International Core Journal of Engineering Volume 5 Issue 10, 2019
ISSN: 2414-1895 DOI: 10.6919/ICJE.201909_5(10).0009

A parallel algorithm for the maximum flow problem

Ruqing Zhang [a], Wei Wei [b, *]

School of Information Science and Engineering, Henan University of Technology, Zhengzhou, China.

[a]zhangruq_haut@163.com, [b]nsyncw@163.com

Abstract

The maximum flow problem is an important research field in graph theory, and it has a wide range of applications in urban traffic flow, network planning and other fields. There are many approaches to solve the maximum flow problem, but there are still some shortcomings. To solve the problem of high computational complexity of the maximum flow algorithm, a parallel maximum flow algorithm is proposed which divides the original graph into overlay graph and subgraphs. The experiments in TIGER/Line show that the maximum flow result calculated by the algorithm in this paper is consistent with that of the Edmonds-Karp algorithm, the proposed algorithm significantly reduces the calculation time and the time complexity.

Keywords

Parallel computing, maximum flow, Edmonds-Karp.

1. Introduction

At present, many practical problems have been translated into the maximum flow problems to be solved. For example, the maximum flow algorithm was used to settle the maximum traffic capacity of urban traffic network[1]; In the aspect of large-scale resource scheduling, the resource scheduling problem was transformed into the construction and solution of minimum cost maximum flow graph[2], the maximum flow has also been widely used in the fields of network planning and image segmentation[3][4]. Many experts and scholars have discussed the maximum flow problem in many aspects.

The algorithm proposed in [5] used the original dual method, in which each iteration used current calculation to found the enhanced s-t flow in the current residual graph and update the dual solution. [6] used distributed algorithm to solve the maximum flow problem, different calculation models were adopted for each step of the algorithm, the built-in interface pregel in the Spark framework was used to improve the parallelism of the algorithm, but the communication cost of the algorithm was relatively large. [7] proposed an adaptive approach where the algorithm alternate GPU/CPU processing according to the number of active nodes and implementations of the global relabeling and gap relabeling heuristics on multi-core approach. In [8], Simple-Path Locality based Max-flow acceleration algorithm (SPLMax) was proposed, this algorithm only compute necessary vertices those between the source vertex and the sink vertex, which reduces lots of useless calculation. [9]proposed a method to deal with a node that has strongly or weakly connectivity with other nodes respectively, which first calculates the part of the tree with weakly connectivity, and then constructed a cut-equivalent tree by Gomory-Hu algorithm to complete the maximum flow calculation.

To solve the above problems, we propose a parallel maximum flow algorithm which divides the original graph into an overlay graph and several subgraphs with different sizes, after the source and sink vertex are given, the unique path in the overlay graph and the corresponding subgraphs on the path are determined, and the maximum flow of those subgraphs are calculated in parallel, finally, the minimum value of the maximum flow of subgraphs is taken as the maximum flow in the original graph. The experimental results show that the proposed algorithm which divides the original graph into overlay graph and subgraphs and parallel computing subgraphs, which can compute the maximum flow correctly, decrease the calculation time obviously, and effectively reduces the time complexity.

2. Model

Given a graph $G=(V,E)$, the maximum flow problem is to find a feasible flow $\hat{f}(s,t)$ in the graph to maximize the flow from the source vertex to sink vertex, the flow $\hat{f}(s,t)$ is called the maximum flow, and this problem is called the maximum flow problem. A feasible flow is one that meets the following conditions:

$$s.t.\begin{cases} \sum\limits_{v\in\delta^{+}(s)} f(s,v) = \sum\limits_{v\in\delta^{-}(t)} f(v,t)\ (a) \\ 0 \le f_e \le c_e \quad \forall e\in E\ (b) \\ \sum\limits_{w\in\delta^{-}(v)} f(w,v) = \sum\limits_{w\in\delta^{+}(v)} f(v,w)\ \ \forall v\in V-\{s,t\}(c) \end{cases} \tag{1}$$

In the formula, $\delta^{-}(t)$ represents the set of vertices flows into t, $f(v,t)$ represents the flow from v to t, f_e represents the flow of each edge e in the graph, the non-negative number c_e on each edge is the capacity of the edge, $\delta^{-}(v)$ represents the set of vertices flows into v, and $\delta^{+}(v)$ represents the set of vertices flows out of v.

According to equation (1), the total flows out of the source vertex is equal to the total flows into the sink vertex; the flow on each edge is non-negative and does not exceed the capacity; except for the source vertex and sink vertex, the total flows into the intermediate vertex is equal to the total flows out of the intermediate vertex.

3. Algorithm

In the classical maximum flow Edmonds-Karp algorithm, the augmentation road was searched from the start vertex, and every augmenting path adopts the BFS strategy until it was not found. The current flow was the maximum flow when the augmentation path cannot be found.

Different from the Edmonds-Karp algorithm, the algorithm in this paper uses DFS to traverse from the root node of the original graph, it traverses each neighbor node of the current vertex and records the $depth$ value, low value is update to be traced back to the vertex with the lowest value through non-parent vertex, and then backtrack after reach the root node. If low of the neighbor vertex is greater than or equal to $depth$ of the current vertex, thus the current vertex is the cut vertex, then map the cut vertex to the overlay graph, and put the vertex that isn't the cut vertex into the corresponding subgraph, after find all the cut vertices, the connections between vertices are established in the overlay graph according to the link between the cut vertices in the original graph. After given the source vertex and sink vertex in the original graph, the vertices in the subgraph and overlay graph where the source vertex and sink vertex are located are found respectively, then find the shortest path of between the source vertex and the sink vertex in the overlay graph, calculate the maximum flow of subgraphs on the path in parallel, finally, the minimum value of the maximum flow of each subgraph is taken as the maximum flow value of the original graph.

4. Experiment

In order to verify the correctness and performance of the proposed algorithm, the time consumption of the Edmonds-Karp algorithm and the algorithm that divides the original graph into overlay graph and subgraphs for parallel calculation maximum flow are compared in terms of dataset size. The experimental platform is configured as i5-4200@1.6GHz CPUs and 8GB memory, the operating system is 64-bit Windows10. The experimental dataset is taken from 5 real-world graphs in TIGER/Line[10].

According to the above experimental environment and data sets, 50 pairs of source vertices and sink vertices are randomly selected on each data sets for the maximum flow calculation, the calculation results and the average processing time of the Edmonds-Karp algorithm and the algorithm that divides the original graph into overlay graph and subgraphs to calculate maximum flow in parallel are compared. The experimental results show that the maximum flow results calculated by the two algorithms are consistent with 50 pairs of source vertices and sink vertices on the 5 data sets. The proposed algorithm in this paper is always superior to Edmonds-Karp algorithm in terms of time cost. Figure 1 is the experimental result of the parallel maximum flow algorithm that divides the original graph into overlay graph and subgraphs and the Edmonds-Karp algorithm that performs maximum flow calculation on 50 pairs of source vertices and sink vertices on 5 data sets, which verifies that the algorithm proposed in this paper has an order of magnitude reduction in time consumption compared with Edmonds-Karp algorithm.

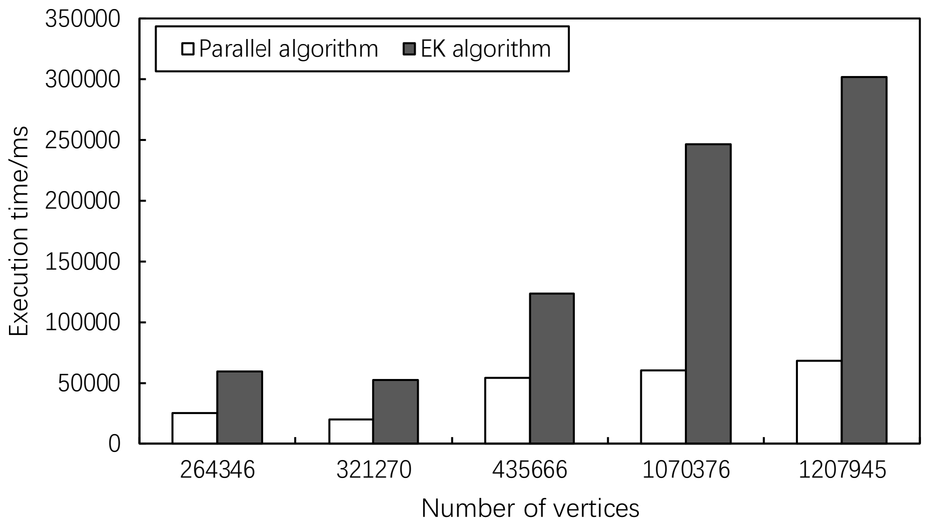

Fig.1 Scale-based time comparison

We can see from Fig.1, the time consumption of the algorithm that divides the original graph into overlay graph and subgraphs in the same data sets to calculate the maximum flow in parallel has significantly less time overhead that of the Edmonds-Karp algorithm. This is because the proposed algorithm generates the overlay graph and subgraphs on each dataset only once, and only the necessary subgraphs need to be calculated, thus the maximum flow calculation on each data set can significantly shorten the maximum flow calculation time. Therefore, the algorithm that divides the original graph into an overlay graph and multiple subgraphs to calculate the maximum flow in parallel can correctly calculate the maximum flow and effectively improve the computation speed of maximum flow.

References

[1] Min, Wang , Q. Yongsheng and W. Shoubao. "Algorithm of urban traffic network capacity based on virtual vertices max-flow." Computer Engineering & Applications 46.11(2010):243-245.

[2] Yi Chen, X. X, et al. "Large-Scale Resource Scheduling Based on Minimum Cost Maximum Flow." Journal of Software (2017).

[3] Sato, Masatoshi , et al. "Image Segmentation Using Graph Cuts Based on Maximum-Flow Neural Network." International Conference on Neural Information Processing 2016.

[4] Liang, Chengchao, F. R. Yu and X. Zhang. "Information-centric network function virtualization over 5g mobile wireless networks." IEEE Network 29.3(2015):68-74.

[5] Madry, Aleksander. "Computing Maximum Flow with Augmenting Electrical Flows." (2016).

[6] Ramesh, Varun, et al. "Max-flow min-cut algorithm with application to road networks." Concurrency & Computation Practice & Experience 29.2(2017):12-22.

[7] Stefanes, Marco Aurelio and L. F. Alvino. "A Hybrid Parallel Implementation for the Maximum Flow Problem." 2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP) 2018.

[8] Wei. Wei, Y. Liu and R. Zhang. "SPLMax: Exploiting the Simple Path Introduced Locality for Maximum Flow Acceleration." IEEE Communications Letters 22.7(2018):1330-1333.

[9] Abboud, Amir , R. Krauthgamer and O. Trabelsi. "New Algorithms and Lower Bounds for All-Pairs Max-Flow in Undirected Graphs." (2019).

[10] United States Road Networks (TIGER/Line) (http://www. dis.uniroma1. it/ challenge 9/data/tiger/)