Are You Tired of Regular Excel Format?
Data Visualization!!

Parallel a Distributed Algorithm of Finding Maximal Weighted Independent Set with Visualization

Tony Hu
Institute of Computer Science and Engineering
tony84822@gmail.com

Lucy Wu
Institute of Computer Science and Engineering
tingwei0501@gmail.com

Jack Chiang
Institute of Computer Science and Engineering
cmeo5xmm@gmail.com

ABSTRACT

When we trying to do some research study, it is necessary to collect a huge amount of data. As a researcher, you might find it difficult to discover some patterns or specific trend of the data. Therefore, we think data visualization is important and needed nowadays. In our project, we separate it into two parts. First, we try to use one of the python library, Bokeh, to realize our motivation to interactively visualize the data. Second, we implement a distributed algorithm to find maximal weighted independent set (MWIS for short), which is the main part we do parallelism. In the end, we combine these two parts to have a quick demo of visualizing the forming process of MWIS.

1. INTRODUCTION

Recently, data collection is everywhere in our mobile phone, laptop, Mi Band…. As a designer, if we want to find certain patterns and user habits in the huge amounts of data in order to improve our product. How do we efficiently analyze the data? It is a difficult task, therefore, in our project, we try to make data more understandable through visual means for the purpose of helping designers and developers to analyze the data easily. We use Bokeh to help realize our project to visualize the data. In the second part of our project, we parallel a distributed algorithm which is to find a maximal weighted independent set.

1.1 Python Library – Bokeh

Bokeh is a Python interactive visualization library that targets modern web browsers for presentation. Its strength lies in the ability to create interactive, web-ready plots, which can be easily output as JSON objects, HTML documents, or interactive web applications. Bokeh also supports streaming and real-time data. However, if it were possible to keep the "model objects" in python and in the browser in sync with one another, then more additional and powerful possibilities immediately open up to use Bokeh Server. In this scenario, a Bokeh server uses the application code to create sessions and documents for all browsers that connect. (Figure 1.)

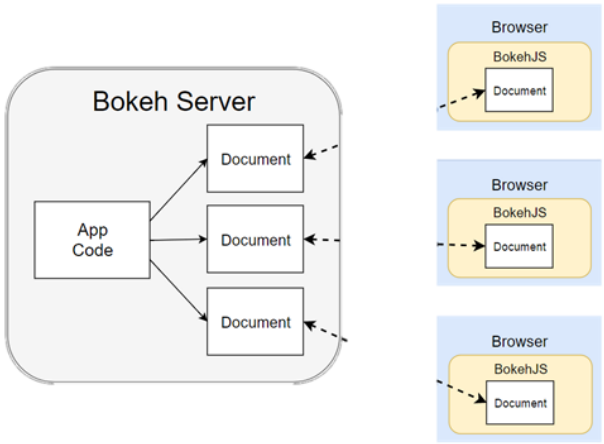

Figure 1.

In our project, we build a Bokeh Server on our own IP domain to handle live demonstration.

1.2 Maximal (Weighted) Independent Set

In an undirected graph G = (V, E), where V is the vertex set and E is the edge set, for example,

in Figure 2-1., S ⊆ V is an independent set if no vertices in S are adjacent to one another like Figure 2-2. An independent set that has the maximum cardinality is called maximum independent set. If each vertex is associated with a weight (a positive real number), then S is a maximum weighted independent set if it is an independent set that has the maximum total weight among all such sets. Clearly, a maximum independent set is a maximum weighted independent set with uniform weight. Finding either set is known to be NP-hard, for which polynomial-time approximation is NP-hard as well.

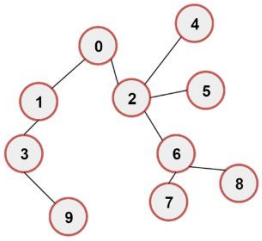

Figure 2-1.

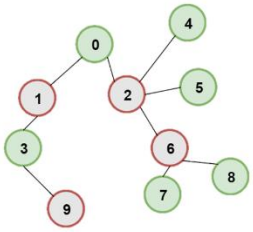

Figure 2-2.

Existing greedy approaches to identifying an MWIS are mostly centralized which works by selecting a vertex into the set, removing it and adjacent vertices from the graph, and repeating this process on the remaining graph.

1.3 Distributed Algorithm Parallelism

It is difficult to do parallel programming based on the centralized algorithm, because the result of MWIS is related in every iteration. Thus, we try to use a distributed algorithm to find MWIS. What do we mean by using a distributed algorithm? The program executions are synchronized in a round basis. All nodes (i.e., processes) executes the following three steps asynchronously within each round.

1. Decide whether it should be in the MWIS or not.
2. Send its decision to each neighboring node. (assuming asynchronous send primitive)
3. Receive each neighbor's decision. (assuming non-blocking receive primitive)

The execution of one process in a round does not affect the execution of another process in the same round. The decision of whether to join the MWIS or not made in the current round is based on the decisions of other processes received in the previous round and does not affect the decision of any process made in the current round. As a result, each process can parallel decide whether or not to join the set. We will have more detailed explanation about how we do parallelism in the next section.

2. PROPOSED SOLUTION

2.1 Bokeh Plots

We ask the user to input a .csv file to start the visualization process. While designing how to display our results, we consider the flexibility of the plots and the user experience. Therefore, we set up the label of x-axis being the first column. Based on the rest of columns, the user can choose one of them to be the label of y-axis. There are more than one kind of plot, we provide at least three kinds of plots, for instance, line, circle, and bar graphs. Figure 3. Shows the architecture of our data visualization programming model.

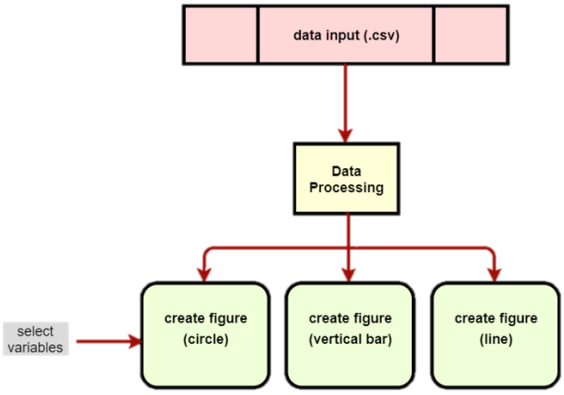

Figure 3.

2.2 MWIS: Using Multi-thread

At first, we try to use multi-thread to parallelize our program. And where is the part we try to parallelize? Because we use Distributed Algorithm, every node would do same actions but independent. So we could parallelize the nodes' behavior. We create threads depended on our computer's core number. But finally we can see the result of the method is not good.

Because of the limitations of python, otherwise GIL the problem we have to face and need to solve in python. So we change the way we take below.

2.3 MWIS: Using Multi-process

According to the above problem, how do we solve? We found the new method is that we use multi-process. Because every process would individually have its own GIL, other process would not get another process' GIL. We use The multiprocessing module, and it also introduces APIs which do not have analogs in the threading module. We also use the Pool object which offers a convenient means of parallelizing the execution of a function across multiple input values, distributing the input data across processes (data parallelism). It would let us more easily parallelize the code.

3. EXPERIMENTAL METHODOLOGY

3.1 Environment

CPU: Intel Core i7-7700 (3.6GHz), 4 Core 8 Thread

Memory: 8GB DDR4 2400MHz'

GPU: NVIDIA GeForce GTX1050 2G

3.2 Data visualization datasets

Our datasets are from Quandle.com which are daily stocks prices of some companies, for instance, Coca-Cola, IBM etc.

3.3 MWIS datasets

We input 1000 nodes into the distributed algorithm to find a maximal weighted independent set.

4. EXPERIMENTAL RESULTS

Before start, we have already compared the method that we used to draw our plot. The blue bar is that we use canvas to draw. And the red bar is used by webgl. If you see the figure below (Figure 4.), you would find that the time used by canvas more than the time used by webgl. Because webgl could make the program run on GPU. That's why we use webgl for this work finally.

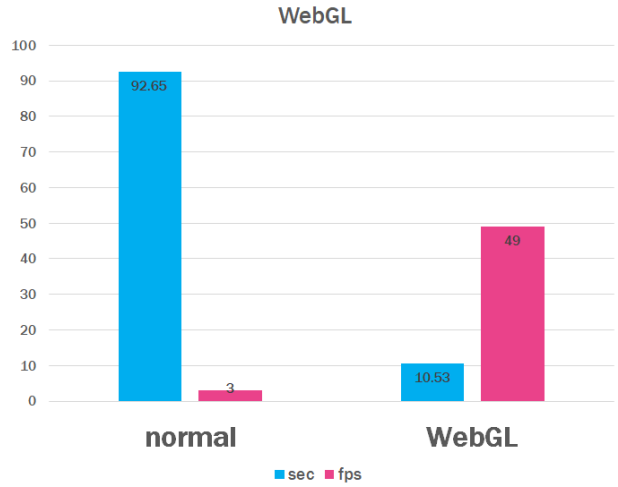

Figure 4.

4.1 Multi-thread

The first method we choose is multi-thread.

But if you see the figure below (Figure 5.), you may wonder why the program time we parallelized by multi-thread was heavily more than the normal time. The problem we figured out is the old defect of cpython -- GIL (Global Interpreter Lock). In python, because of the usage bygones. It is not easy to solve. One process would have one GIL. Even though you create more thread, there is only one thread would get the lock and go to run.

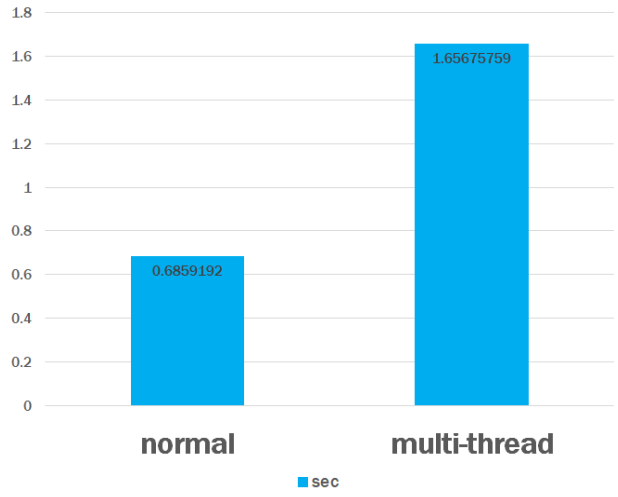

Figure 5.

4.2 Multi-process

Because of GIL, we turned to use multi-process to work on. Even though the time we create process is not a little overhead. But in our experiment there is still positive influence on it.

There is something interesting that the execution time used by one process (0.8225 sec) is larger than normal time (0.6859 sec). In Figure 6, we can see the efficiency of program use by one process is not actually equals to one. The answer we think is because the process we created would cost some overhead.

And we can see the time figure below (Figure 7.). We could find the time in 4 and 5 process decreased slowly, because our environment CPU is 4 cores 8 threads, and the hyper thread is not work fine.

Finally, our other part of experiment statistic is generally rational and expected.

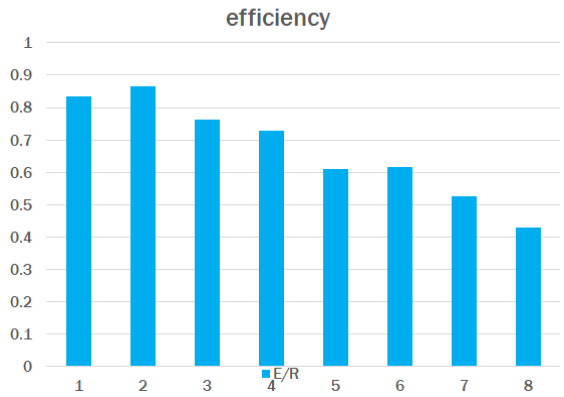

Figure 6.

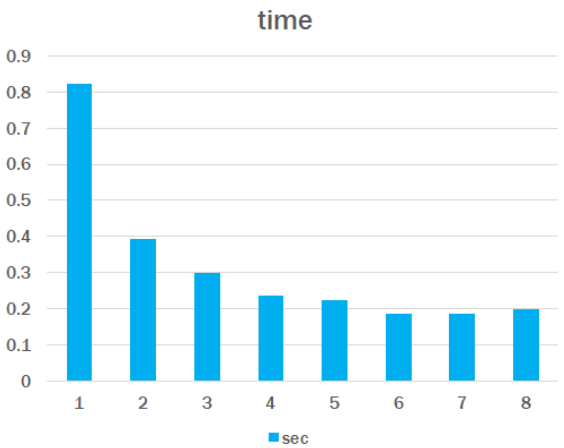

Figure 7.

5. RELATED WORK

The exploration of large datasets is an important but difficult problem. Information visualization techniques may help to solve the problem. D.A. Keim [1] proposed a classification of information visualization and visual data mining techniques which is based on the data type to be visualized, the visualization technique, and the interaction and distortion technique.

6. CONCLUSIONS

In the first part of our project, we use Bokeh to represent the data in several ways like line, circle, and bar graphs. However, we use an off-line algorithm to visualize the data that requires all the data to be available before plotting begins. In the future work, we try to use an on-line algorithm that starts plotting the data while it is being generated.

In the second part of our project, we use multi-thread and multi-process to parallel our program. Nevertheless, we did not get the expected speed up while using multi-thread because of GIL. On the other hand, we truly optimize the running time of the program using multi-process. It is interesting to try different methods to speed up our program.

All in all, maybe we are not the best, but we have learned some other parallel programming skills while working on this project.

7. REFERENCES

[1] Keim, Daniel A. "Information visualization and visual data mining." IEEE transactions on Visualization and Computer Graphics 8.1 (2002): 1-8.

[2] Alon, Noga, László Babai, and Alon Itai. "A fast and simple randomized parallel algorithm for the maximal independent set problem." Journal of algorithms 7.4 (1986): 567-583.

[3] Luby, Michael. "A simple parallel algorithm for the maximal independent set problem." SIAM journal on computing 15.4 (1986): 1036-1053.

[4] https://bokeh.pydata.org/en/latest/