

Parallelize Serial Graph Analytics With Data Centric Graph Primitives

Chenshan Yuan

Introduction

A lot of big data problem can be interpreted as graph problems. For large-scale graphs, the performance of the analytic tools is important. Gunrock is the state-of-art graph analytics library on GPU.[2] The goal of this project is to explore Gunrock library programmability and provide users an alternative high level interface in Python. NetworkX is a popular graph analytic library programmed in Python. [1] NetworkX provides a lot of graph primitives that are implemented ideal for CPU. The challenge is to parallelize serial graph algorithms in Gunrock framework.

Background

Gunrock has a data-centric framework. A subset of edges or vertices in a graph is represented as a frontier. Gunrock provides a frontier operator abstraction. Each graph analytic algorithm then can be interpreted as a iterative process of operation executed on a frontier. Given this data-centric framework, it's possible to translate or map serial and iterative graph algorithms into a parallel graph algorithms suited for GPU.

Approach

In Gunrock framework, there are three traversal operators: advance, filter and segmented intersection. There is also a compute operator that can be fused with one of the traversal operators. Each operator has a different functionality. An advance operator generates a new frontier from current frontier by visiting neighbors of current frontier. A filter operator generates a new frontier by choosing a subset from current frontier given programmer-specified condition. A compute operator defines an operations to be done on all elements in the input frontier. A segmented intersection takes two frontier and return number of intersections and intersected node IDS as new frontier.

Given Gunrock's operator level abstraction, it's critical to characterize a serial graph algorithm and map it to Gunrock operators. Shown in Figure 1 is a Bellman For d algorithm for solving SSSP problem in a graph. This piece of algorithm has

been broken down into sections of code which each section represents an operator in Gunrock framework. Shown in Figure 2 is Bellman Ford algorithm interpreted in Gunrock framework.

```
def _bellman_ford(G, source, weight, pred=None, paths=None, dist=None,
                 cutoff=None, target=None):
    """Relaxation loop for Bellman-Ford algorithm"""
    if pred is None:
        pred = {v: [None] for v in source}

    if dist is None:
        dist = {v: 0 for v in source}

    G_succ = G.succ if G.is_directed() else G.adj
    inf = float('inf')
    n = len(G)

    count = {}
    q = deque(source)
    in_q = set(source)
    while q:
        u = q.popleft()
        in_q.remove(u)

        # Skip relaxations if any of the predecessors of u is in the queue.
        if all(pred_u not in in_q for pred_u in pred[u]):
            dist_u = dist[u]
            for v, e in G_succ[u].items():
                dist_v = dist_u + weight(v, u, e)

                if cutoff is not None:
                    if dist_v > cutoff:
                        continue

                if target is not None:
                    if dist_v > dist.get(target, inf):
                        continue

                if dist_v < dist.get(v, inf):
                    if v not in in_q:
                        q.append(v)
                        in_q.add(v)
                        count_v = count.get(v, 0) + 1
                        if count_v == n:
                            raise nx.NetworkXUnbounded(
                                "Negative cost cycle detected.")
                        count[v] = count_v
                    dist[v] = dist_v
                    pred[v] = [u]

                elif dist.get(v) is not None and dist_v == dist.get(v):
                    pred[v].append(u)

    if paths is not None:
        dsts = [target] if target is not None else pred
        for dst in dsts:
            path = [dst]
            cur = dst

            while pred[cur][0] is not None:
                cur = pred[cur][0]
                path.append(cur)

            path.reverse()
            paths[dst] = path

    return dist
```

Figure 1: SSSP, Bellman Ford.

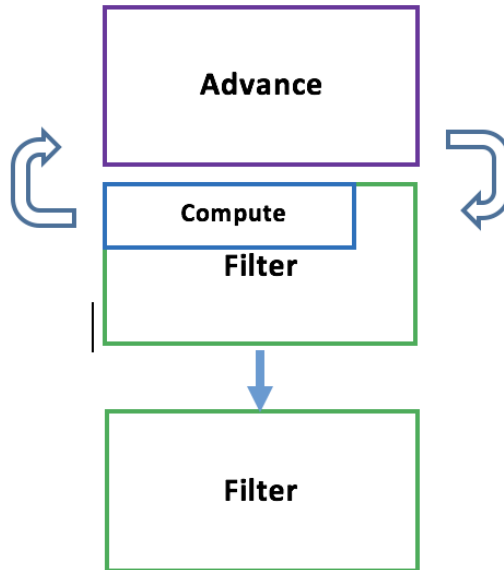


Figure 2: SSSP, Bellman Ford in Gunrock framework.

References

- [1] A. A. Hagberg, D. A. Schult, and P. J. Swart. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy2008)*, pages 11–15, Pasadena, CA USA, Aug. 2008.
- [2] Y. Wang, Y. Pan, A. A. Davidson, Y. Wu, C. Yang, L. Wang, M. Osama, C. Yuan, W. Liu, A. T. Riffel, and J. D. Owens. Gunrock: GPU graph analytics. *CoRR*, abs/1701.01170, 2017.