PARALLEL ALGORITHMS FOR BUTTERFLY COMPUTATIONS

GROUP MEMBERS:

ARSHIQ - 22I1023
REHAL SAEED - 22I1071
SAMRA MASHAAM - 22I0757

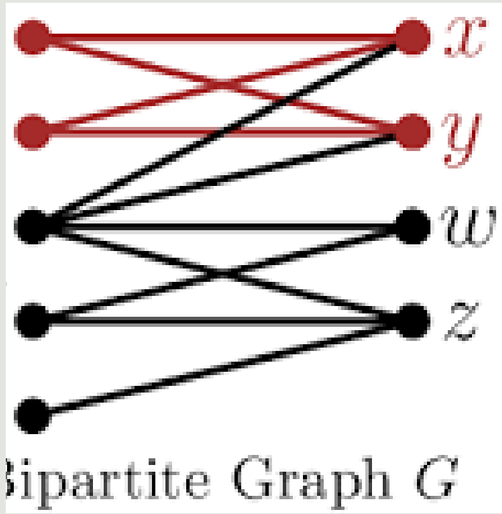
INTRODUCTION

- Graph analysis allows us to find useful patterns within them - called ‘motifs’
- When applied to real-life scenarios, such as a network, they can be used to find dense communities or detect spam.
- But not all graphs can be broken down using the same method

OVERVIEW

Bipartite Graphs:
Consists of two sets of vertices. Only vertices belonging to opposite sets are connected with an edge.

Butterfly:
Involving 4 vertices, two from each set, which map to each other completely.



Bipartite Graph G

Vertices $\{a, b, x, y\}$ form a butterfly

Set of butterflies:

$\{\{a, b, x, y\},$
$\{b, c, x, y\},$
$\{a, c, x, y\},$
$\{c, d, w, z\}\}$

CONT.

Wedge:
A path consisting of two edges and three vertices, with one vertex being a central node and the other two being endpoints.

Peeling:
A procedure to iteratively ‘peel’ off parts of a graph, starting from the least-connected subgraphs to reach the core connections.

PROBLEM

- In Unipartite graphs, triangle-based algorithms are used to detect non-trivial subgraphs.
- But since triangles cannot occur in Bipartite graphs, butterfly-based algorithms are needed.
- Butterflies are more complex than triangles and so, their detection and computation needs more time and resources

OBJECTIVE

- Reduce the overall time taken to traverse a Bipartite graph, detect all unique butterfly subgraphs, and use them to find the densest parts of the graph

- This is achieved using paralellization

“PARBUTTERFLY”

AN EFFICIENT FRAMEWORK FOR ANALYZING DENSE PATTERNS IN BIPARTITE GRAPHS

BASED ON THE WORK BY JESSICA SHI AND JULIAN SHUN

MAIN STEPS

1. Find Wedges
2. Group Wedges
3. Aggregate Wedge Info
4. Rank Vertices
5. Peeling for Dense Structures

PREPROCESSING

- Every vertex is given a unique “rank”. Rank can be based on aspects such as degree.
- Efficient ranking can help reduce workload later on
- Sort the graph via rank, resulting in a generalized graph

Both ranking and sorting can be parallelized

1. FIND WEDGES

A butterfly consists of two wedges with the **same endpoints** and **different central vertices**

Method:

- Pick a set and loop through each of its vertices.
- Locate and store all 2-hop paths

Since each vertex can be processed independently, this step can be **parallelized**.

2. GROUP WEDGES

Group wedges that have the same endpoints (u1, u2)

Method:

- Group wedges by endpoints
- Count how many share the same pair

Grouping massive wedge arrays into endpoint pairs is a compute-heavy task that benefits hugely from **multicore execution.**

3. BUTTERFLY COUNTING

If multiple wedges **share endpoints**, they can form butterflies.

Compute:

- Per-vertex count
- Per-edge count
- Global count

Each pair (u1, u2) and its wedges can be processed independently, with **Atomic addition** being used for safe concurrent updates.

4. PEELING

Peeling can be done via **tips** (vertices) or **wings** (edges)

Method:

- Remove all nodes/edges with the current minimum butterfly count
- Recalculate how many butterflies are affected
- Update butterfly counts for neighbors

Parallelism: **YES, but iterative**

- Each round of peeling is parallel, but rounds happen sequentially due to dependencies.

OUTPUT

We assign a "tip number" to each vertex (or "wing number" to each edge) based on when it got peeled.

These numbers tell you how deeply embedded that vertex/edge is in the butterfly structure of the graph.

EXPERIMENT

Machine: AWS m5d.24xlarge EC2 instance

- 48 cores (Intel Xeon Platinum 8175, 3.1 GHz)
- 384 GiB RAM

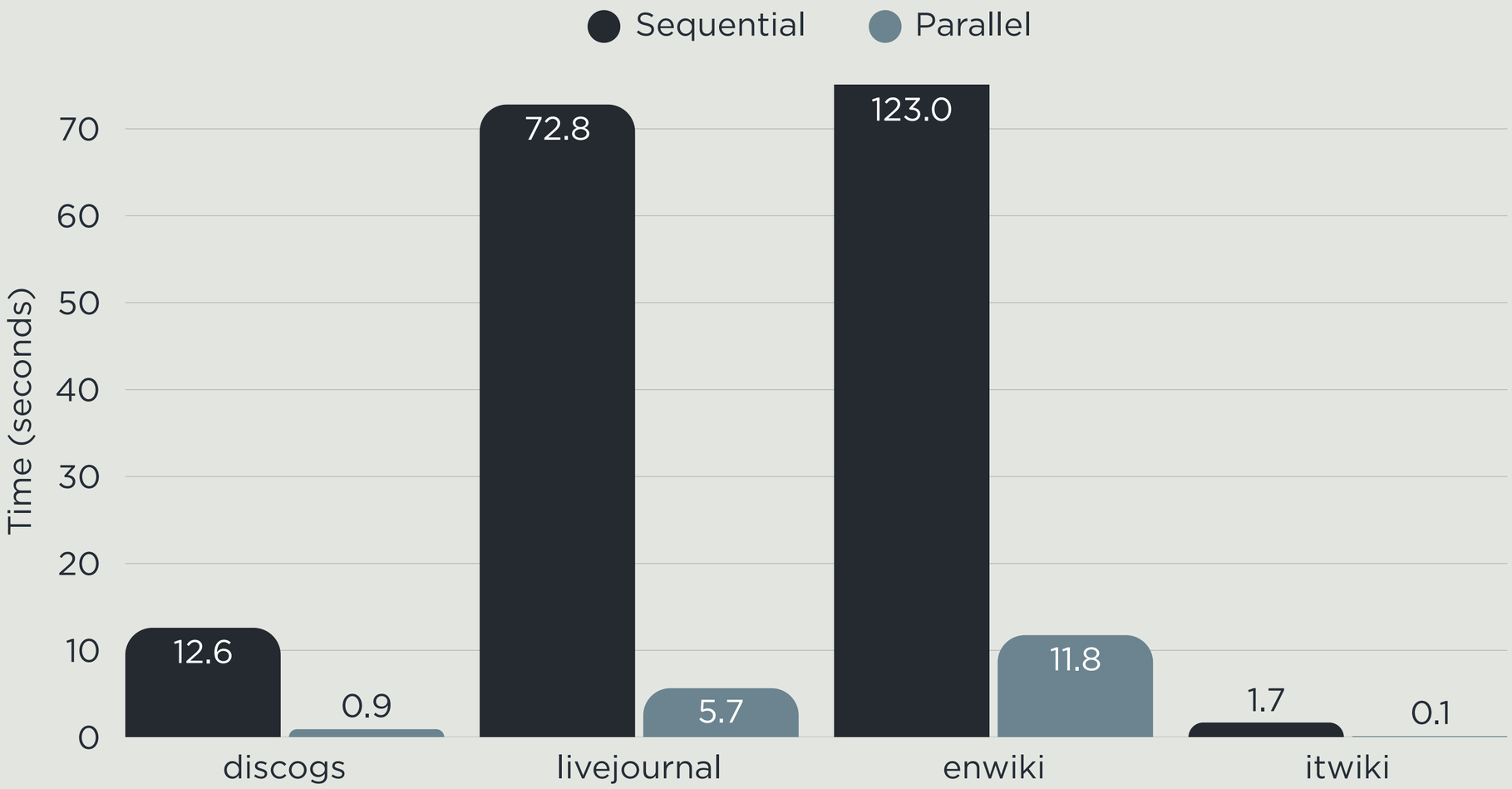
Compiler & Parallel Runtime:

- Compiled with g++ -O3
- Used Cilk Plus work-stealing scheduler

Datasets: Real-world bipartite graphs from KONECT

- Cleaned by removing self-loops and duplicate edges

RESULT


Sequential
Parallel
Time (seconds)
70
60
50
40
30
20
10
0
12.6
0.9
72.8
5.7
123.0
11.8
1.7
0.1
discogs
livejournal
enwiki
itwiki

CONCLUSION

Summary:

- Efficiently finds and analyzes dense substructures
- Highly parallel, scalable to large graphs

Why PARBUTTERFLY?

- Much faster than general-purpose subgraph algorithms
- Modular, fast, and flexible.
- Great for analyzing real-world networks like recommendation systems or spam detection.

Thank You

Any Questions?