Tigr

paper: [http://www.cs.ucr.edu/~anode001/asplos18.pdf]

code: [https://github.com/AutomataLab/Tigr]

Important Concepts:

- power law: a small portion of nodes own a large number of neighbours while most nodes are connected to only a few neighbors (high irregularity of degree distribution)
- effectiveness: the transformed graphs' irregularity can be effectively reduced
- correctness: algorithms should have the same results on the transformed graphs
- efficiency: minimize the transformation cost
- node splitting: first identify nodes with high degrees, then iteratively split the nodes with high degrees, then iteratively split the nodes until their degrees reach a predefined limit
- transformation tradeoff: extent of irregularity reduction <-> convergence rate of graph algorithm
- vertex-centric programming: computations are defined from the view of a vertex rather than a graph, computations of different vertices are synchronized at the graph level, iteration by iteration, until a certain number of iterations or a convergence property is met (BSP model)
- algorithm scheme:
    - push-based: update neighbors' values through outgoing edges -- outdegree
    - pull-based: gather neighbors' values through incoming edges -- indegree
- degree bound: predefined degree threshold K, if $d(V) > N$, v is a high-degree node
- split transformation:
    - I: internal split node set
    - B: boundary split node set
    - $I \bigcup B$ refered as a family
    - residual node: split node with degree less than K
    - connecting strategy:

	space cost	irregularity reduction	value propagation rate
clique connection	high	low	fast
circular connection	low	high	slow
star-shaped connection	low	varies(hub node high)	fast

Key Idea:

	space cost	irregularity reduction	value propagation rate

- transform irregular graphs into more regular ones such that the graphs can be processed more efficiently on GPU-like architectures while guaranteeing correctness
- *Uniform-Degree Tree transformation* (UDT):

  Feature: transforms a high-degree node into a tree structure with nodes of identical degrees

  - distances among split nodes only increase logarithmically as the degree of the to-split node increase
  - preserve basic graph properties
  - ensure at most one residual node in the generated family (introduces new split nodes on demands)
    - recursive star-trans will introduce many residual nodes, which:
      - compromise the irregularity reduction
      - introduce unnecessary split nodes

  Property:

  - output forms a tree structure where the degree of each node (or except the root) equals to K
  - there exits a unique path connecting the incoming edges of the original node to each of its outgoing edges

  Correctness: depends on the graph properties that graph analyses rely on

  - graph connectivity: preserved'
  - path property
    - add-based analyses: zero weighted UDT-introduced edges
    - min-based analyses: infinity weighted UDT-introduced edges
  - degree-based analyses: see challenge
  - neighborhood-based analyses: may fail to preserve (the word "may" is interesting)

  Pseudocode

```pseudocode
/** ------------------ **/
/** UDT Transformation **/
/** ------------------ **/
if degree(v) > K then                  /** for each high-degree node **/
    q = new_queue()
    for each v_n from v's neighbors do
        q.add(v_n)                     /** add all original neighbors **/
        v.remove_neighbor(v_n)
    while q.size() > K do
        v_n = new_node()
        for i = 1...K do
            v_n.add_neighbor(q.pop())
        q.push(v_n)                    /** add a new node **/
```

```
    S = q.size()
    for i = 1...S do                  /** connect to the root node **/
        v.add_neighbor(q.pop())
/** --------------------------- **/
/** time complexity O(|V| + |E|) **/
/** --------------------------- **/
```

- virtual split transformation: add a virtual layer (computation schedule, programming model) on the physical layer (physical layer, value propagation)
    - drawback of physically transforming:
        - extra time and space
        - slow down value propagation
    - virtualization: node mapping and edge mapping (a node mapping is sufficient)
        - virtual node array
        - dynamic mapping reasoning
        - edge-array coalescing (reorder the edges during the construction of CSR for GPU processing) ()
    - implicit value synchronization: the values of virtual nodes are all stored to the same memory location
    - correctness: push-based OK, pull-based vertex function need to be associative

Problem definition:

1. How to efficiently process graph data on GPU architecture?
2. How to achieve a good balance between irregularity reduction and convergence speed, while preserving the result correctness?

Precessed Work:

- programming method (CuSha, Gunrock): modify the graph programming abstraction -> hard to program
- thread method (warp segmentation, maximum warp): change low-level thread execution models -> not adaptive

Metrix gain:

datasets:

Pokec social, LiveJournal, Hollywood, Orkut, Sinaweibo, Twitter2010

graph analyses:

breath-first search (BFS), connected components (CC), single-source shortest path (SSSP), single-source widest path (SSWP), between centrality (BC), PageRank(PR)

performance matrix: execution time

result: Tigr-V+ achieves substantial performance improvements over the existing methods for most datasets and algorithms

Challenge:

correctness of degree-based analyses:

push-based: preserve indegree, but outdegree matters computation

pull-based: preserve outdegree, but indegree matters computation