Performance comparison of GraphChi, GridGraph and Mosaic

by

Farzana Yesmin
B.Sc, Ahsanullah University of Science and Technology, 2011

A Project Submitted in Partial Fulfillment
of the Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

Supervisory Committee

Performance Comparison of GraphChi, GridGraph and Mosaic

by

Farzana Yesmin
B.Sc, Ahsanullah University of Science and Technology, 2011

Supervisory Committee

Dr. Alex Thomo, Supervisor
Department of Computer Science

Dr. Venkatesh Srinivasan, Departmental Member
Department of Computer Science

Abstract

Supervisory Committee

Dr. Alex Thomo, Supervisor
Department of Computer Science

Dr. Venkatesh Srinivasan, Departmental Member
Department of Computer Science

In this report, we have presented a performance comparison based on run time among three graph processing frameworks. The frameworks we have chosen for performance analysis, are: GraphChi, GridGraph and Mosaic. All these frameworks are vertex-centric computation model which can process large scale graph on a single machine. Each of these framework provides many computational application programs. For performing comparison among these frameworks, we have chosen two application programs: page rank and connected components, which are common in all these three frameworks.

We have shown performance comparison based on run time, which represents the time that page rank and connected component take to run on several input graphs. We have considered a total of thirteen input graphs ranged from small to large.

Table of Contents

List of Tables

List of Figures

Acknowledgments

I would like to thank Dr. Alex Thomo for mentoring and encouraging throughout the project.

Dedication

I dedicate this project report to my supervisor Dr. Alex Thomo who have always supported me while doing this project.

Chapter 1

1.1 Introduction:

The importance of processing and analyzing graphs have drawn interests in recent years. A graph is a connection of vertices and edges, where vertices represents entities or people and edges represent connections between vertices. Almost everything can be represented as a graph. Many real-world problems, for example web graphs, online social networks, user-item matrices, and many more, can be represented as graph computing problems (Zhu, Han, & Chen, 2015). For making the analysis of graphs easier, many computational programs can be applied on graph. Therefore, analyzing and applying many computational programs on graph have become important research.

In this project work, we have planned to do a performance comparison among three big graph processing frameworks: GraphChi, GridGraph and Mosaic. All these systems can process large graphs on a single machine and are vertex-centric computation model where user defines a program and the program is executed locally for each vertex in parallel (Kyrola, Blelloch, & Guestrin).

Various programs are provided by each of these frameworks. But we have chosen only two of these application programs since we are comparing performance of those frameworks which have these two programs in common. One is ‘page rank’ which produces a ranking for each of the vertices in a graph and the other program is ‘connected component’ that finds all subgraphs in which two or more vertices are connected to each other by paths, and are not connected to additional vertices in the supergraph.

In this work, we have therefore considered ‘run time’, the time taken to run various input graphs by both page rank, and ‘connected component’ programs.

1.2 Summary of chapters:

This section provides an overview of the report and the contents of each chapter are summarized as follows:

In chapter 2, we have listed all our test input datasets in a table. We have mentioned the name, number of vertices, number of edges and degree of maximum of each of the input graph.

Chapter 3 provides a background study about three graph processing systems: GraphchChi, GridGraph and Mosaic. It gives a summary about those three systems and the methodologies behind these frameworks. All commands and instructions are described to build and run both ‘page rank’ and ‘connected component’ programs for each of those frameworks.

In chapter 4, we have presented all our experiment data in tables. We also show an example of the command for both ‘page rank’ and ‘connected component’ programs that we ran for one of our test input graphs for GraphChi, GridGraph and Mosaic.

In Chapter 5, we have presented some result charts based on our experiment results. We discussed which framework is faster based on run time according to our results and future work.

Chapter 6 provides a conclusion for the overall work we performed.

Chapter 2

2.1 Datasets overview:

We have performed our experiment using 13 graph datasets. All datasets characteristics are given in the following table (Khaouid, Barsky, Srinivasan, & Thomo, 2015). In the dataset table, we show the name of these datasets, abbreviation, number of vertices of each dataset, number of edges and maximum degree. We can divide the datasets in small, medium and large group according to the number of edges.

First five datasets: Astro Physics (ca-astroph); Condensed Matter (ca-condmat); Gnutella P2P network (p2p-gnutella31); Slashdot 1 (soc-sign-Slashdot-090221); Slashdot 2 (soc-Slashdot0902) belong to small group as these datasets have less than 1 million edges.

Next six datasets: Amazon product co-purchasing network (amazon0601); Berkeley-Stanford web graph (web-BerkStan); Texas road network (roadNet-TX); California road network (roadNet-CA); California road network (roadNet-CA); Wikipedia Talk network (wiki-Talk); LiveJournal network (LiveJournal) are in medium group because these datasets have number of edges in the range 2.4 – 43.1 million.

The last two datasets: UK 2005 web crawl (uk-2005) and Twitter 2010 followers network (twitter-2010) belong to large group since they have 790 million and 2405 million edges respectively.

Frist eleven graph are taken from http://snap.stanford.edu/data/index.html (Khaouid, Barsky, Srinivasan, & Thomo, 2015).

The last two are obtained from http://law.di.unimi.it/webdata (Khaouid, Barsky, Srinivasan, & Thomo, 2015)

Name	**Abbreviation**	**#Vertices**	**#Edges**	**d_{max}**
Astro Physics (ca-astroph)	AP	18.7 K	198.1 K	504
Condensed Matter (ca-condmat)	CM	23.1 K	93.5 K	280
Gnutella P2P network (p2p-gnutella31)	GN	62.6 K	147.9 K	95
Slashdot 1 (soc-sign-Slashdot-090221)	S1	82.1 K	500.5 K	2,548
Slashdot 2 (soc-Slashdot0902)	S2	82.2 K	543.4 K	2,553
Amazon product co-purchasing network (amazon0601)	AM	0.4 M	2.4 M	2,752
Berkeley-Stanford web graph (web-BerkStan)	BS	0.7 M	6.6 M	84,230
Texas road network (roadNet-TX)	TX	1.4 M	1.9 M	12
California road network (roadNet-CA)	CA	2.0 M	2.8 M	12
Wikipedia Talk network (wiki-Talk)	WT	2.4 M	4.7 M	100,029
LiveJournal network (LiveJournal)	LJ	4.8 M	43.1 M	20,334
UK 2005 web crawl (uk-2005)	UK	39.5 M	790 M	1,776,858
Twitter 2010 followers network (twitter-2010)	TW	41.7 M	2,405 M	2,997,487

Table 2.1: All datasets name, abbreviation, number of vertices, number of edges, maximum degree (Khaouid, Barsky, Srinivasan, & Thomo, 2015).

2.2 Test setup:

We performed all our experiment on a machine with Intel i7-2600 processor, 3.40ghz cpu and 12GB RAM, running Ubuntu 16.04.4 (linux) and the hard disk is 1TB ST31000524AS 7200 rpm.

Chapter 3

3.1 GraphChi:

GraphChi, a disk-based system, can solve a wide variety of computational problems on large graph with billions of edges efficiently on a single machine (Kyrola, Blelloch, & Guestrin). It uses a novel algorithm for processing the input graph from SSD or hard drive. GraphChi is a vertex-centric model, proposed by GraphLab and Google's Pregel. In order to process large graph from disk, a method called Parallel Sliding Windows (PSW) (Kyrola, Blelloch, & Guestrin) has been implemented in GraphChi. Vertex-centric programs in GraphChi runs asynchronously which means all changes are written to edges and exposed immediately to subsequent computation (GraphChi/graphchi-cpp, n.d.).

Implementation of Parallel Sliding Windows (PSW) is more efficient than synchronous computation for many purposes because PSW automatically implements the asynchronous model of computation (Bertsekas & Tsitsiklis, 1989), (Low, et al., 2010).

GraphChi also has two important features: streaming graph updates and removal of edges from the graph (GraphChi/graphchi-cpp, n.d.). While simultaneously performing computation, GraphChi can process over one hundred thousand graph updates per second (Kyrola, Blelloch, & Guestrin).

No installation and no configuration are needed for running example programs of GraphChi. The input graphs are automatically converted to GraphChi's internal shards (GraphChi/graphchi-cpp, n.d.). So all example programs just need to compile and run. The system will detect if preprocessing is needed for the input graph (GraphChi/graphchi-cpp, n.d.). For our experiment, we have selected two example programs of GraphChi: page rank and connected component. In next section, we have described the commands (GraphChi/graphchi-cpp, n.d.) to build and run both page rank and connected component.

3.1.1 Building and running page rank of GraphChi:

Since preprocessing is done automatically by the application of GraphChi, we need the commands and give proper argument to build and run the application program.

To build the page rank application program, the following command (GraphChi/graphchi-cpp, n.d.) needs to be run:

```
make example_apps/pagerank
```

The above command needs to be run only for once for all input files.

After building page rank application program, the following command (GraphChi/graphchi-cpp, n.d.) is run:

```
bin/example_apps/pagerank file GRAPH-NAME niters 5
```

For the run command, two parameters need to be specified: file and niters. For file parameter, instead of GRAPH-NAME, we have to provide input graph path. User can provide number of iterations by giving a value for niters parameter. The output of this program is the ids of the top 20 vertices with highest page rank.

3.1.2 Building and running connected component of GraphChi:

Similar to the page rank program, connected component converts the input graph to GraphChi's internal shards (GraphChi/graphchi-cpp, n.d.)**.** So this program is just need to compile and run. To build the connected component program of GraphChi, the following command (GraphChi/graphchi-cpp, n.d.) needs to run:

```
make example_apps/connectedcomponents
```

Then connected component program can be run by the following command (GraphChi/graphchi-cpp, n.d.):

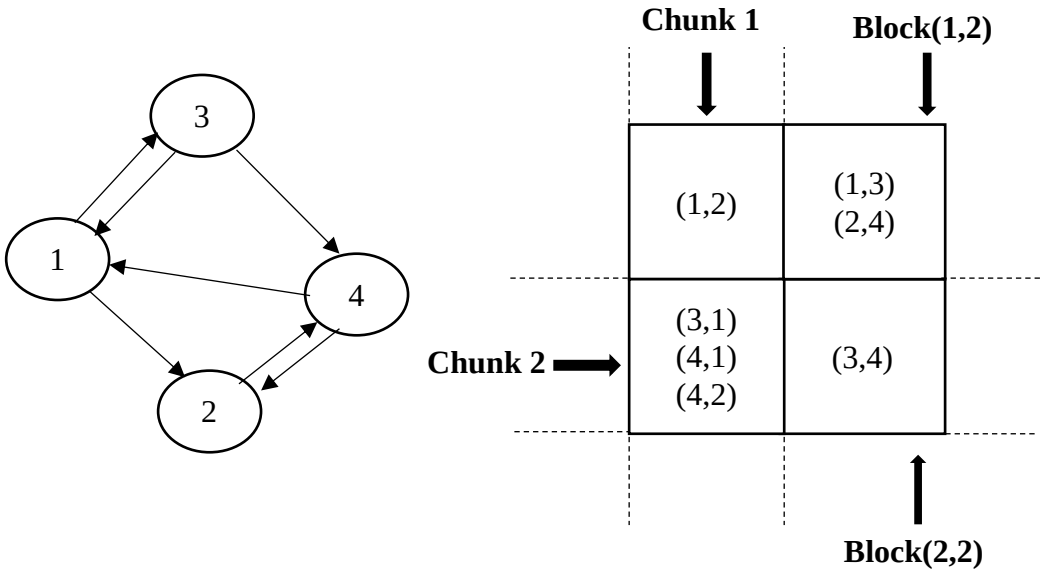
```
bin/example_apps/connectedcomponents file GRAPH-NAME
```

where, input graph needs to be given in place of GRAPH-NAME for file parameter. The output of this program is component id and number of vertices**.**

3.2 GridGraph:

GridGraph processes large-scale graph on a single machine (Zhu, Han, & Chen, 2015). Two level partitioning: a first fine-grained which is applied to break graphs into 1D-partitioned vertex chunks and 2D-partitioned edge blocks in preprocessing and a second level partitioning is coarsegrained which is applied in runtime (Zhu, Han, & Chen, 2015). GridGraph uses a novel dual sliding windows method in order to reduce the I/O amount required for computation by streaming edges and applying on-the-fly vertex updates (Zhu, Han, & Chen, 2015). GridGraph also offers selective scheduling that reduce streaming on unnecessary edges and the uses of selective scheduling improves performance for many iterative algorithms (Zhu, Han, & Chen, 2015).

GridGraph groups edges into a grid format, which can be utilized in all algorithms running on the same graph, in preprocessing and it takes very short time for preprocessing to complete (Zhu, Han, & Chen, 2015). We have taken an example graph to show how this example graph is partitioned into gird format. Figure 3.1 illustrates how the example graph in Figure (a) is represented into grid Figure (b). We choose P = 2 in this example. There are total 4 vertices in this graph and 2 vertex chunks: {1, 2} and {3, 4}. Edge (1, 2) is partitioned to Block (1, 1) since Vertex 1 belongs to Chunk 1 and Vertex 2 also belongs to Chunk 1. So edge (2, 4) and (1, 3) both belong to block (1, 2). Similarly, we can have all other edges put into the block where these edges belong.



(a): example graph (b): grid format

Figure 3.1: Grid representation of edge blocks

Global information of the represented graph, including the number of vertices and number of edges, the type of edge: weighted or unweighted and the partition count are contained in a metadata file which is created by GridGraph (Zhu, Han, & Chen, 2015).

3.2.1 Preprocessing of input graph:

Before running any program of GridGraph, the input graph needs to be preprocessed where GridGraph partitions the original edge list into the grid format (thu-pacman/GridGraph, n.d.).

Two types of edge list files are supported (thu-pacman/GridGraph, n.d.):

- Unweighted. Edges are tuples of <4 byte source, 4 byte destination>.
- Weighted. Edges are tuples of <4 byte source, 4 byte destination, 4 byte float typed weight>.

To partition the edge list (thu-pacman/GridGraph, n.d.):

```
./bin/preprocess -i [input path] -o [output path] -v [vertices] -p [partitions] -t [edge type: 0=unweighted, 1=weighted]
```

Where [input path] is the path of input file and [output path] is the path where the preprocessed grid format data would be saved. [vertices] is total number of vertices of input graph. [partitions] is how the unweighted graph would be partitioned into a grid and the last argument [edge type] is whether the edges are weighted or unweighted. So either 0 or 1 would be the value of the last parameter.

This preprocess needs to be done once for each input graph. Once the preprocess is done for an input graph, then any program can be run for that input graph.

3.2.2 Running page rank of GridGraph:

After preprocessing an input graph, page rank program can be run by the following command (thu-pacman/GridGraph, n.d.).

```
./bin/pagerank [path] [number of iterations] [memory budget]
```

Where [path] is the path of the grid format and [number of iterations] is total number of iterations we want to run. Another parameter is [memory budget] and it is the memory budget in GB.

3.2.3 Running connected component of GridGraph:

If the preprocess is done for the input graph, then we can execute the run command (thu-pacman/GridGraph, n.d.) for connected component.

```
./bin/wcc [path] [memory budget]
```

The run command needs two parameters: [path] which is the path of grid format and another parameter is [memory budget] for which it needs the memory budget in GB.

3.3 Mosaic:

Mosaic is an engine that can process graph with trillion on a single heterogeneous machine (Maass, et al., 2017). In Mosaic, a new data structure called Hilbert-ordered tiles has been designed for locality, load balancing and compression which produces a compact representation of the graph (Maass, et al., 2017). Both vertex-centric operations (on host processors) and edge-centric operations (on coprocessors) are executed efficiently by a hybrid computation and execution model proposed by Mosaic (Maass, et al., 2017).
Mosaic breaks a graph into disjoint sets of edges called tiles and each of the tiles represents a subgraph of the input graph (Maass, et al., 2017).
There are some steps to get the input graph ready for running any application program of Mosaic. We will be describing those steps based on our experiment in the following section.
We need to build Mosaic at the very first step. To build Mosaic (sslab-gatech/mosaic, n.d.), we first need to change the directory to scr and the command is:

```
$ cd src
```

Then we need to run the following two commands (sslab-gatech/mosaic, n.d.):

```
$ make cmake
$ make
```

Running these three commands will build the binaries of Mosaic, in a release and debug configuration into the build-folder (sslab-gatech/mosaic, n.d.).

3.3.1 Preparation to make the input graph runnable:

The input graph is needed to be included in the configuration of Mosaic and it has to be added in the configuration files in config/default.py (sslab-gatech/mosaic, n.d.). This is done by adding the appropriate settings: the variables SG_INPUT_WEIGHTED,

SG_INPUT_FILE and SG_GRAPH_SETTINGS_DELIM following a similar pattern as the other example graphs mentioned in the default.py (sslab-gatech/mosaic, n.d.).

For faster generation, a script is provided in src/tools/scripts/convert_graph.py to convert the txt-based input graph to a binary format and the following commands (sslab-gatech/mosaic, n.d.) need to be executed for that purpose.

```
$ cd src/tools/scripts
$ ./convert_graph.py --input /data/graph/input/twitter-small/twitter_rv_small.net --output /data/graph/input/twitter- small.net
```

By running another script in src/tools/scripts/generate_graph.py, Mosaic-internal format of a graph can be generated.

Examples of this conversion (sslab-gatech/mosaic, n.d.):

```
$ cd src/tools/scripts
$ ./generate_graph.py --dataset twitter-small --binary --in-memory
```

In the above command, name of the example dataset is “twitter-small” which has been given in config/default.py for the corresponding input graph.

Next step is to run rebalance script in in src/tools/scripts/rebalance_input.py and running rebalance_input.py script enables Mosaic to skip the computation for tiles without active vertices and edges (sslab-gatech/mosaic, n.d.). Commands (sslab-gatech/mosaic, n.d.) are as follows:

```
$ cd src/tools/scripts
$ ./rebalance_input.py --dataset twitter-small
```

In order to build the index structure needed for optimization which is a very useful optimization for Connected Components and BFS, we need to run the following commands (sslab-gatech/mosaic, n.d.):

```
$ cd src/tools/scripts
$ ./run_tiles_indexer.py --dataset twitter-small
```

3.3.2 Running page rank of Mosaic:

Now programs of mosaic are ready to run. For example, executing page rank needs to run the following command (sslab-gatech/mosaic, n.d.):

```
cd src/tools/scripts
./run_mosaic.py --dataset datasetName --algorithm pagerank --max-iteration 5
```

Here, for the parameter "dataset", the name, which has been given in default.py for the input graph file, needs to be given. Next parameter is the name of the program that we want to run. Finally, the third parameter is the total number of iteration.

3.3.3 Running connected component of Mosaic:

Similarly, command (sslab-gatech/mosaic, n.d.) to run connected component program:

```
$ cd src/tools/scripts
$ ./run_mosaic.py --dataset datasetName --algorithm cc --max-iteration 5
```

All parameters are same as the command for running pageRank of Mosaic, except the name of the algorithm.

Chapter 4

In this chapter, we have represented all our results in tabular format. Our target was to run both page rank and connected component for all thirteen datasets for five iterations. For some input graphs: Slashdot 1 (soc-sign-Slashdot-090221), Slashdot 2 (soc-Slashdot0902), UK 2005 web crawl (uk-2005) and Twitter 2010 followers network (twitter-2010), we got error while running for five iterations. So we ran those input graphs only for one iteration for GraphChi and GridGraph. We also gave one example of command to run one of the input graphs to show how exactly we ran page rank and connected component for GraphChi, GridGraph and Mosaic.

4.1 Results for page rank on GraphChi:

We ran the commands to build and run page rank for the first input graph as follows:

```
make example_apps/pagerank
time ./bin/example_apps/pagerank file ./bin/example_apps/data/data1/data1_astrocnet-undirected.txt niters 5
```

Input graph	Run time (s)
Astro Physics (ca-astroph)	7.254s
Condensed Matter (ca-condmat)	4.374s
Gnutella P2P network (p2p-gnutella31)	4.822s
Slashdot 1 (soc-sign-Slashdot-090221)	7.869s
Slashdot 2 (soc-Slashdot0902)	5.654s
Amazon product co-purchasing network (amazon0601)	6.153s
Berkeley-Stanford web graph (web-BerkStan)	9.094s
Texas road network (roadNet-TX)	7.221s
California road network (roadNet-CA)	23.375s
Wikipedia Talk network (wiki-Talk)	5.496s

LiveJournal network (LiveJournal)	5.779s
UK 2005 web crawl (uk-2005)	602.745s
Twitter 2010 followers network (twitter-2010)	1114.979s

Table 4.1: Run time of page rank on GraphChi for five iterations

Input graph	**Run time (s)**
Slashdot 1 (soc-sign-Slashdot-090221)	7.956s
Slashdot 2 (soc-Slashdot0902)	6.032s
UK 2005 web crawl (uk-2005)	503.13s
Twitter 2010 followers network (twitter-2010)	835.741s

Table 4.2: Run time of page rank on GraphChi for one iteration

4.2 Result of connected component on GraphChi:

The following commands were run to build and run connected component for the first input graph as follows:

```
make example_apps/connectedcomponent
time ./bin/example_apps/connectedcomponent file ./bin/example_apps/data/data1/data1_astrocnet-undirected.txt niters 5
```

Input graph	**Run time (s)**
Astro Physics (ca-astroph)	11.117s
Condensed Matter (ca-condmat)	8.456s
Gnutella P2P network (p2p-gnutella31)	7.056s
Slashdot 1 (soc-sign-Slashdot-090221)	5.889s
Slashdot 2 (soc-Slashdot0902)	5.725s
Amazon product co-purchasing network (amazon0601)	7.072s

Berkeley-Stanford web graph (web-BerkStan)	32.136s
Texas road network (roadNet-TX)	8.675s
California road network (roadNet-CA)	38.032s
Wikipedia Talk network (wiki-Talk)	28.040s
LiveJournal network (LiveJournal)	25.365s
UK 2005 web crawl (uk-2005)	871.034s
Twitter 2010 followers network (twitter-2010)	Memory exceed error

Table 4.3: Run time of connected component on GraphChi

4.3 Results for page rank on GridGraph:

All input graphs were needed to be converted from the text format into the binary format. We used the following program (thu-pacman/GridGraph, n.d.) for txt to binary conversion and we gave this program name as “firstPreprocessing”.

```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
int main(int argc, char ** argv) {
    FILE * fin = fopen(argv[1], "r");
    FILE * fout = fopen(argv[2], "wb");
    char * line = new char [1024];
    size_t line_length;
    ssize_t read_length;
    int max_vid = 0;
    while ((read_length = getline(&line, &line_length, fin)) != -1) {
        if (line[0]=='#') continue;
        int src, dst;
        assert(sscanf(line, "%d %d", &src, &dst)==2);
        fwrite(&src, sizeof(int), 1, fout);
        fwrite(&dst, sizeof(int), 1, fout);
        if (src > max_vid) max_vid = src;
        if (dst > max_vid) max_vid = dst;
    }
    fclose(fin);
    fclose(fout);
    printf("|V|=%d\n", max_vid + 1);
}
```

At first, we converted txt-format input graph to binary format by running the following command:

```
./firstPreprocessing ../data/data1_astrocnet-undirected.txt data/data1/d1.txt
```

Then the preprocessing program was run to generate grid format from the binary file as follows:

```
./bin/preprocess -i ./data/data1/d1.txt -o ./data/dataP1 -v 133280 -p 4 -t 0
```

Finally, we executed the command to run page rank for the first input graph by giving the grid format name as parameter:

```
./bin/pagerank  ./data/dataP1  5  8
```

Input graph	**Run time (s)**
Astro Physics (ca-astroph)	0.04 s
Condensed Matter (ca-condmat)	0.04 s
Gnutella P2P network (p2p-gnutella31)	0.04 s
Slashdot 1 (soc-sign-Slashdot-090221)	0.06 s
Slashdot 2 (soc-Slashdot0902)	0.06 s
Amazon product co-purchasing network (amazon0601)	0.11 s
Berkeley-Stanford web graph (web-BerkStan)	0.25 s
Texas road network (roadNet-TX)	0.29 s
California road network (roadNet-CA)	1.26 s
Wikipedia Talk network (wiki-Talk)	0.12 s
LiveJournal network (LiveJournal)	0.16 s
UK 2005 web crawl (uk-2005)	319.58 s
Twitter 2010 followers network (twitter-2010)	482.27 s

Table 4.4: Run time of page rank on GridGraph for five iterations

Input graph	Run time (s)
Slashdot 1 (soc-sign-Slashdot-090221)	0.04 s
Slashdot 2 (soc-Slashdot0902)	0.03 s
UK 2005 web crawl (uk-2005)	82.96 s
Twitter 2010 followers network (twitter-2010)	168.41 s

Table 4.5: Run time of page rank on GridGraph for one iteration

4.4 Result of connected component on GridGraph:

Once we created the binary format from txt format and grid format from binary format for a particular input graph, we can use this grid format to run connected component program for that specific input graph. So the preprocess steps are same. Then we ran the connected component program by the following command:

```
./bin/wcc ./data/dataP1  8
```

Input graph	Run time (s)
Astro Physics (ca-astroph)	0.06 s
Condensed Matter (ca-condmat)	0.06 s
Gnutella P2P network (p2p-gnutella31)	0.04 s
Slashdot 1 (soc-sign-Slashdot-090221)	0.10 s
Slashdot 2 (soc-Slashdot0902)	0.09 s
Amazon product co-purchasing network (amazon0601)	0.31 s
Berkeley-Stanford web graph (web-BerkStan)	0.71 s
Texas road network (roadNet-TX)	0.12 s

California road network (roadNet-CA)	3.34 s
Wikipedia Talk network (wiki-Talk)	0.47 s
LiveJournal network (LiveJournal)	0.41 s
UK 2005 web crawl (uk-2005)	43.30 s
Twitter 2010 followers network (twitter-2010)	242.30 s

Table 4.6: Run time of connected component on GridGraph

4.5 Result of page rank on Mosaic:

First we added all input graphs in config/default.py where we made changes in variables: SG_INPUT_WEIGHTED, SG_INPUT_FILE and SG_GRAPH_SETTINGS_DELIM. We have given some screen shots of those changes made in default.py.

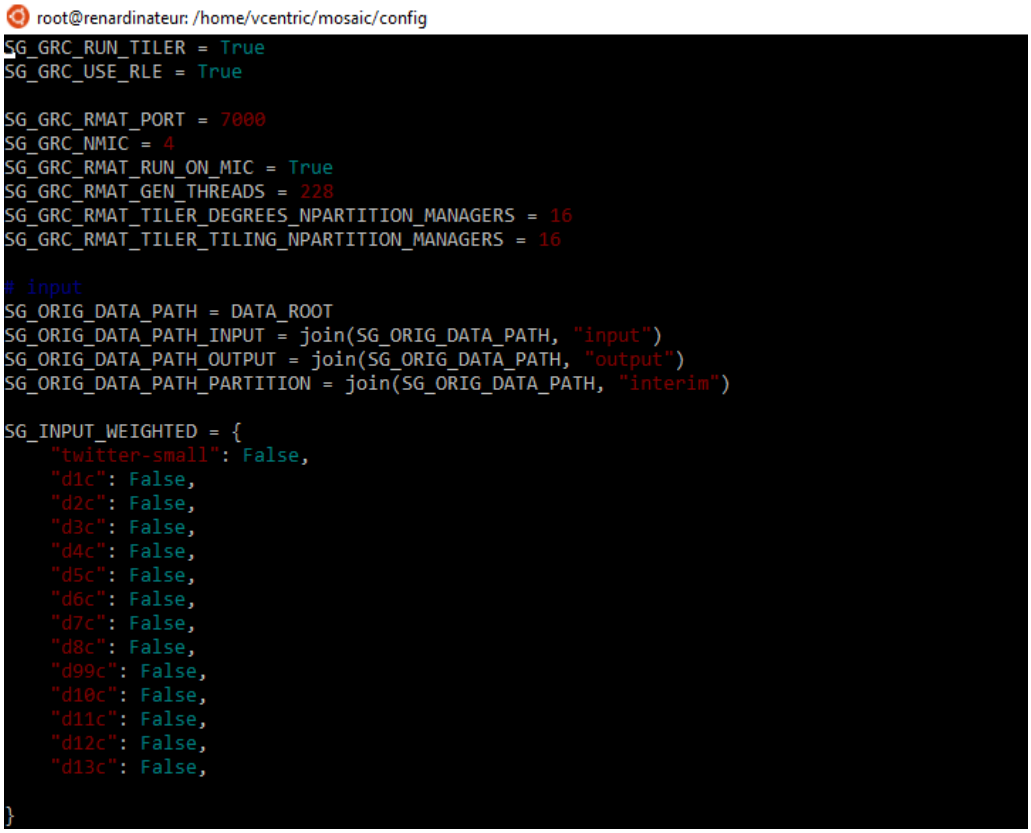


Figure 4.1: Changes in SG_INPUT_WEIGHTED variable in default.py

root@renardinateur: /home/vcentric/mosaic/config

```
}
SG_INPUT_FILE = {
    "twitter-small": {
        "delim": join(DATA_ROOT, "twitter-small/twitter_rv_small.net"),
        "binary": join(DATA_ROOT, "twitter-small/twitter_rv_small.bin"),
    },
    "d10c":{
        "delim": join(DATA_ROOT, "data10/d10.txt"),
        "binary": join(DATA_ROOT, "data10/d10.bin"),
    },
    "d1c":{
        "delim": join(DATA_ROOT, "data1/d1.txt"),
        "binary": join(DATA_ROOT, "data1/d1.bin"),
    },
    "d2c":{
        "delim": join(DATA_ROOT, "data2/d2.txt"),
        "binary": join(DATA_ROOT, "data2/d2.bin"),
    },
    "d3c":{
        "delim": join(DATA_ROOT, "data3/d3.txt"),
        "binary": join(DATA_ROOT, "data3/d3.bin"),
    },
    "d4c":{
        "delim": join(DATA_ROOT, "data4/d4.txt"),
        "binary": join(DATA_ROOT, "data4/d4.bin"),
    },
    "d5c":{
        "delim": join(DATA_ROOT, "data5/d5.txt"),
        "binary": join(DATA_ROOT, "data5/d5.bin"),
    },
    "d6c":{
        "delim": join(DATA_ROOT, "data6/d6.txt"),
        "binary": join(DATA_ROOT, "data6/d6.bin"),
    },
    "d7c":{
        "delim": join(DATA_ROOT, "data7/d7.txt"),
        "binary": join(DATA_ROOT, "data7/d7.bin"),
    },
    "d8c":{
        "delim": join(DATA_ROOT, "data8/d8.txt"),
        "binary": join(DATA_ROOT, "data8/d8.bin")
```

Figure 4.2: Changes in SG_INPUT_FILE variable in default.py

root@renardinateur: /home/vcentric/mosaic/config

```
    "sssp": True,
    "bp": False,
    "tc": False
}

SG_DATASET_DISABLE_SELECTIVE_SCHEDULING = [
    "rmat-32"
]

SG_DELIM_TAB = "tab"
SG_DELIM_COMMA = "comma"
SG_DELIM_SPACE = "space"
SG_DELIM_SEMICOLON = "semicolon"

SG_GRAPH_SETTINGS_DELIM = {
    "d10c": {"count_vertices": 1393383, "delimiter": SG_DELIM_TAB, "use_original_ids": False},
    "d1c": {"count_vertices": 133280, "delimiter": SG_DELIM_TAB, "use_original_ids": False},
    "d2c": {"count_vertices": 108300, "delimiter": SG_DELIM_TAB, "use_original_ids": False},
    "d3c": {"count_vertices": 62586, "delimiter": SG_DELIM_TAB, "use_original_ids": False},
    "d4c": {"count_vertices": 82144, "delimiter": SG_DELIM_TAB, "use_original_ids": False},
    "d5c": {"count_vertices": 82168, "delimiter": SG_DELIM_TAB, "use_original_ids": False},
    "d6c": {"count_vertices": 403394, "delimiter": SG_DELIM_TAB, "use_original_ids": False},
    "d7c": {"count_vertices": 685231, "delimiter": SG_DELIM_TAB, "use_original_ids": False},
    "d8c": {"count_vertices": 2394385, "delimiter": SG_DELIM_TAB, "use_original_ids": False},
    "d99c": {"count_vertices": 4847571, "delimiter": SG_DELIM_TAB, "use_original_ids": False},
    "d11c": {"count_vertices": 1971281, "delimiter": SG_DELIM_TAB, "use_original_ids": False},
    "d12c": {"count_vertices": 39459923, "delimiter": SG_DELIM_TAB, "use_original_ids": False},
    "d13c": {"count_vertices": 41652230, "delimiter": SG_DELIM_TAB, "use_original_ids": False},
    "twitter-small": {"count_vertices": 2391579, "delimiter": SG_DELIM_TAB, "use_original_ids": False},
    "twitter-full": {"count_vertices": 41652230, "delimiter": SG_DELIM_TAB, "use_original_ids": False},
    "buzznet": {"count_vertices": 101169, "delimiter": SG_DELIM_COMMA, "use_original_ids": False},
    "test": {"count_vertices": 4, "delimiter": SG_DELIM_COMMA, "use_original_ids": False},
    "uk2007": {"count_vertices": 105896555, "delimiter": SG_DELIM_SPACE, "use_original_ids": False},
    "yahoo": {"count_vertices": 1413511394, "delimiter": SG_DELIM_SPACE, "use_original_ids": True},
    "wdc2012": {"count_vertices": 3563602789, "delimiter": SG_DELIM_TAB, "use_original_ids": True},
    "wdc2014": {"count_vertices": 1724573718, "delimiter": SG_DELIM_TAB, "use_original_ids": True}
}
```

Figure 4.3: Changes in SG_GRAPH_SETTINGS_DELIM variable in default.py

Then we ran the following commands to prepare the input graph in order to run page rank program.

```
$ cd src/tools/scripts
$ ./convert_graph.py --input ../../../data/data1/d1.txt --output ../../../data/data1/d1.bin
```

Name of txt format file and binary file should be match with the name mentioned in SG_INPUT_FILE variable.

Next we ran the commands as follows:

```
$ ./generate_graph.py --dataset d1c --binary --in-memory
$ ./rebalance_input.py --dataset d1c
$ ./run_tiles_indexer.py --dataset d1c
```

Then we executed the command to run page rank program.

```
$ time ./run_mosaic.py --dataset d1c --algorithm pagerank --max-iteration 5
```

Input graph	**Run time (s)**
Astro Physics (ca-astroph)	Tile error
Condensed Matter (ca-condmat)	Tile error
Gnutella P2P network (p2p-gnutella31)	Tile error
Slashdot 1 (soc-sign-Slashdot-090221)	Did not finish
Slashdot 2 (soc-Slashdot0902)	Did not finish
Amazon product co-purchasing network (amazon0601)	5.974s
Berkeley-Stanford web graph (web-BerkStan)	8.216s
Texas road network (roadNet-TX)	7.192s
California road network (roadNet-CA)	26.774s
Wikipedia Talk network (wiki-Talk)	6.459s
LiveJournal network (LiveJournal)	7.418s
UK 2005 web crawl (uk-2005)	Bus error
Twitter 2010 followers network (twitter-2010)	Bus error

Table 4.7: Run time of page rank on Mosaic for five iterations

Input graph	Run time (s)
Slashdot 1 (soc-sign-Slashdot-090221)	6.033s
Slashdot 2 (soc-Slashdot0902)	5.876s
UK 2005 web crawl (uk-2005)	41.028s
Twitter 2010 followers network (twitter-2010)	Bus error

Table 4.8: Run time of page rank on Mosaic for one iteration

4.6 Result of connected component on Mosaic:

Once we prepare an input graph by running those scripts as page rank, we can just execute the command to run connected component program as follows:

```
$ time ./run_mosaic.py --dataset d1c --algorithm cc --max-iteration 5
```

We can mention number of iterations for connected component in Mosaic.

Input graph	Run time (s)
Astro Physics (ca-astroph)	Tile error
Condensed Matter (ca-condmat)	Tile error
Gnutella P2P network (p2p-gnutella31)	Tile error
Slashdot 1 (soc-sign-Slashdot-090221)	Did not finish
Slashdot 2 (soc-Slashdot0902)	Did not finish
Amazon product co-purchasing network (amazon0601)	1.954s
Berkeley-Stanford web graph (web-BerkStan)	2.341s
Texas road network (roadNet-TX)	4.300s
California road network (roadNet-CA)	33.657s
Wikipedia Talk network (wiki-Talk)	2.293s
LiveJournal network (LiveJournal)	2.896s

UK 2005 web crawl (uk-2005)	Bus error
Twitter 2010 followers network (twitter-2010)	Bus error

Table 4.9: Run time of connected component on Mosaic for five iterations

Input graph	**Run time (s)**
Slashdot 1 (soc-sign-Slashdot-090221)	1.300s
Slashdot 2 (soc-Slashdot0902)	1.297s
UK 2005 web crawl (uk-2005)	47.647s
Twitter 2010 followers network (twitter-2010)	Bus error

Table 4.10: Run time of connected component on Mosaic for one iteration

Chapter 5

In this chapter, We have compared Slashdot 1 (soc-sign-Slashdot-090221) and Slashdot 2 (soc-Slashdot0902) datasets for GraphChi, GridGraph and Mosaic and both these two datasets belong to small group datasets. Then we have compared medium grouped datasets: Amazon product co-purchasing network (amazon0601), Berkeley-Stanford web graph (web-BerkStan), Texas road network (roadNet-TX), California road network (roadNet-CA), Wikipedia Talk network (wiki-Talk), LiveJournal network (LiveJournal). The two large datasets: UK 2005 web crawl (uk-2005) and Twitter 2010 followers network (twitter-2010) have been compared separately.

5.1 Analysis of results of page rank:

In both figure: 5.1 and figure: 5.2, GridGraph shows negligible value compared to GraphChi and Mosaic for both small and medium datasets. For small datasets, GraphChi has higher run time than Mosaic. On the other hand, Mosaic has higher run time than GraphChi for medium datasets. For figure 5.1, we have considered time for running page rank for one iteration. For figure 5.2, we have considered time for running page rank for five iteration.

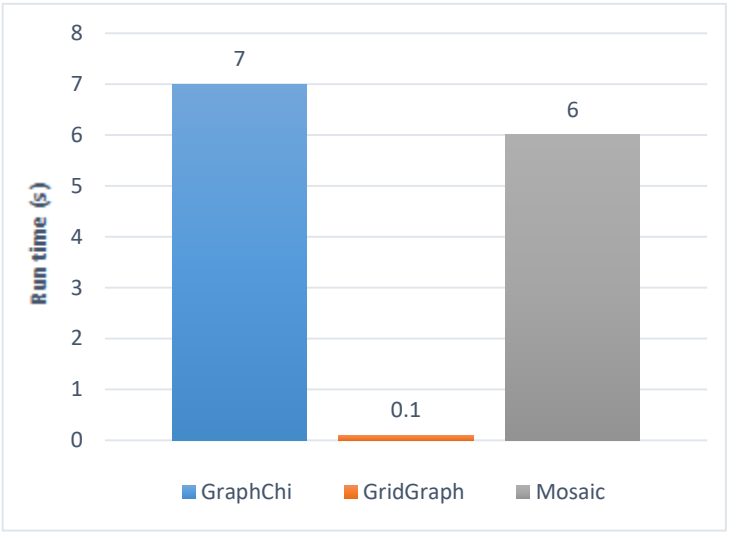


Figure 5.1: Average run time (s) of page rank for small datasets

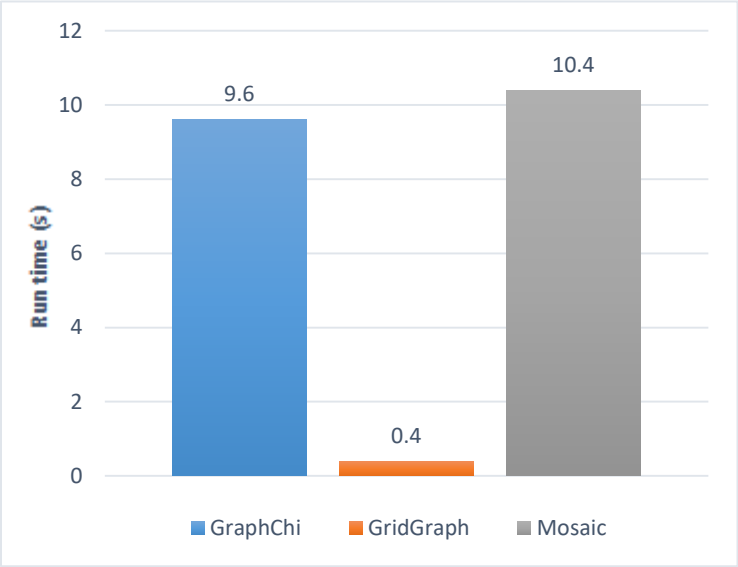


Figure 5.2: Average run time (s) of page rank for medium datasets

According to figure 5.3, the chart of run time for large dataset (UK 2005 web crawl (uk-2005)) shows different pattern in which run time of GraphChi is significantly higher than the other two. In this case Mosaic has the lowest run time having value almost half of the GridGraph. For the comparison, we have taken time for running page rank for one iteration on all three frameworks.

.

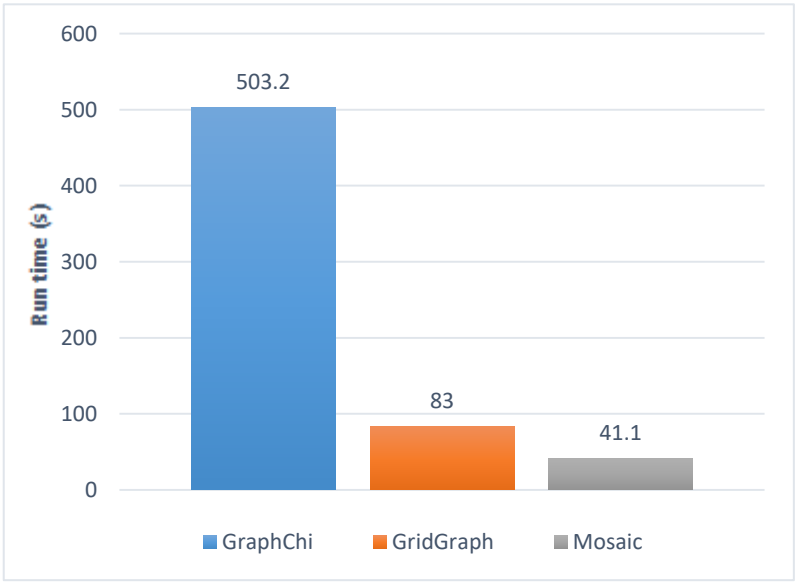


Figure 5.3: Run time (s) of page rank for UK datasets

5.2 Analysis of results of connected component:

In figure 5.4 and figure 5.5, run time of GridGraph for both small and medium datasets are negligible compared to GraphChi and Mosaic. According to chart of the small datasets, GraphChi shows almost 4 times higher run time whereas for medium datasets, GraphChi shows almost 3 times higher run time than Mosaic. For figure 5.4, we have considered time for running connected component for one iteration on Mosaic. For figure 5.5, we have considered time for running connected component for five iteration on Mosaic.

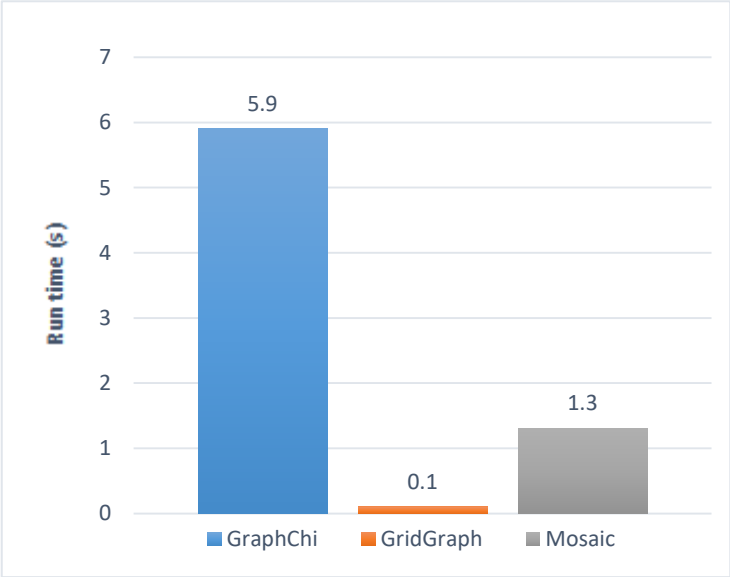


Figure 5.4: Average run time (s) of connected component for small datasets

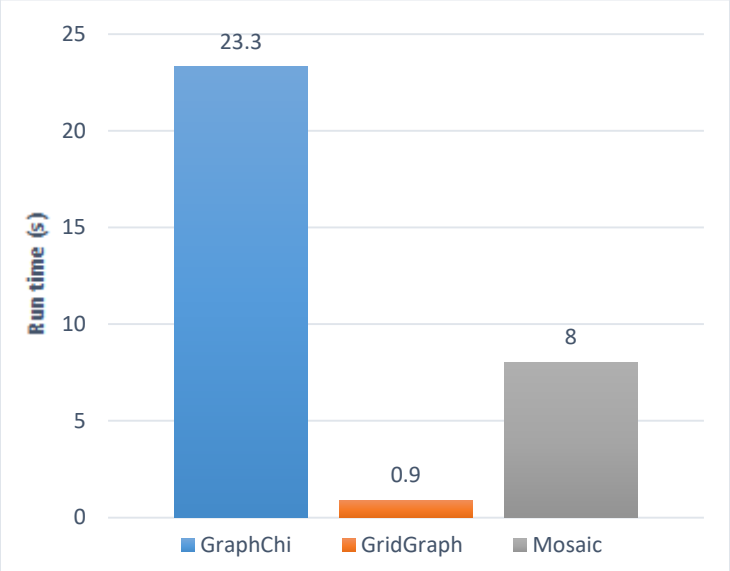


Figure 5.5: Average run time (s) of connected component for medium datasets

From figure 5.6, we can see that run time of GraphChi is significantly higher than GridGraph and Mosaic. On the other hand, run time of GridGraph and Mosaic are almost similar. We have considered time for running connected component for one iteration on Mosaic.

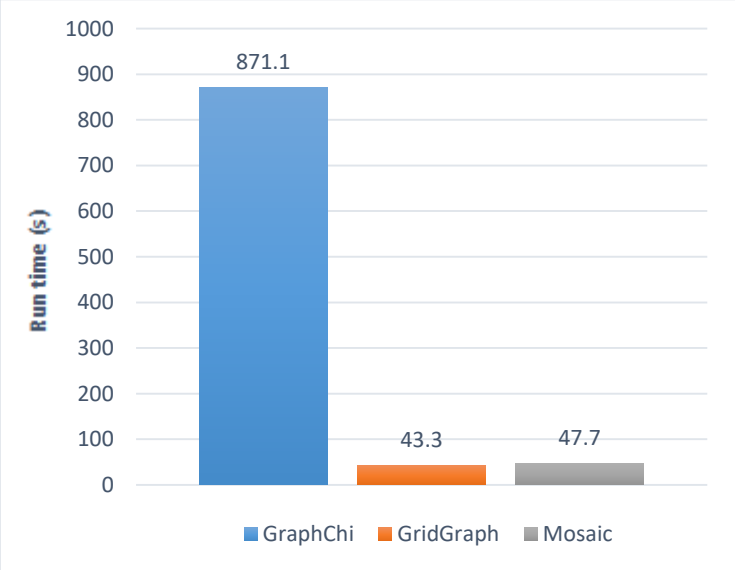


Figure 5.6: Run time (s) of connected component for UK dataset

5.3 Future work:

We have done our experiment on three big graph processing systems on a single machine. A new efficient semi-external-memory big graph processing system named GraphMP (cap-ntu/Graphee, n.d.) which can process big graphs on a single machine, would be considered for future experiment. GraphMP claims that it could outperform state-of-the-art systems such as GraphChi and GridGraph by 31.6x and 23.1x respectively for popular graph applications (such as PageRank, Single-Source-Shortest Path) on billion-vertex graphs (cap-ntu/Graphee, n.d.). Therefore, it can be a part of future work to justify the claim by doing a comparison with the run time, page rank and connected component take on GraphChi, GridGraph and Mosaic.

Chapter 6

6.1 Conclusion:

We have presented all our experiment results for all thirteen datasets. Those datasets are grouped in small, medium and large based on the size. After performing experiment, we have found that the results of page rank for different group of datasets shows different pattern though in all cases, we found the run time of GridGraph is significantly negligible comparing to the run time of GraphChi and Mosaic.

For smaller datasets, GridGraph outperforms 60x and 70x than Mosaic and GraphChi respectively whereas for medium datasets, GridGraph outperforms by 26x and 24x than Mosaic and GraphChi respectively. While GridGraph outperforms the other two for small and medium datasets, Mosaic shows betters performance and it outperforms GridGraph and GraphChi by 2x and 12x respectively for UK dataset. According to the results of connected component, again for small and medium grouped input graphs, GridGraph outperforms Mosaic by 13x and almost 9x respectively and GraphChi by 59x and almost 26x respectively. But for UK dataset, runtime of GridGraph and Mosaic are similar and runtime of GraphChi is 20x and 18x higher than GridGraph and Mosaic respectively. Hence, we can draw a conclusion that runtime of GraphChi is always higher regardless of the size of input graph because the internal preprocessing time is included in running each of the application program. Mosaic shows faster runtime for large input graphs while running for one iteration. But GridGraph performs better for any size of input graphs.

Bibliography

Zhu, X., Han, W., & Chen, W. (2015). GridGraph: Large-Scale Graph Processing on a Single Machine Using 2-Level Hierarchical Partitioning. *USENIX Annual Technical Conference*, 375--386.

Maass, S., Min, C., Kashyap, S., Kang, W., Kumar, M., & Kim, T. (2017). Mosaic: Processing a Trillion-Edge Graph on a Single Machine. *Proceedings of the 12th European Conference on Computer Systems (EuroSys)*.

Kyrola, A., Blelloch, G., & Guestrin, C. (n.d.). GraphChi: Large-Scale Graph Computation on Just a PC.

Khaouid, W., Barsky, M., Srinivasan, V., & Thomo, A. (2015). KCore Decomposition of Large Networks on a Single PC. *Proceedings of the VLDB Endowment*.

Bertsekas, D., & Tsitsiklis, J. (1989). Parallel and Distributed Computation:Numerical Methods. *Prentice-Hall, Inc.*

cap-ntu/Graphee. (n.d.). Retrieved from https://github.com/cap-ntu/Graphee (visited on 2018-04-04)

GraphChi/graphchi-cpp. https://github.com/GraphChi/graphchi-cpp (visited on 2018-04-04)

GraphChi/graphchi-cpp. https://github.com/GraphChi/graphchi-cpp/wiki/Example-Apps (visited on 2018-04-04)

Low, Y., Gonzalez, J., Kyrola, A., Bickson, D., Guestrin, C., & Hellerstein, J. (2010). GraphLab: A New Framework for Parallel Machine Learning. *26th Conference on Uncertainty in Artificial Intelligence*.

sslab-gatech/mosaic. https://github.com/sslab-gatech/mosaic (visited on 2018-04-04)

thu-pacman/GridGraph. https://github.com/thu-pacman/GridGraph (visited on 2018-04-04)

thu-pacman/GridGraph. https://github.com/thu-pacman/GridGraph/issues/2 (visited on 2018-04-04)