Tigr: Transforming Irregular Graphs for GPU-Friendly Graph Processing
Sadet, Qiu, Zhao (2018)

Presented by Edward Fan
Real-world graphs are irregular

- Power-law or not, large imbalances in vertex degree are common

- Lots of frameworks that we’ve seen have worked around this
  - Partitioning the graph by edges
  - Splitting work into sub-tasks
  - Duplicating vertices across nodes
Real-world graphs are irregular

- For GPUs, the issue is even more important
- Threads are organized in warps
  - A warp is SIMD-like- a group of threads execute the same instructions in parallel over different data
  - There is implicit synchronization- all threads must complete before execution can continue
- If vertices are assigned to threads, this can lead to a major bottleneck
Tigr: vertex transformations

- Basic idea: split a high degree node into multiple nodes
- Many possible connections, but tradeoffs between propagation time, edge count, and vertex count
- Solution: uniform-degree tree transformation (UDT)
Correctness of split nodes

- Many algorithms still work correctly on the split nodes
- Can assign internal edges zero weight (or equivalent)
  - Just need to treat split vertex as if it were a single vertex
- Total in/out-degree does not change, so PageRank still works
  - Sum the split nodes
Virtual transformation

- Benefits of vertex splitting, without having to synchronize!

- Slightly modified CSR serves this purpose well
Virtualization details

- Use atomic operations to resolve concurrent writes
  - For push execution: no correctness difference (as every vertex needs to be able to accept concurrent pushes)
  - For pull execution: writes must be associative, as neighbors are processed in parallel

- Clever trick for GPU coalescing
  - A normal split would result in strided accesses
  - Solution: *stride the split!*
  - Leads to sequential access
Performance

- On single GPU, beats CuSha and Gunrock most of the time
- Caveat: custom code vs framework limitations
Questions?

Thanks!