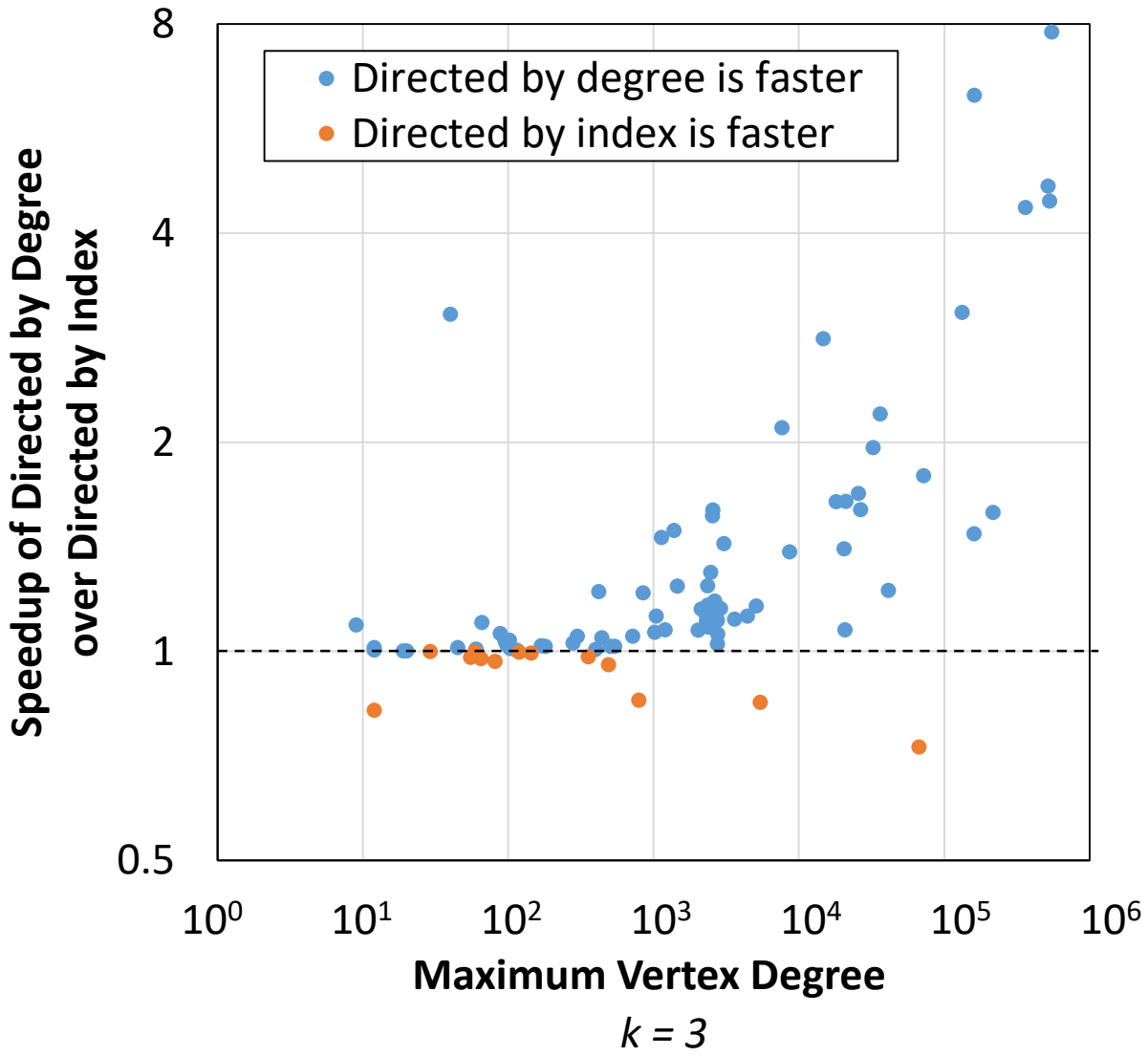
Directed by index

- Keep edges from vertex with lower index to vertex with higher index

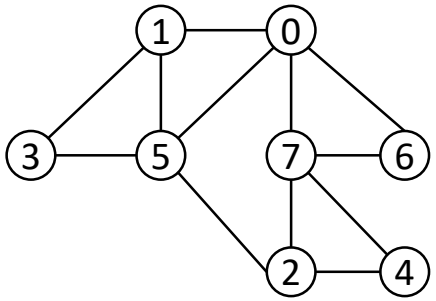
Directed by degree

- Keep edges from vertex with lower degree to vertex with higher degree

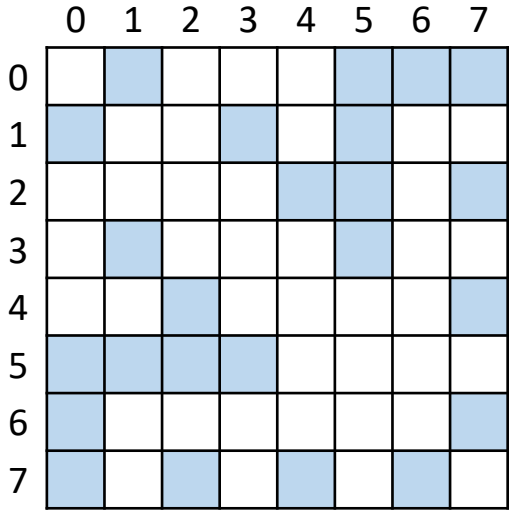
✓ Advantage: shrink large adjacency lists to reduce load imbalance



Tiling



Example Graph



Logical Adjacency List without Tiling

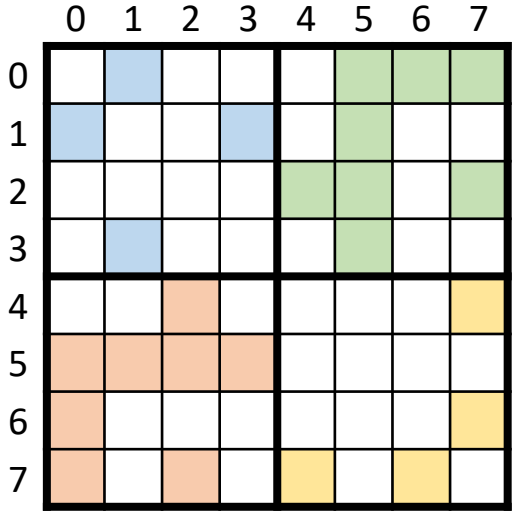
srcPtr

0	4	7	10	12	14	18	29	24

dstIdx

1	5	6	7	0	3	5	4	5	7	1	5	2	7	0	1	2	3	0	7	0	2	4	6

CSR Representation



Logical Adjacency List with Tiling

- ✓ Better locality
- ✓ Partitioning intersections into smaller sub-intersections

srcPtr

0	1	3	3	4	7	8	11	12	13	17	18	20	21	21	22	24

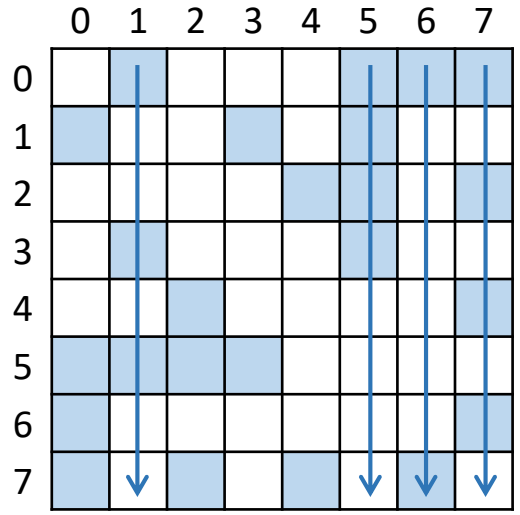
dstIdx

1	0	3	1	5	6	7	5	4	5	7	5	2	0	1	2	3	0	0	2	7	7	4	6

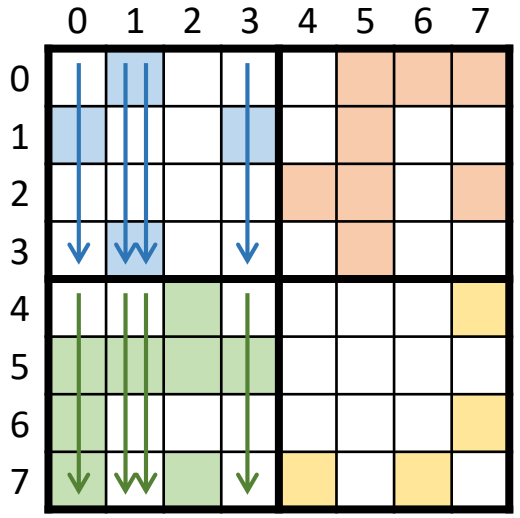
Tiled CSR Representation

Benefits of Tiling

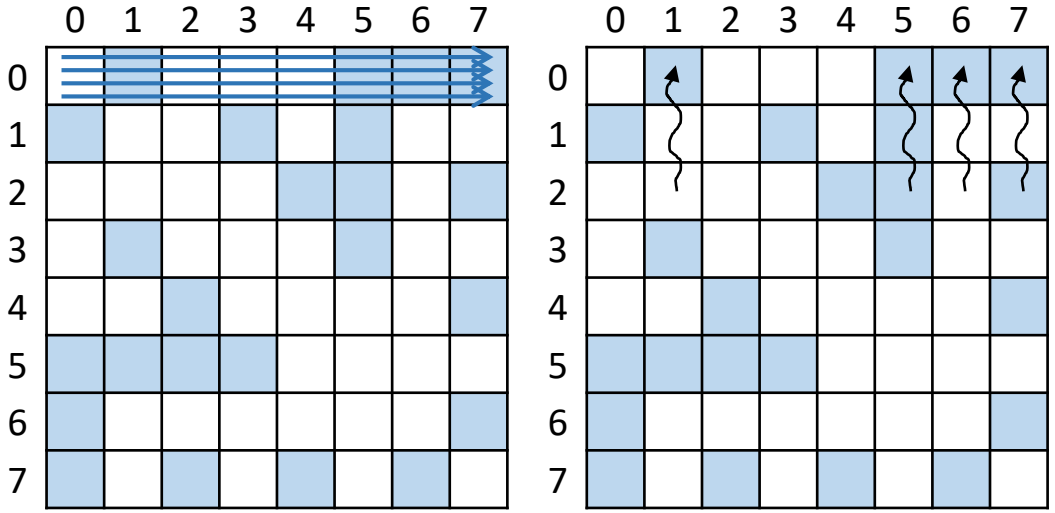
- Bad locality



✓ Good locality

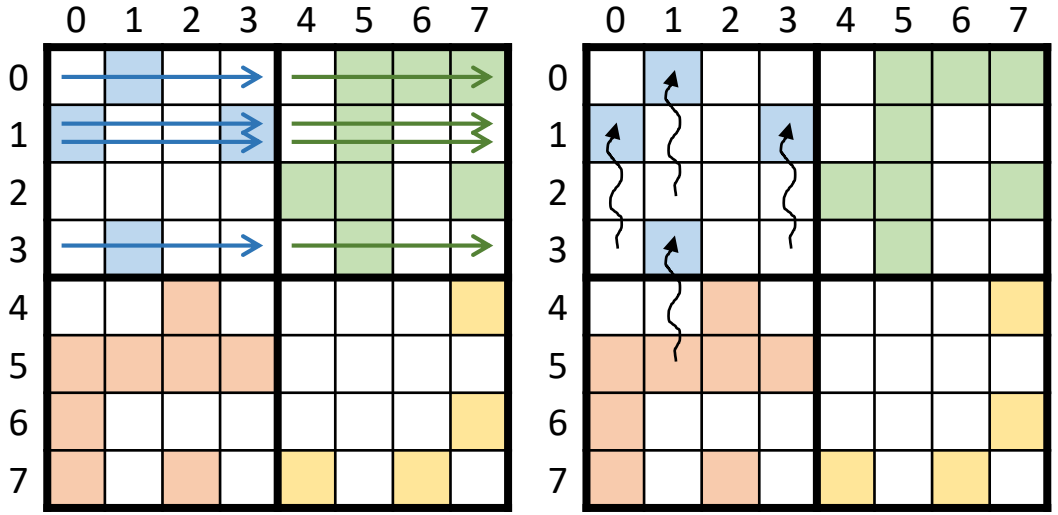


✓ Good locality



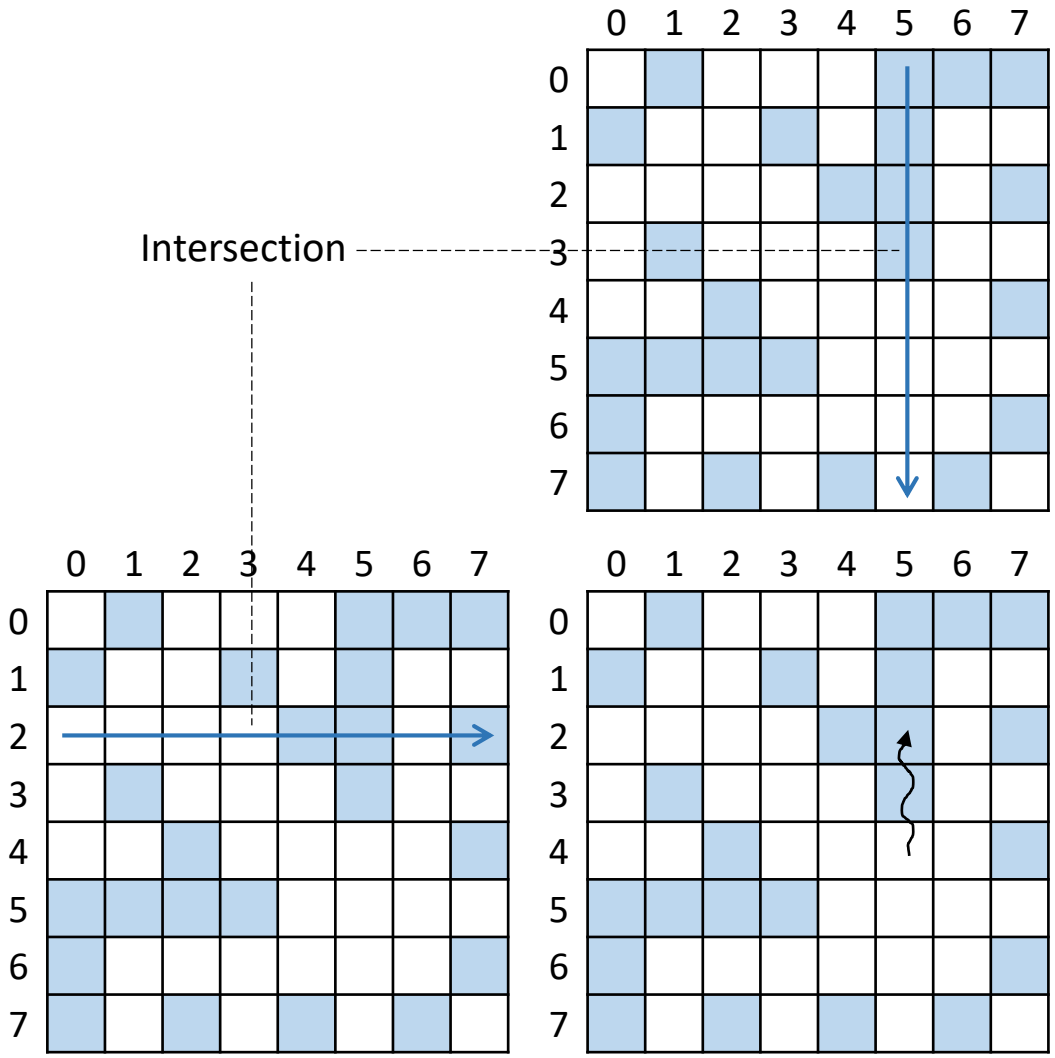
Memory Access Pattern without Tiling

✓ Good locality

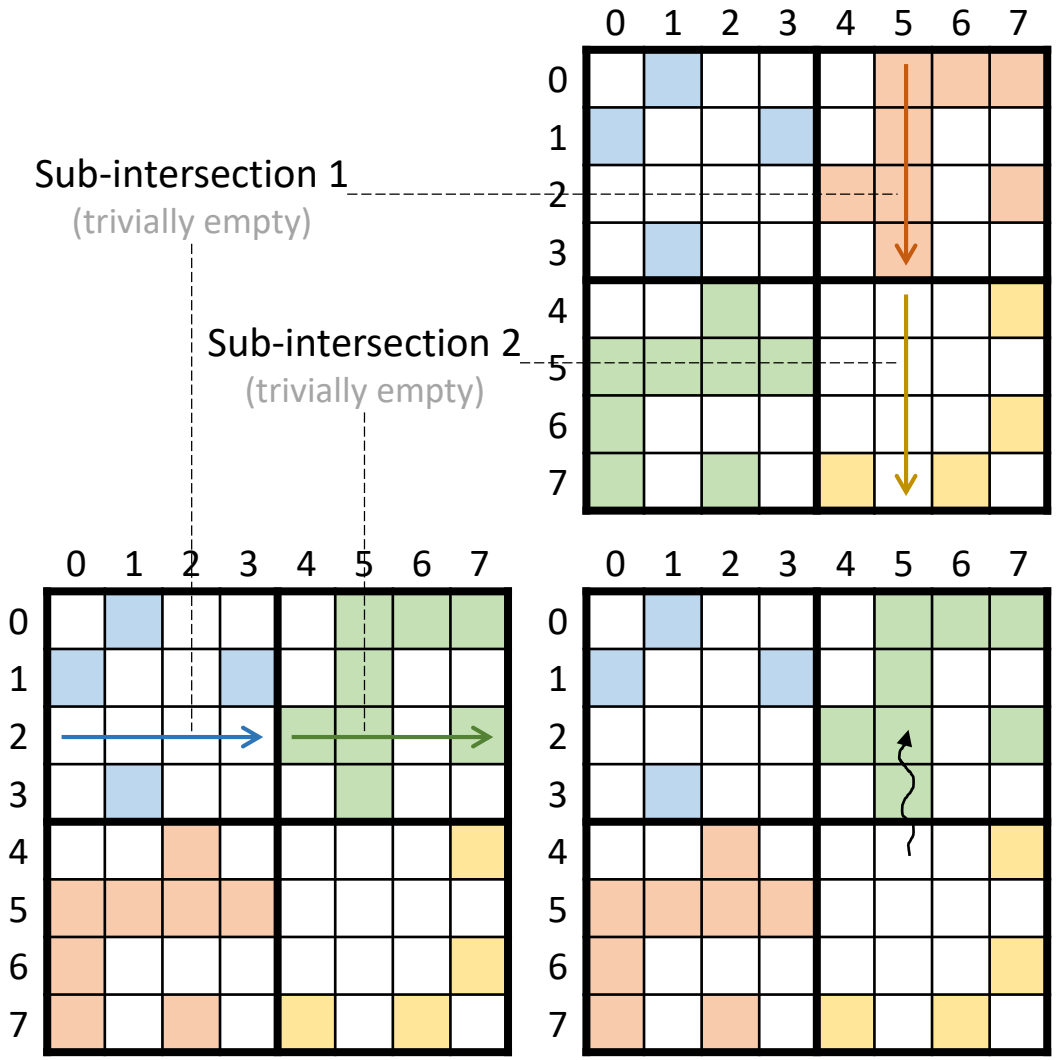


Memory Access Pattern with Tiling

Benefits of Tiling

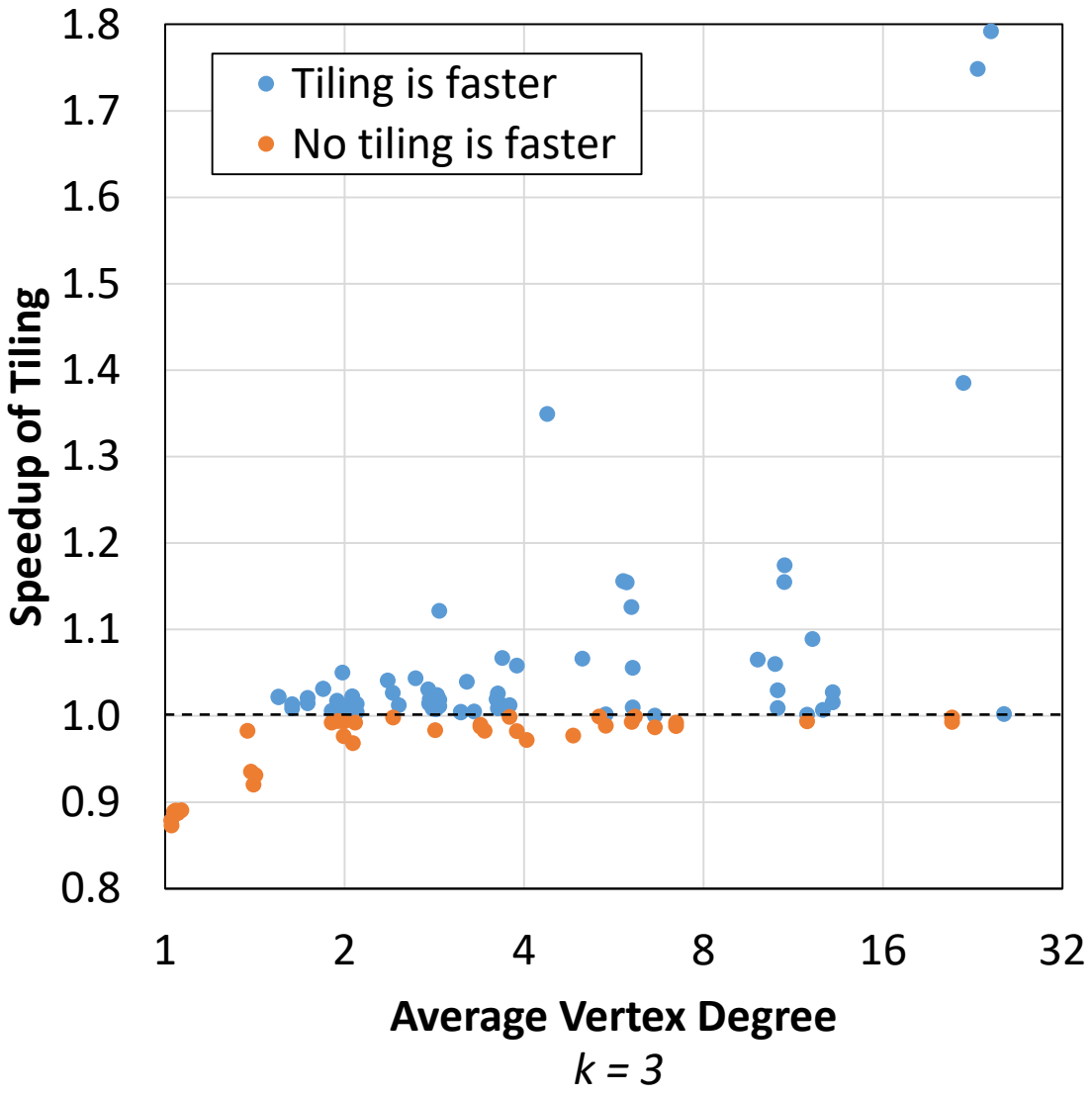


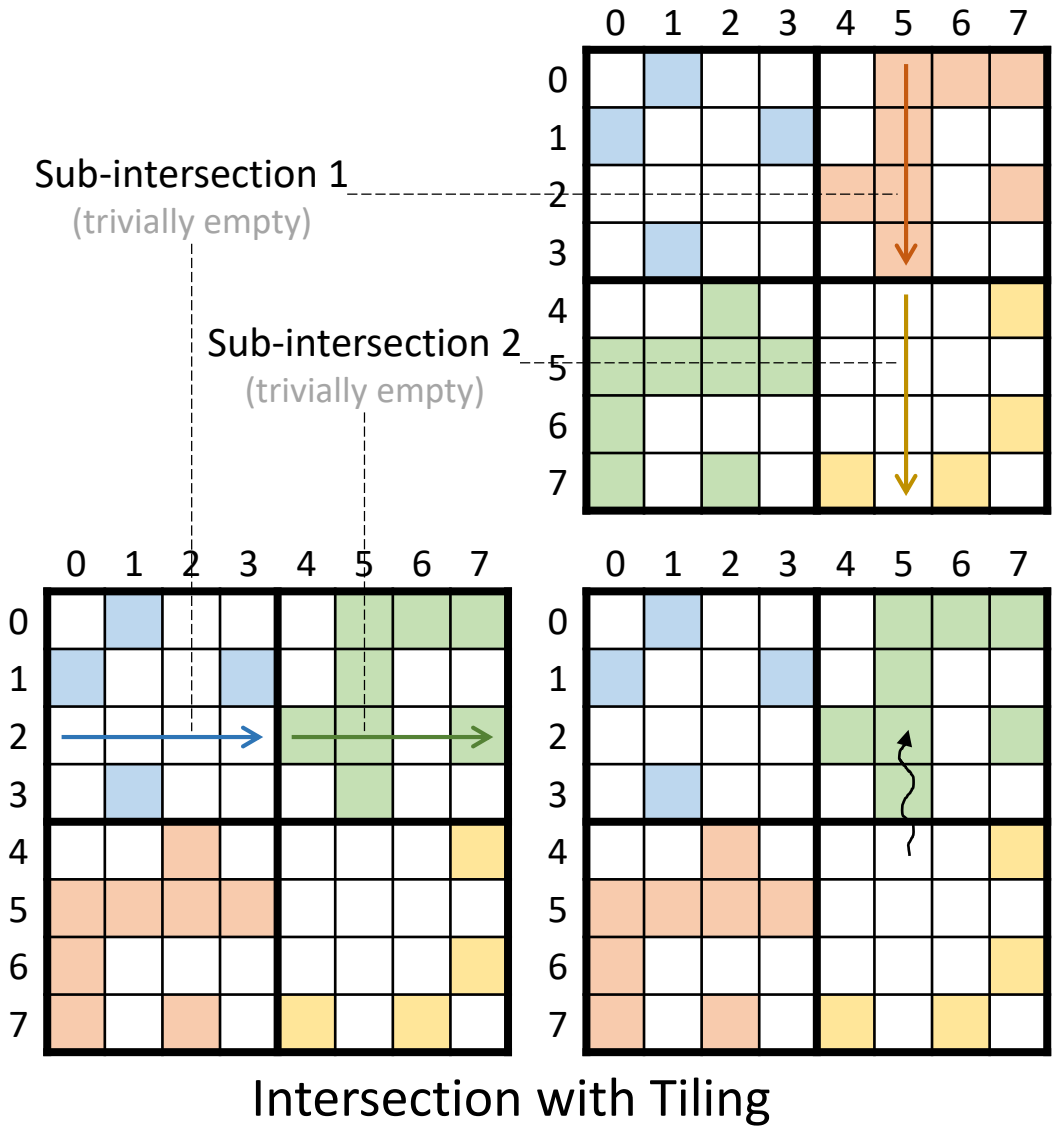
Intersection without Tiling



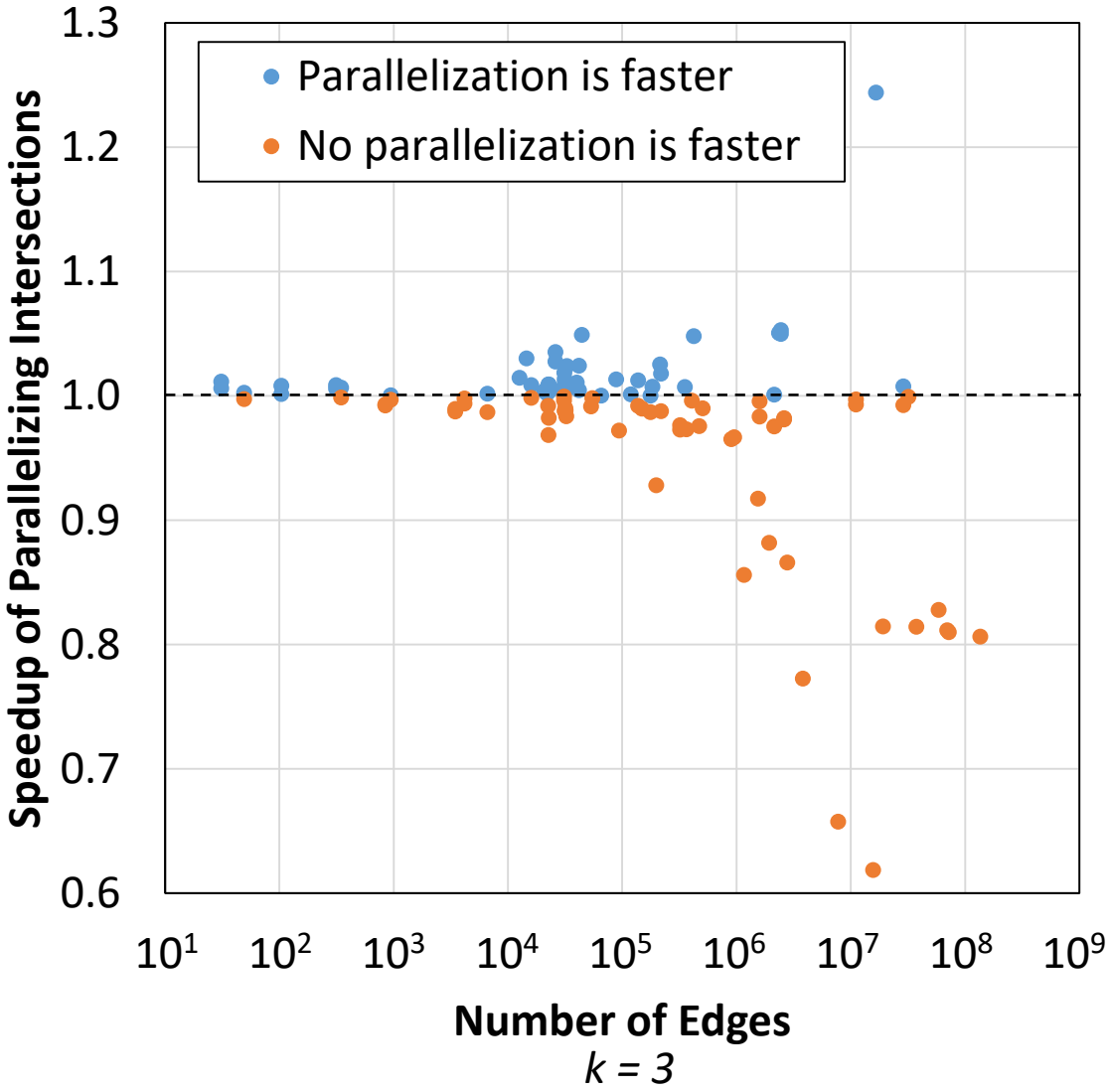
Intersection with Tiling

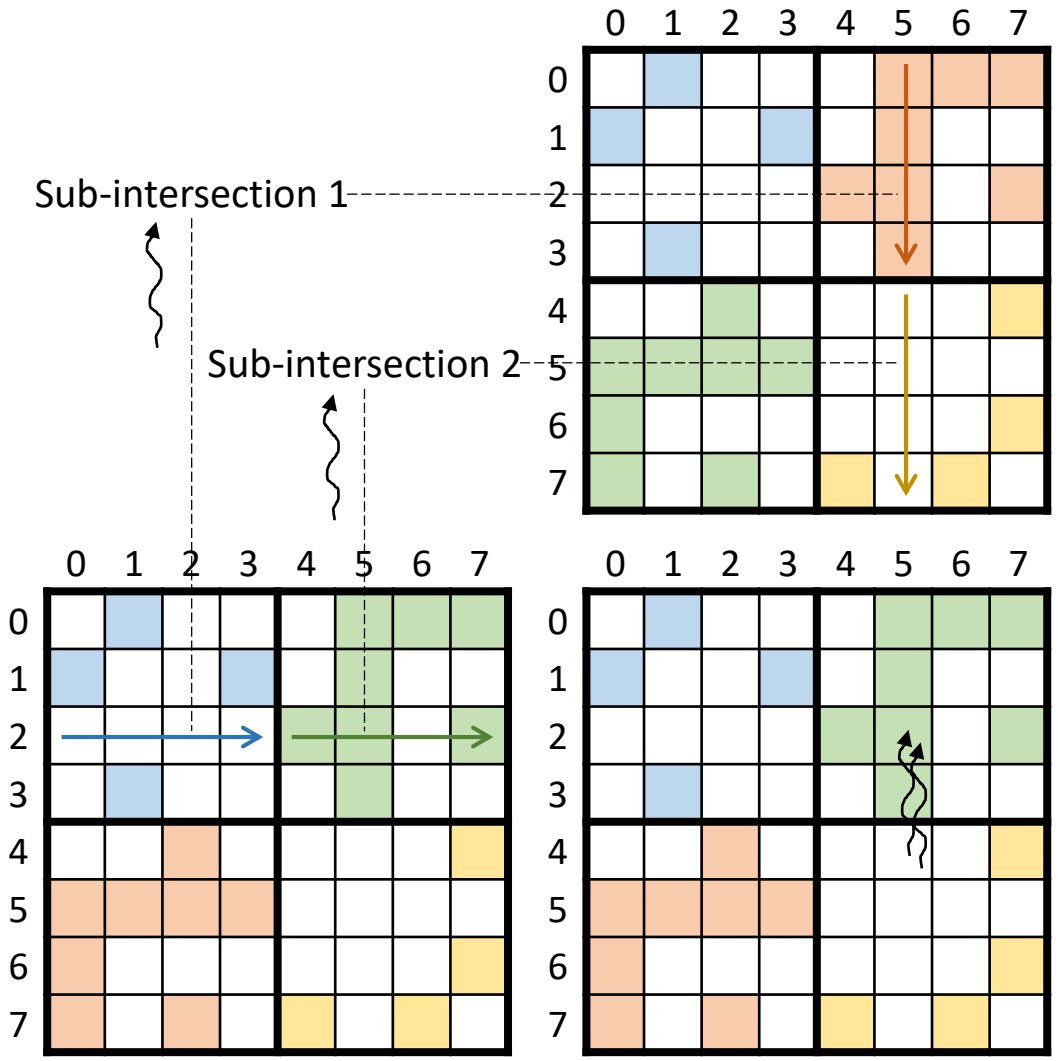
Benefits of Tiling



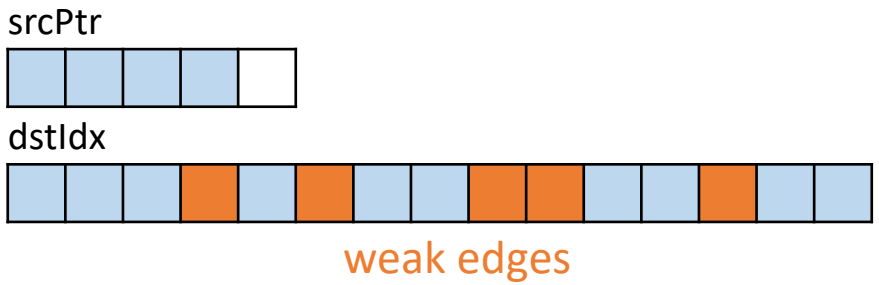


Parallelizing Intersections

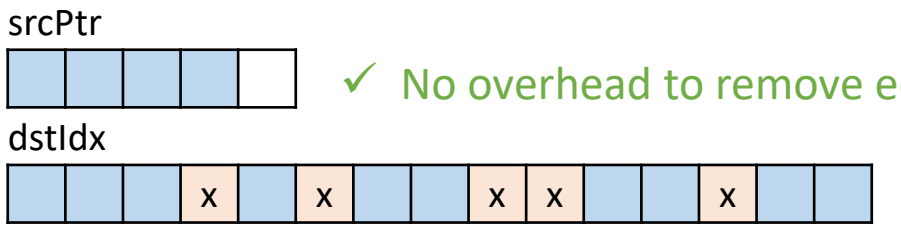




Removing Deleted Edges Intermediately

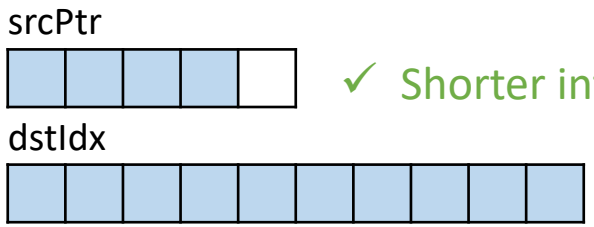


Mark deleted edges

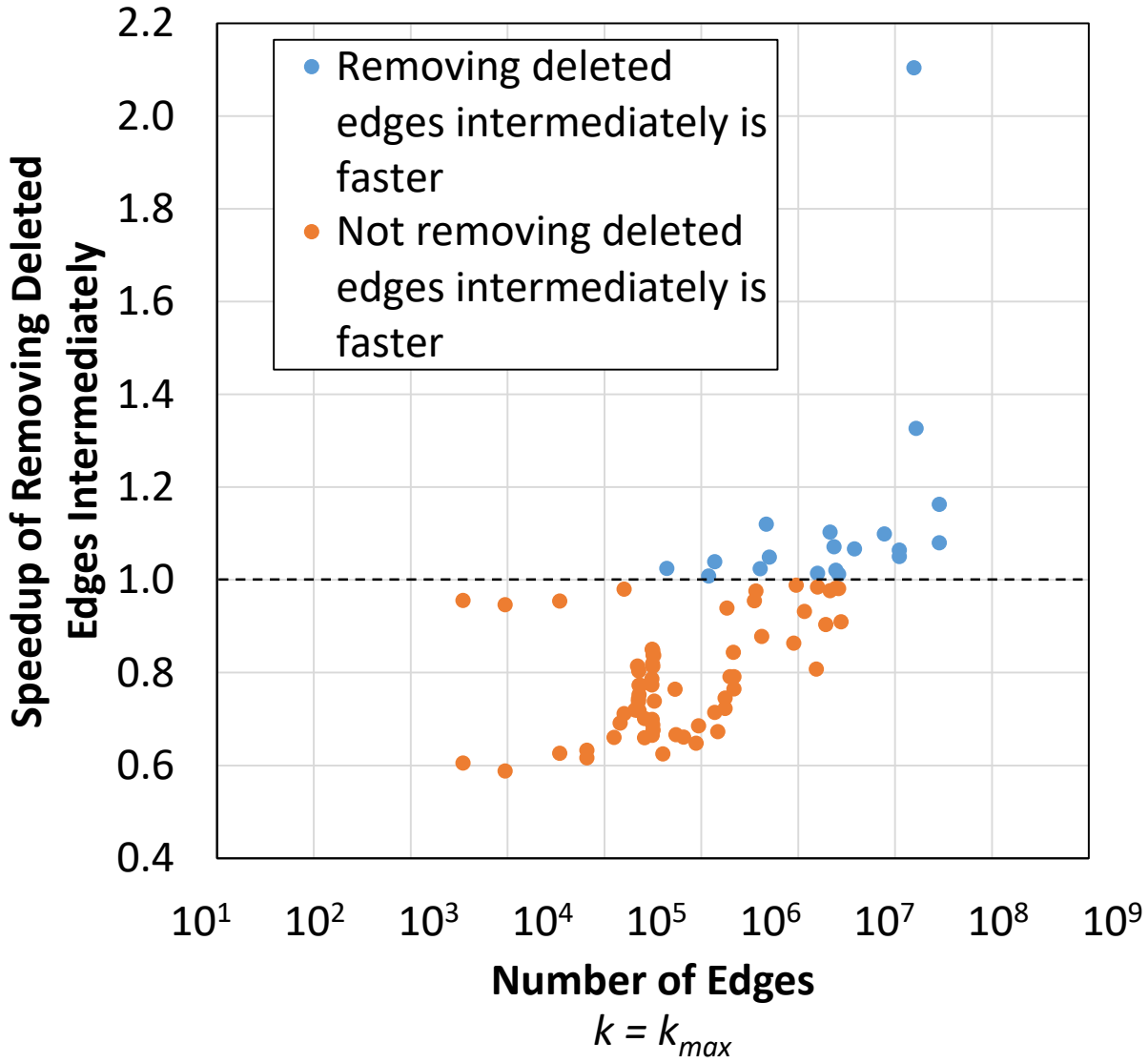


✓ No overhead to remove edges

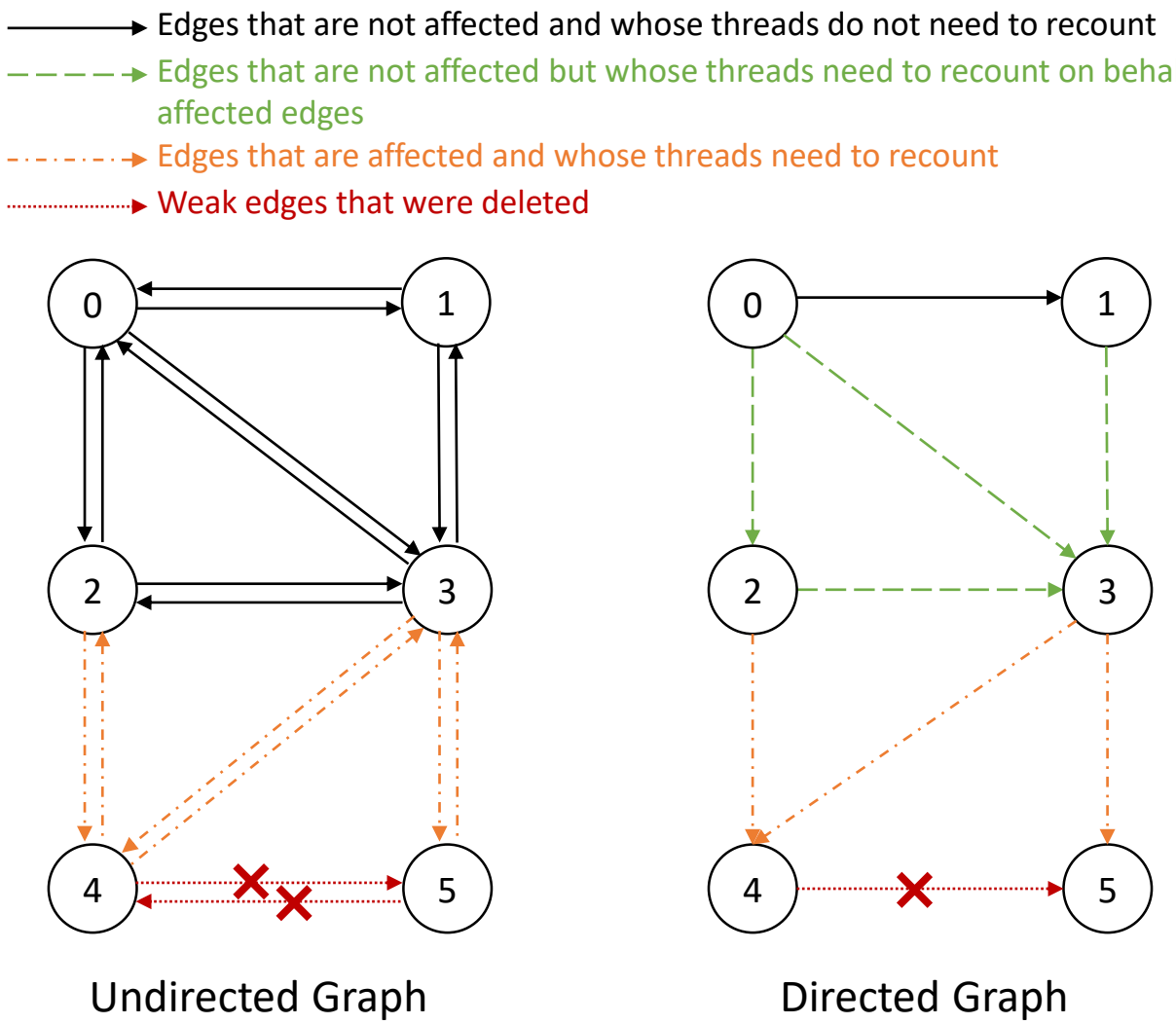
Remove deleted edges (for select iterations)



✓ Shorter intersections



Recomputing Support for All or Affected Edges



Graphs performing better with only affected edges reprocessed:

- graph500-scale20-ef16
- graph500-scale21-ef16
- graph500-scale23-ef16

For further investigation:

- Recomputing for affected edges on select iterations (later iterations)

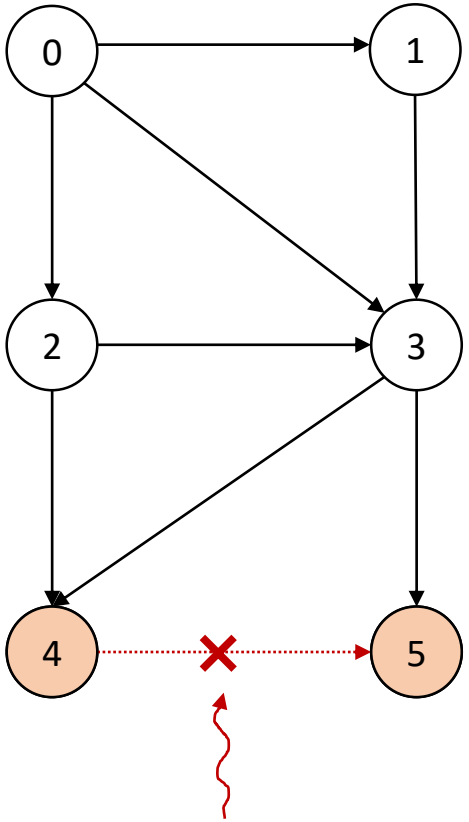
Marking Affected Edges

Edges that are not affected and whose threads do not need to recount

Edges that are not affected but whose threads need to recount on behalf of affected edges

Edges that are affected and whose threads need to recount

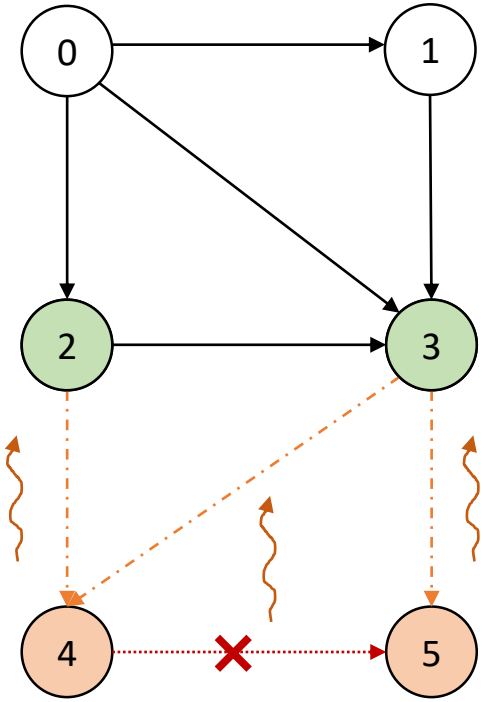
Weak edges that were deleted




```
01: parallel for e = {u, v} ∈ E do
02:     if e is deleted then
03:         mark u as affected, mark v as affected
```

Pseudocode for Marking Affected Edges

Marking Affected Edges




```
01: parallel for e = {u, v} ∈ E do
02:     if e is deleted then
03:         mark u as affected, mark v as affected
04: parallel for e = {u, v} ∈ E do
05:     if e is not deleted and (u is affected or v is affected) then
06:         mark e as affected
07:         if u is not affected then mark u as needs to recount
08:         else if v is not affected then mark v as needs to recount
```

Pseudocode for Marking Affected Edges

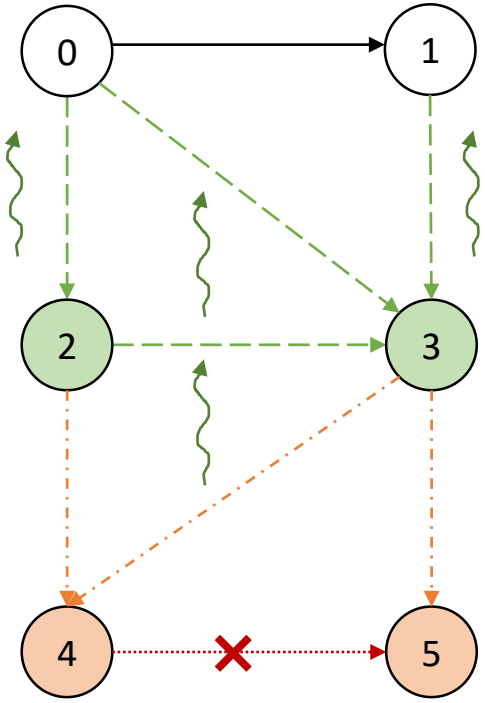
Marking Affected Edges

Edges that are not affected and whose threads do not need to recount

Edges that are not affected but whose threads need to recount on behalf of affected edges

Edges that are affected and whose threads need to recount

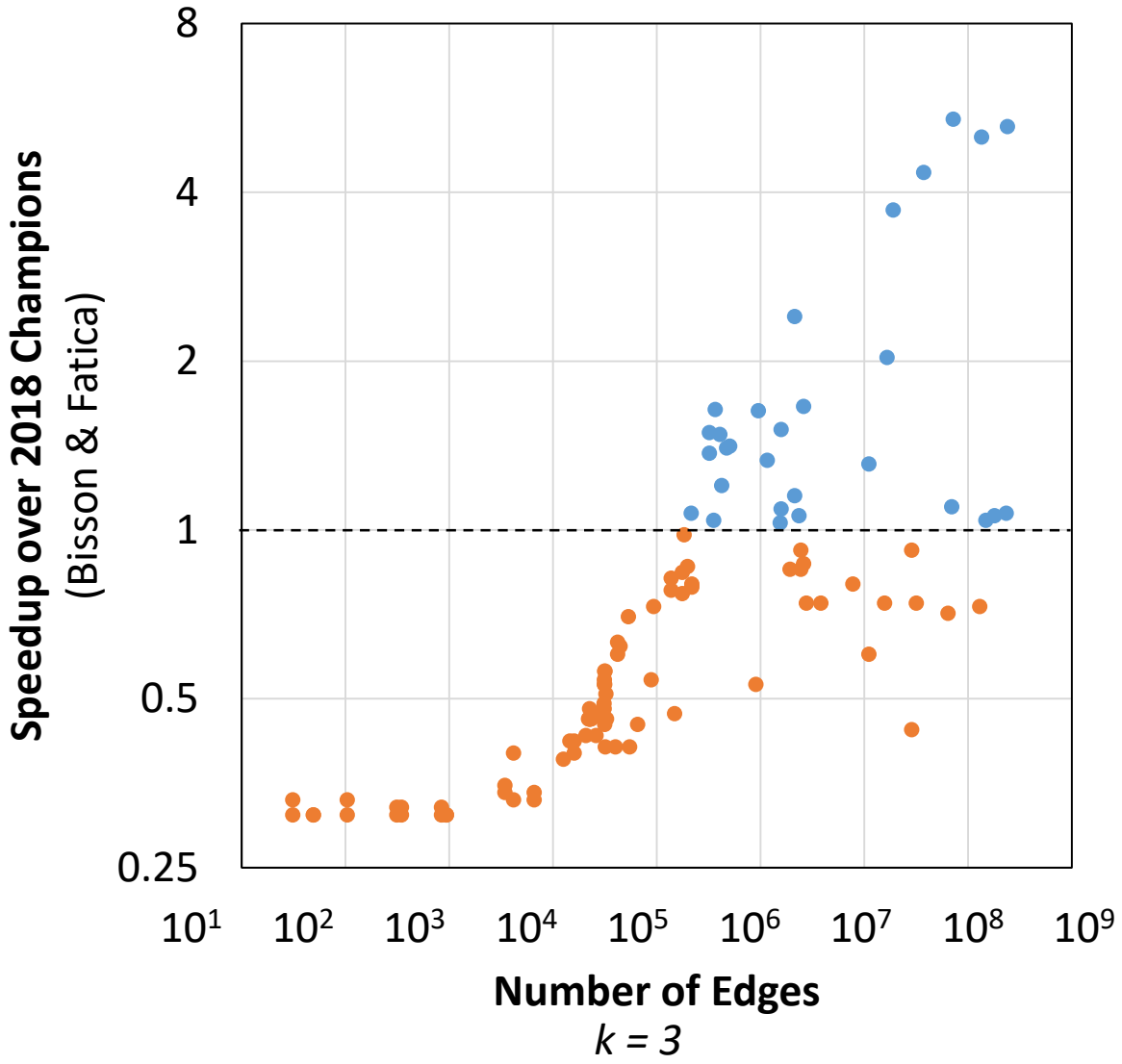
Weak edges that were deleted




```
01: parallel for e = {u, v} ∈ E do
02:     if e is deleted then
03:         mark u as affected, mark v as affected
04: parallel for e = {u, v} ∈ E do
05:     if e is not deleted and (u is affected or v is affected) then
06:         mark e as affected
07:         if u is not affected then mark u as needs to recount
08:         else if v is not affected then mark v as needs to recount
09: parallel for e = {u, v} ∈ E do
10:     if e is not deleted and e is not affected then
11:         if u needs to recount or v needs to recount then
12:             mark e as needs to recount
```

Pseudocode for Marking Affected Edges

Comparison with Prior Champions



KTrussExplorer: Exploring the Design Space of K-truss Decomposition Optimizations on GPUs

Safaa Diab, Mhd Ghaith Olabi, Izzat El Hajj

American University of Beirut

github.com/ielhajj/ktruss-explorer