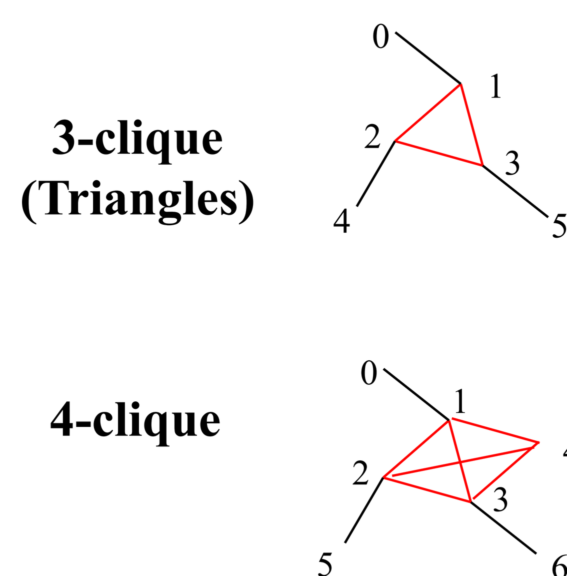
GPU Accelerated Multi-predicate Join Algorithms for Listing Cliques in Graphs

Haicheng Wu[1], Daniel Zinn[2], Molham Aref[2], and Sudhakar Yalamanchili[1]

[1] Georgia Institute of Technology, [2] LogicBlox Inc.

Listing Cliques

- Key ingredients for many graph algorithms such as
  - Triangular clustering
  - Cohesive subgraph
- Extensive attention from
  - Graph theory
  - Database
  - Network analysis



Motivations

- Empower domain users to use general declarative language (e.g., SQL or Datalog)
- Multi-predicate join can
  - Reduce data movement between binary joins
  - Reduce data reorganization (sorting or hash table construction)

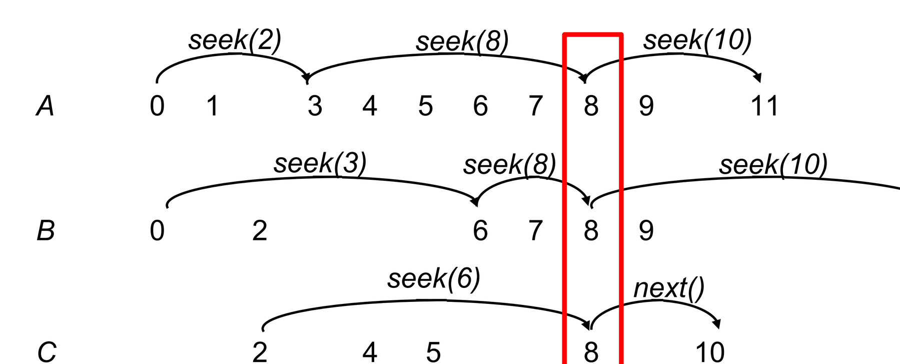
Datalog Rule

triangle(x,y,z)<-E(x,y),E(y,z),E(x,z), x<y<z.

Multi-predicate Join

4cl(x,y,z,w)<-E(x,y),E(x,z),E(x,w),E(y,z),E(y,w),E(z,w), x<y<z<w.

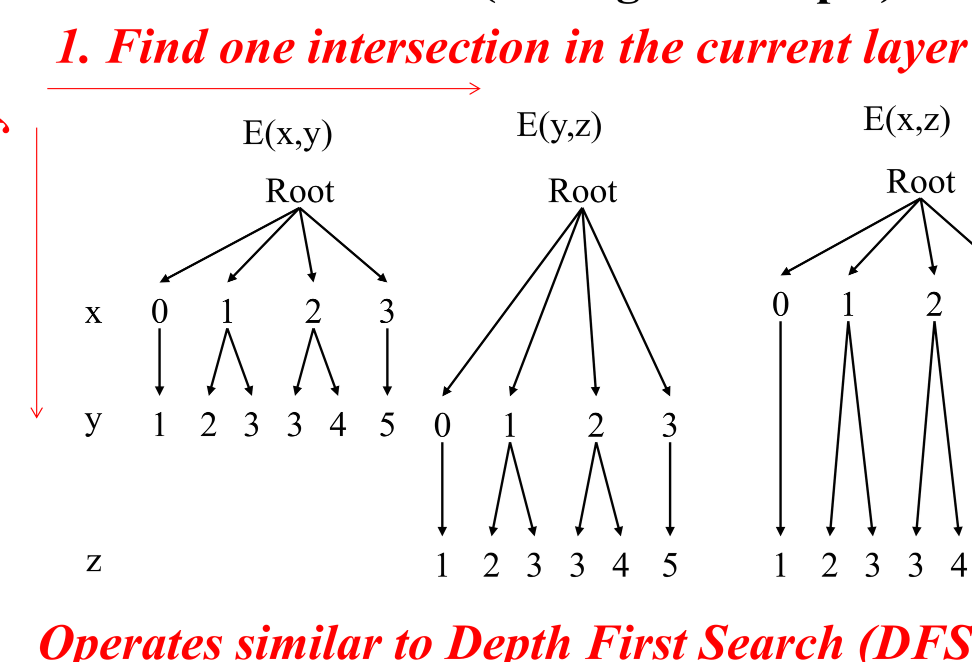
LFTJ on 3 unary relations (intersection)



Goal: Porting LeapFrog TrieJoin to GPUs

- LeapFrog TrieJoin (LFTJ)[1]
  - A general multi-predicate join algorithm
  - Worst case optimal
  - Sequential algorithm

LFTJ on 3 tries (Triangle Example)

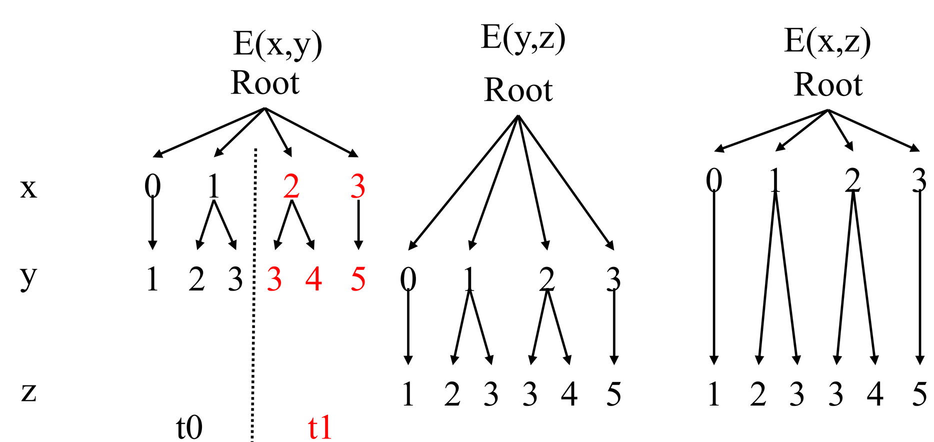


Operates similar to Depth First Search (DFS)

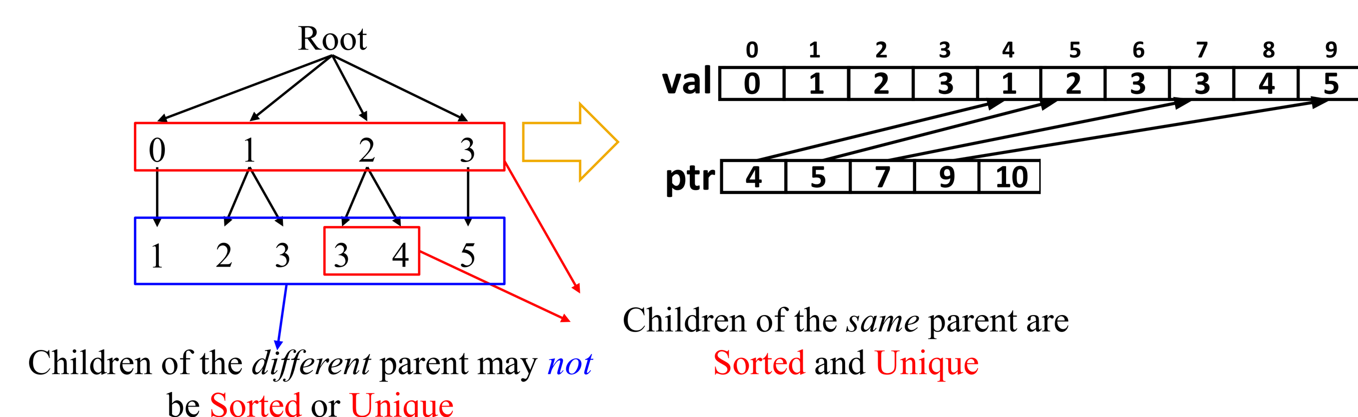
Algorithm 1: LFTJ-GPU

- Split the first trie and map sub-trees to parallel threads
- Parallel threads still run sequential LFTJ
- Intersections are implemented as binary searches
- This method also works for CPU parallelization (LFTJ-CPU)

Example: Mapping to 2 Parallel Threads



Data structure: CSR

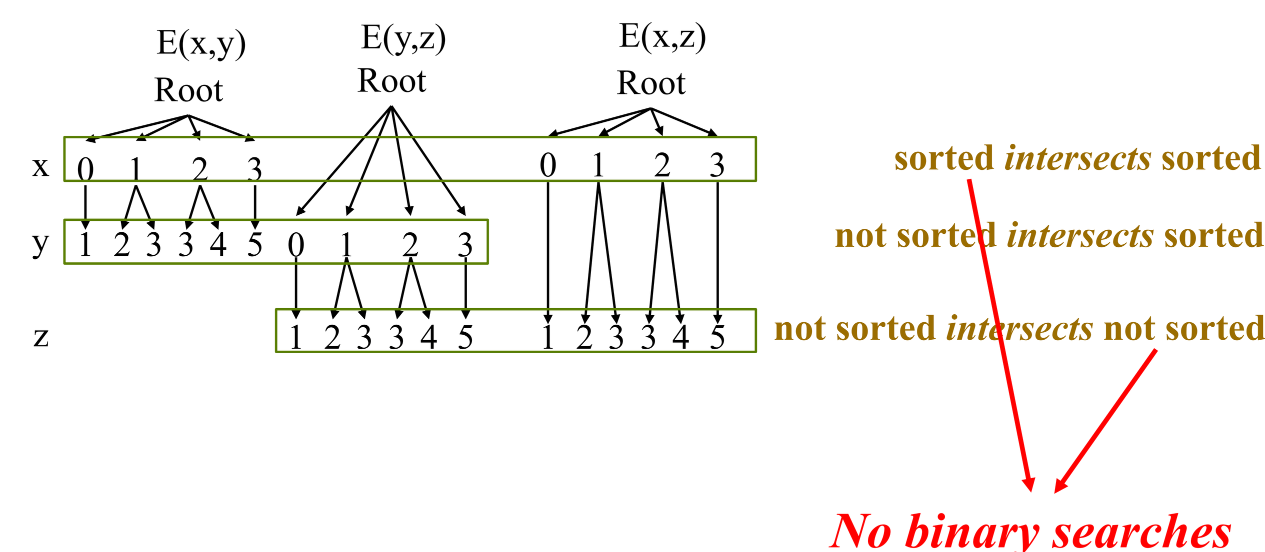


Algorithm 2: GPU-Optimized

- Same data structure as Algorithm 1
- Operates breadth-first rather than depth-first
  - Pro: More parallelism
  - Con: Larger memory footprint
- Built on top of ModernGPU library [2]
  - Uses Merge-Path framework to partition data for CTAs/threads
    - load-balancing between CTAs/threads
    - Good memory usage control
  - Optimized for coalesced memory accesses.
- Rely on sorted and unique property to reduce binary searches

BFS Exploration

- Intersects layer by layer from the top to the bottom
- Divide the problem into
  - Parallel node expansion
  - Parallel array intersection



Memory Usage Control

- Join output size depends on input data
  - Maybe empty or maybe much larger than the input size
- Naive Implementations are inefficient
  - Conservatively reserve large memory to meet the worst case
  - Reserve small memory, abort when not enough
- In Algorithm 2, each input computes its output size first
  - Partition the data by their output size for kernels
  - This is also needed by Merge-Path Framework

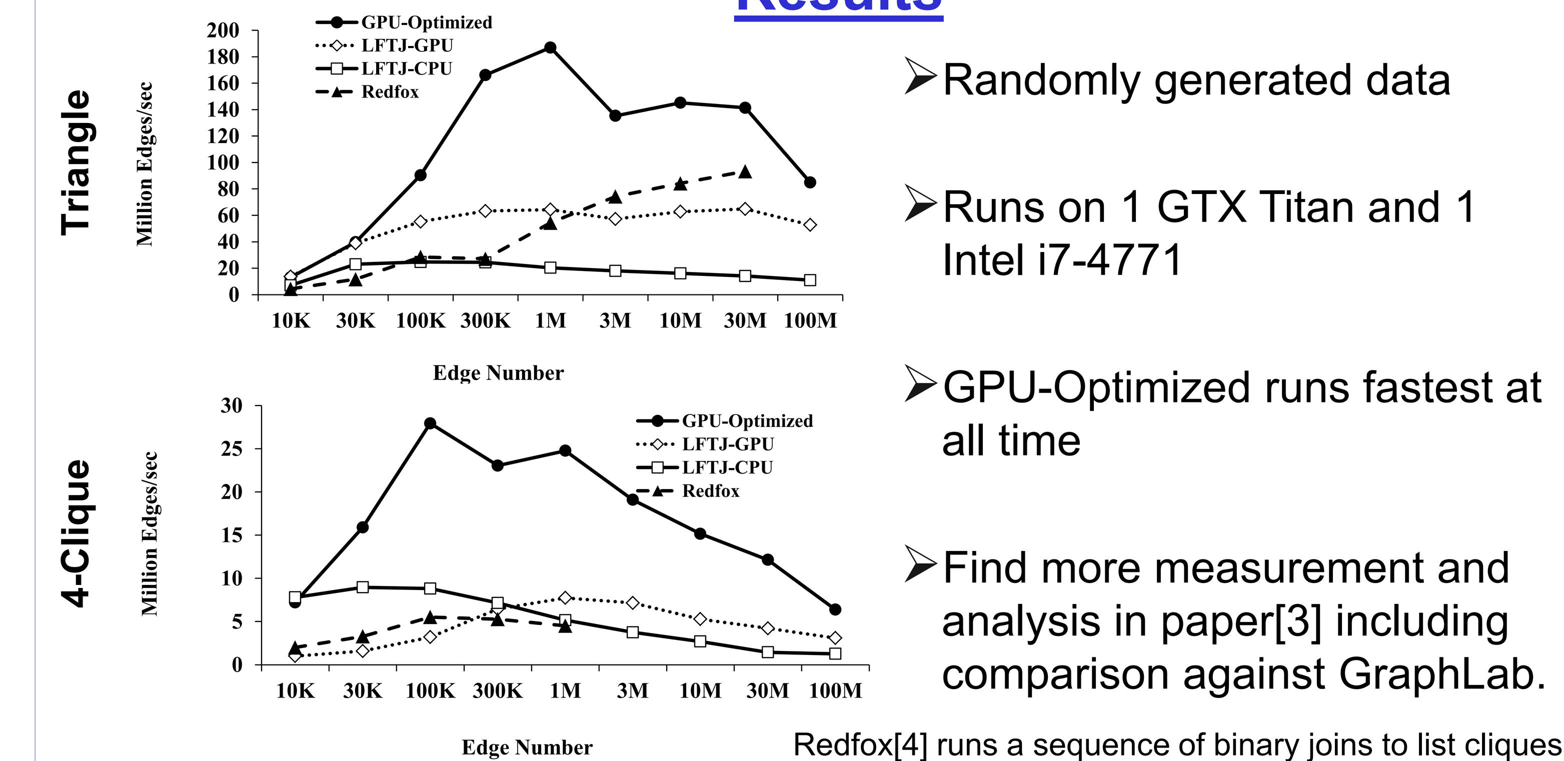
Free mem: 10

	p1		p2		p3	
Output size generated by each input	3	4	8	2	5	3
prefix sum	0	3	7	15	17	20

Total output: 23

Use 3 partitions

Results



- Randomly generated data
- Runs on 1 GTX Titan and 1 Intel i7-4771
- GPU-Optimized runs fastest at all time
- Find more measurement and analysis in paper[3] including comparison against GraphLab.

Redfox[4] runs a sequence of binary joins to list cliques

References

[1] Todd L. Veldhuizen. *Triejoin: A Simple, Worst-Case Optimal Join Algorithm.* ICDT 2014.
[2] Sean Baxter. *Modern GPU Library.* http://nvlabs.github.io/moderngpu/index.html
[3] H. Wu, D. Zinn, M. Aref, S. Yalamanchili. *Multipredicate Join Algorithms for Accelerating Relational Graph Processing on GPUs.* ADMS 2014.
[4] H. Wu, G. Diamos, T. Sheard, M. Aref, S. Baxter, M. Garland, S. Yalamanchili. *Red Fox: An Execution Environment for Relational Query Processing on GPUs.* CGO 2014.