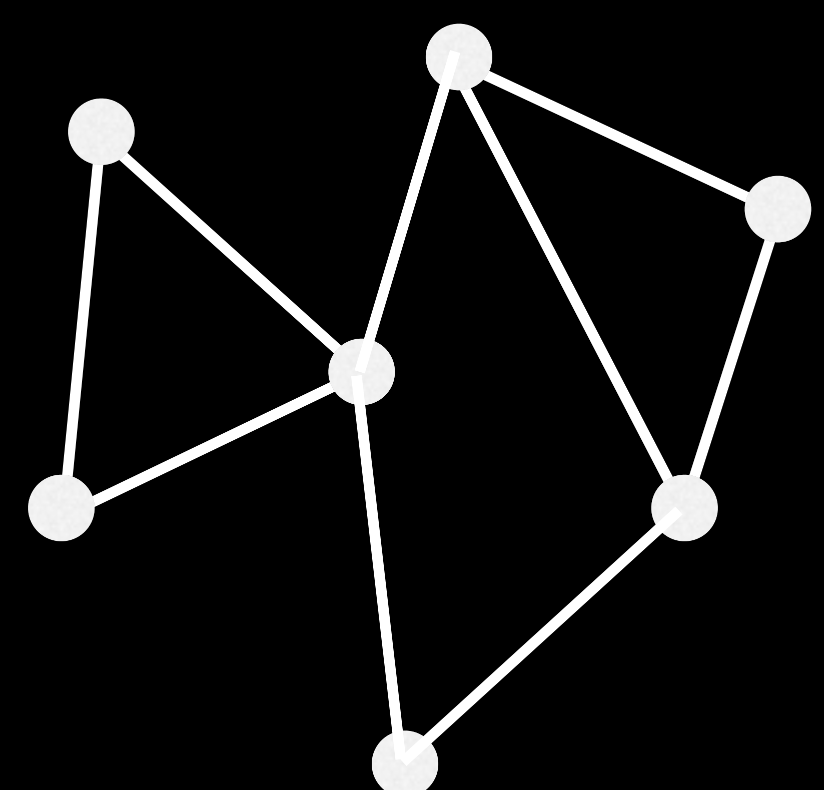
Parallel Triangle Counting in MPI

Jason Li and David Wise

Background

- A triangle in a undirected graph is a collection of 3 vertices such that all 3 pairs of vertices are connected by an edge.

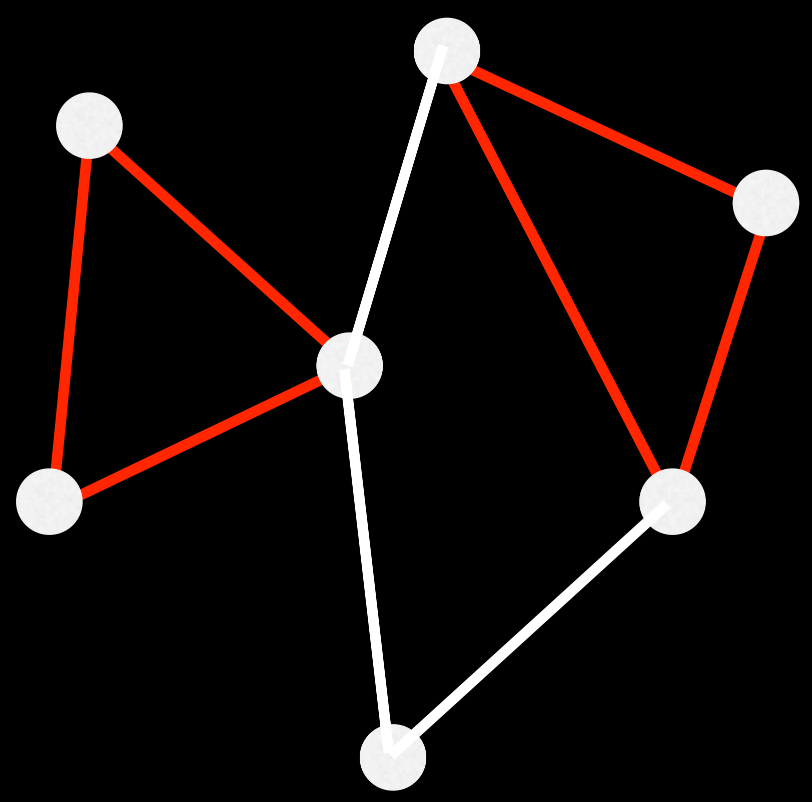
- “Triangle counting has emerged as an important building block in the study of social networks, identifying thematic structures of networks, spam and fraud detection, link classification and recommendation, and more” [1]



Background

- A triangle in a undirected graph is a collection of 3 vertices such that all 3 pairs of vertices are connected by an edge.
- “Triangle counting has emerged as an important building block in the study of social networks, identifying thematic structures of networks, spam and fraud detection, link classification and recommendation, and more” [1]

This graph has 2 triangles:



The Underlying Algorithm

- Initialize the counter to 0.
- Sort the vertices in order of increasing degree, breaking ties arbitrarily. Similarly, sort the adjacency lists according to the same ordering.
- For each edge (v, w) with $v < w$:
  - Let u_v and u_w be the first vertices in the adjacency lists of v and w respectively.
  - While u_v exists and $u_v < v$ and u_w exists and $u_w < v$:
    - If $u_v < u_w$ then set u_v to the next neighbor of v.
    - Else if $u_w < u_v$ then set u_w to the next neighbor of w.
    - Else increment the counter and set u_v and u_w to their next neighbors.

Complexity of the Algorithm

- The space complexity is just $O(m)$ since we store the graph
- Because the vertices are sorted by degree and each edge is assigned to its smaller neighbor, it can be shown that the sequential time complexity is $O(m^{3/2})$.

Parallelizing the Algorithm

- The focus of our project was efficiently parallelizing this algorithm
- Naive idea: each edge is a task and can be arbitrarily assigned to a processor
    - The catch is that to process an edge, the processor needs to know the neighbors of each vertex on the edge
    - If the edges are arbitrarily assigned, each processor needs a copy of the whole graph

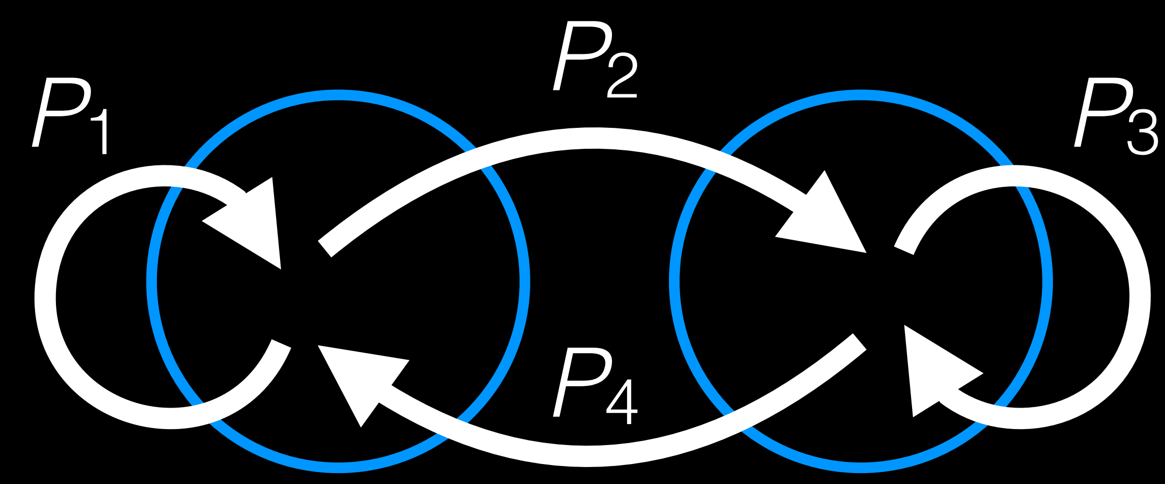
Reducing Communication

- We want the edges assigned to each processor to hit as few vertices as possible
- We can approach the problem by grouping the vertices
  - We partition the vertices into $r = \sqrt{P}$ groups $v_1, \ldots, v_r$ and assign each processor a pair (v_i, v_j)
  - The processor assigned pair (v_i, v_j) is responsible for all edges going from a vertex in v_i to v_j.

Cost Analysis

- In the average case, each processor gets $1/P$ edges: we expect near perfect speedup
- Each processor gets the adjacency lists of $2n/\sqrt{P}$ vertices, which on average has total size $2m/\sqrt{P}$

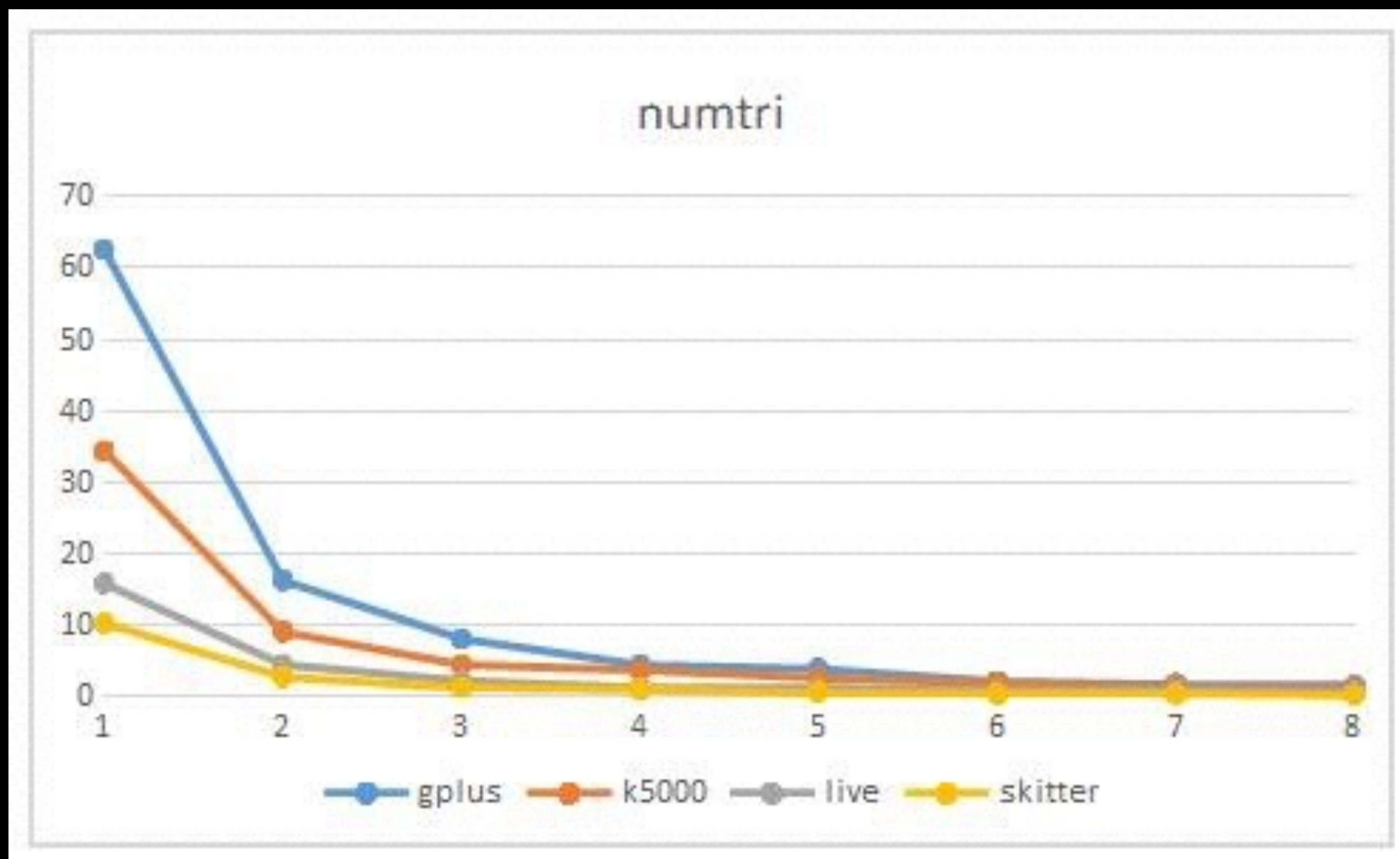
4 processors
2 groups of vertices



The thick arrows represent groups of edges.

Actual Speedups Matched Expectations

Time (s)



1 4 9 16 25 36 49 64

Number of Processors

More Results

	gplus	k5000	live	skitter
Speedup	17.23	12.72	1.95	3.98

Thank you!

Questions?

References

1. “Counting and Sampling Triangles from a Graph Stream” A. Pavan, Kanat Tangwongsan, Srikanta Tirthapura, Kun-Lung Wu, Proceedings of the VLDB Endowment VLDB Endowment Hompage archive, Volume 6 Issue 14, September 201, Pages 1870-1881.

2. “15-418 Final Report”, Shu-Hao Yu, YiCheng Qin. http://www.cs.cmu.edu/afs/cs/user/shuhaoy/www/Final_Project.pdf.