Assignment 8

Sahand Sabour - 山姆 - 2022380024

Introduction

In this assignment, we were tasked to implement two graph processing algorithms using the Gridgraph framework. We chose the delta version of the PageRank algorithm and the conductance algorithm. In this report, we provide detailed information about our implementation and demonstrate the obtained results. To add and compile our implementations, we put our files in the examples folder. Accordingly, we modify and add the following lines to the Makefile of GridGraph.

```
TARGETS= bin/preprocess  bin/make_undirect bin/t2b bin/add_weight bin/bfs
bin/wcc bin/pagerank bin/spmv bin/mis bin/radii bin/conductance
bin/pagerank_delta
bin/conductance: examples/conductance.cpp $(HEADERS)
  $(CXX) $(CXXFLAGS) -o $@ $< $(SYSLIBS)
bin/pagerank_delta: examples/pagerank_delta.cpp $(HEADERS)
  $(CXX) $(CXXFLAGS) -o $@ $< $(SYSLIBS)
```

PageRank-Delta

This algorithm is very similar to the original PageRank implementation. Therefore, we leverage the pagerank implementation from Gridgraph as the foundation of our implementation. Accordingly, we declare a new BigVector **delta** to store delta values for each vertice and create a new commandline argument to obtain the propagation threshold from the user. Similarly, we stream the vertices and initialize the delta value for all vertices to 1 and their rank to 1/number of vertices. .

```
float threshold = atof(argv[4]); // propagation threshold
int num_vertices = graph.vertices; // number of vertices
BigVector<float> delta(graph.path + "/delta", num_vertices); // delta vector

graph.stream_vertices<VertexId>(
  [&](VertexId i)
  {
    pagerank[i] = 1.f / (float)num_vertices; // initial rank based on Ligra
    sum[i] = 0; // initial sum
    delta[i] = 1.f; // initial delta
    return 0;
```

```
  },
  nullptr, 0,
  [&](std::pair<VertexId, VertexId> vid_range)
  {
    pagerank.load(vid_range.first, vid_range.second);
    sum.load(vid_range.first, vid_range.second);
    delta.load(vid_range.first, vid_range.second);
  },
  [&](std::pair<VertexId, VertexId> vid_range)
  {
    pagerank.save();
    sum.save();
    delta.save();
  });
```

In this algorithm, for a vertice A, we have

$$Delta(A) = 0.85 * (\frac{Delta(B)}{L(B)} + \frac{Delta(C)}{L(C)} + \dots)$$

$$Rank(A) = Rank(A) + Delta(A)$$

, where delta is only added if delta/rank is larger than the threshold. Therefore, we first stream the edges to calculate the sums of delta/degree for each vertice V (i.e., delta(V)) .

```
graph.hint(pagerank, delta);
graph.stream_edges<VertexId>(
  [&](Edge &e)
  {
    // Calculate delta for edge
    write_add(&sum[e.target], delta[e.source] / degree[e.source]);
    return 0;
  },
  nullptr, 0, 1,
  [&](std::pair<VertexId, VertexId> source_vid_range)
  {
    delta.lock(source_vid_range.first, source_vid_range.second);
  },
  [&](std::pair<VertexId, VertexId> source_vid_range)
  {
    delta.unlock(source_vid_range.first, source_vid_range.second);
  });
```

Accordingly, we stream the vertices and only add the delta to the vertice's rank if the

mentioned ratio (i.e., delta/rank) exceeds the threshold.

```
graph.hint(pagerank, sum, delta);
graph.stream_vertices<float>(
  [&](VertexId i)
  {
    delta[i] = 0.85f * sum[i];
    if ((delta[i]/pagerank[i])> threshold){
      pagerank[i] += delta[i];
    }
    sum[i] = 0;
    return 0;
  },
  nullptr, 0,
  [&](std::pair<VertexId, VertexId> vid_range)
  {
    pagerank.load(vid_range.first, vid_range.second);
    sum.load(vid_range.first, vid_range.second);
    delta.load(vid_range.first, vid_range.second);
  },
  [&](std::pair<VertexId, VertexId> vid_range)
  {
    pagerank.save();
    sum.save();
    delta.save();
  });
```

Lastly, we iterate through the vertices to find the vertice with the highest rank.

```
// Find vertice with maximum rank
float max_val = -1;
int max_idx = -1;
for (int i = 0; i < num_vertices; i++)
{
  if (pagerank[i] > max_val)
  {
    max_val = pagerank[i];
    max_idx = i;
  }
}
```

Conductance

This algorithm is fairly trivial. We stream over the edges and increment three counters based on the following logic: if an edge's source and target vertices belong to different categories, it is considered as a crossover edge; if the edge's source or target vertice & 1 != 0, then it belongs to the red category; else, it belongs to the black category.

```
int black_num = 0;
int red_num = 0;
int crossover_num = 0;

graph.stream_edges<VertexId>(
  [&](Edge &e)
  {
    // Crossover edge
    if ((e.source & 1) != (e.target & 1))
    {
      write_add(&crossover_num, 1);
    }
    // Red edge
    else if (e.source & 1)
    {
      write_add(&red_num, 1);
    }
    // Black edge
    else
    {
      write_add(&black_num, 1);
    }
    return 0;
  },
  nullptr, 0, 0);
```

Lastly, conductance is calculated as follows

```
float conductance = (float)crossover_num / (float)min(red_num, black_num);
```

Guide

For running the program, simply compile and run the program as follows:

```
# head to GridGraph Directory (we use the provided repo)
cd hw9_GridGraph
# Compile
make
# Preprocess the livejournal dataset
./bin/preprocess -i /data/hw9_data/livejournal -o livejournal_grid -v 4847571 -
p 4 -t 0
# Run Page-Rank Delta
./bin/pagerank_delta [data_path] [num_iterations] [memory budget] [threshold]
# Run Conductance
./bin/conductance [data_path] [memory budget]
```

For instance, by running the following commands, we get the corresponding results:

```
> ./bin/pagerank_delta livejournal_grid 20 8 0.05
degree calculation used 0.30 seconds
20 iterations of pagerank took 11.22 seconds
Highest Rank Vertice - Index 8737 - Value 670.420044

> ./bin/conductance livejournal_grid 8
Conductance is calculated as 2.01 - Process took 6.34 seconds
Crossover: 34486389 - #Reds: 17313478 - #Blacks: 17193906
```