Parallel Graph Voronoi Diagram on the GPU

HAIMING LONG and JINGWEI JIA

ABSTRACT

Graph Voronoi diagram has important applications in many network problems. We use the GPU computing framework Gunrock to solve the graph Voronoi diagram in parallel with a load balancing method. Experiments show that GPU algorithm has achieved good acceleration compared to CPU algorithm in large scale graphs of hundreds of thousands to millions of nodes. More importantly, the GPU algorithm is not sensitive to the number of Voronoi nodes. The execution time of the algorithm shows a slight downward trend with the increase of the number of Voronoi nodes.

INTRODUCTION

The graph Voronoi diagram data structure is defined by M Erwig et al. [1] The data structure effectively helps solve many network problems such as Nearest Facilities, All Nearest Neighbors and Closet pair, collision-Free Moving [8], anti-centers and furthest points [5] and so on. Graph Voronoi diagram includes inward Voronoi diagram and outward Voronoi diagram.

In a directed graph $G = (N, E)$, the outward Voronoi diagram for a set of nodes $K = \{v_1, \ldots, v_k\} \subseteq N$ is a partition $\{N_1, \ldots, N_k, U\}$ of $N$ so that for each node $v \in N_i$, $d(v_i, v) \leq d(v_j, v)$ for all $j \in \{1, \ldots, k\}$, and $U$ contains all nodes $v$ with $d(v_i, v) = \infty$ for all $i \in \{1, \ldots, k\}[1]$. Note that $d(s, t)$ means the cost of a shortest path between node $s$ and node $t$. The inward Voronoi diagram is the dual of the outward Voronoi diagram.

The inward Voronoi diagram is the symmetry of the outward Voronoi diagram. When building the inward Voronoi diagram, the direction of edges of the graph can be reversed, and then the algorithm of the outward Voronoi diagram can be used to establish the inward Voronoi diagram.

M. Erwig et al. gave a serial and parallel algorithm for the establishment of graph Voronoi diagram. With the expansion of the scale, we need a suitable platform to parallel the establishment of large-scale graph Voronoi diagram algorithm. So in this paper we proposed the use of GPU computing framework - Gunrock[6] to build parallel graph Voronoi diagram.

RELATED WORKS

Mehlhorn [7] used this data structure in solving the Steiner tree problem. Mehlhorn uses the modified Dijkstra algorithm to build the graph Voronoi diagram, but has not yet considered the unreachable nodes. M Erwig et al. formally defined the graph.

Haiming Long, Jingwei Jia, Harbin Institute of Technology, Shenzhen, China
Voronoi diagram data structure, taking into account the unreachable Voronoi node in the graph and further defining the inward Voronoi diagram and outward Voronoi diagram. M. Erwig et al. also proposed a serial and parallel algorithm for solving the graph Voronoi diagram.

PARALLEL GRAPH VORONOI DIAGRAM ALGORITHM

Graph Voronoi diagram Algorithm

M Erwig et al. proposed the algorithm for the graph Voronoi diagram - Parallel Dijkstra. The pseudocode of the algorithm is shown in figure 1. The algorithm is actually a modified version of the Dijkstra algorithm. The algorithm for building outward Voronoi diagram can be simply ported to the establishment of the inward Voronoi diagram by inverting the direction of edges. Therefore, the following default graph Voronoi diagram represents the outward Voronoi diagram. The algorithm first performs an initialization operation. For each node \( v \) in the graph, if the node \( v \) is a Voronoi node, set \( d(v) = 0 \) and put the node \( v \) into Voronoi set \( v \), then insert node \( v \) into the minimum heap; otherwise \( d(v) \) is set to \( \infty \), and the node \( v \) is belong to \( \perp \), indicating that the node is temporarily not belong to any Voronoi set. Next, the algorithm enters a loop. The algorithm executes the expandnext process until the minimum heap is empty. The expandnext procedure first takes the smallest element of the minimum heap and removes \( v \), and then scans the successors of \( v \), that is, the execution \( \text{scansuc}(v) \). \( \text{scansuc}(v) \) scans all successors of node \( v \), ignores the successors that are already marked, and if they have not yet been marked, perform subsequent actions. First, the distance \( d(v) \) of \( v \) plus the distance from \( v \) to \( w \) is set to \( \Delta \). If the distance \( d(w) \) is the original value \( \infty \), then update \( d(w) \) to \( \Delta \), then the node \( w \) is set belong to the Voronoi set that node \( v \) belong to, and the node \( w \) is inserted into the minimum heap \( h \). If the distance \( d(w) \) is less than \( \infty \), then the node \( w \) has been inserted into the minimum heap \( h \). At this point, if the distance \( \Delta \) is smaller than the distance \( d(w) \), the node \( w \) is set belong to a Voronoi set that node \( v \) belong to and the value of the element \( w \) in the minimum heap is reduced to \( \Delta \). Parallel Dijkstra algorithm can be directly implemented in serial and is easy to parallel, the following will detail the implementation of Parallel Dijkstra on the GPU.

![Figure 1. Pseudocode of function of Parallel Dijkstra.](image-url)
Introduction of Gunrock

In order to implement Parallel Dijkstra algorithm with a load balancing method on the GPU, here introduced a GPU computing framework-Gunrock. Gunrock has achieved good acceleration on five graph problems (BFS, BC, SSSP [9], CC and PageRank). The following is a brief introduction to the composition and principles of Gunrock.

Gunrock takes the edges or vertexes of the graph as frontier, and the iterations of Gunrock are equal to the frontier's expansion and contraction processes. As shown in figure 2, each iteration consists of three steps: Advance, Filter, Compute.

Advance expand the frontier while Filter compact the frontier. Advance implement the expansion of frontier by scanning the successors of vertexes or edges of current frontier. Filter compact the frontier by excluding some vertexes or edges that do not satisfy the condition. Compute performs some operations on the current frontier. In Gunrock, Compute is a function nested in Advance and Filter.

GPU Parallel Graph Voronoi diagram Algorithm

The flow chart of solving graph Voronoi diagram on Gunrock is shown in figure 3. First, put |K| nodes into the frontier, and then continue to execute the Advance expansion operation and the Filer filter operation until the frontier is empty. When iteration stops, the graph Voronoi diagram is established.

The pseudo code of solving graph Voronoi diagram on Gunrock is shown in figure 4. We input the directed graph G and the Voronoi node set K, then output the distance array D, the array V indicating the Voronoi set that each node belongs, and the predecessor node array P. Among it, the distance array D holds the shortest distance of each node from the corresponding Voronoi node. The algorithm first put the Voronoi node set into the frontier F, and then enters the loop iteration process. When the frontier F is empty, the iteration stops, and the graph Voronoi diagram is established. Each iteration calls ADVANCE and FILTER operations. ADVANCE is the frontier expansion process which scans the successors of the frontier nodes and performs the COMPUTE_RELAX_DISTANCE operation. COMPUTE_RELAX_DISTANCE calculates the distance of successor nodes. If the calculated distance is smaller than the distance of successor in the D array, the update distance D, the Voronoi set value V and the predecessor P. Otherwise the successor is marked as invalid. FILTER filters out successors that marked as invalid in ADVANCE.
EXPERIMENT AND DISCUSSION

Experiment compare the running time between the GPU Voronoi diagram algorithm and the serial Voronoi diagram algorithm on some large scale graphs.

Data. Experiments test running time and cost of GPU and serial algorithms in three sets of large-scale data coAuthorsDBLP [2] (from social network data, containing 299 thousand nodes, 978 thousand edges), roadNet-CA [3] (from California Highway Network, total Including 1.97 million nodes, 2.77 million edges) and belgium_osm [4] (from Belgium's street network, which contains 1.44 million nodes, 1.55 million edges). Among them, each group of data contains 10 data set, the size of Voronoi nodes is from 100 to 1000. The Voronoi nodes are selected evenly from the vertexes.

![Algorithm Graph Voronoi Diagram on Gunrock](image)

Figure 3. Illustration of algorithm.

![Pseudocode of graph Voronoi diagram on Gunrock](image)

Figure 4. Pseudocode of graph Voronoi diagram on Gunrock.
Environment. The computer configuration of this experiment is as follows. CPU is Intel (R) Core (TM) i3-2100 with frequency 3.10GHZ. The memory is 4.00GB. GPU version is NVIDIA GTX 650. The operating system version of PC is Ubuntu 16.04 64bit, equipped with cuda 8.0 and gcc5.0, g ++ 4.8 and nvcc compiler.

The results of the three sets of data are shown in figure 5. It can be seen from the figure that the GPU algorithm is accelerated in three sets of data with respect to the serial algorithm, averaging 9.8 times on the roadNet-CA data set, averaging 3.5 times on the belgium_osm data set, and averaging 14.3 times on the coAuthorsDBLP data set. It is worth noting that the GPU algorithm is not sensitive to the number of Voronoi nodes in the graph, and as the number of Voronoi nodes increases, the execution time of the GPU algorithm tends to decrease slightly. This is due to the fact that the number of Voronoi nodes is greater, the more the number of GPU threads scheduled for algorithm iteration, the faster the solution of building graph Voronoi diagram.

CONCLUSION

In this paper, the Gunrock calculation framework is used to solve the graph Voronoi diagram in parallel. The experiment shows that the Voronoi diagram algorithm with Gunrock achieves a good acceleration effect when compare to serial version. It is worth noting that the GPU graph Voronoi diagram algorithm is insensitive to the number of Voronoi nodes, and the execution time of the algorithm exhibits a slight downward trend as the number of Voronoi nodes grows.

REFERENCES