I. Introduction

Polygon overlay is one of the complex operations in Geographic Information Systems (GIS). In GIS, a typical polygon tends to be large in size often consisting of thousands of vertices. Sequential algorithms for this problem are in abundance in literature and most of the parallel algorithms concentrate on parallelizing edge intersection phase only. Moreover, spatial data files tend to be large in size (in GBs) and the underlying overlay computation is highly irregular and compute intensive. Motivated by MapReduce programming model and GPGPU, we propose to develop and implement scalable distributed algorithms to solve large-scale overlay processing.

II. Related work

1. Euler approaches include parallelization of Sutherland-Hodgman polygon clipping algorithm and Liang-Barsky line clipping algorithm.

2. MapReduce implementation focuses on spatial join and distributed R-tree implementation.

III. Problem description

Polygon overlay combines the input polygons from different maps into a single new map. The input to binary map overlay are two map layers \( L_1 = \{ p_1, p_2, \ldots, p_m \} \) and \( L_2 = \{ q_1, q_2, \ldots, q_n \} \) where \( p_i \) and \( q_j \) are polygons represented as \( (x_i, y_i) \) co-ordinates of vertices. The output of the overlay operation is a third layer \( L_3 = L_1 \cup L_2 \) \( \setminus \{ q_1, q_2, \ldots, q_n \} \) represented by \( q \) output polygons and this output depends on the overlay operator denoted as \( *_i \) \( (\cup, \cap, \cap, \cap) \). (Union, Intersection, XOR, Difference) determines how map layers are combined.

IV. Technical Contributions

1. \( O(n^2) \) time polygon clipping algorithm: using plane sweep method using \( O(n^2+k) \) number of processors using plane-sweep tree in PRAM algorithm.

2. Design and implementation of an end-to-end spatial overlay system using Hadoop MapReduce platform and MPI.

3. Absolute speedup of 44x using R-tree indexing and grid partitioning using MPI compared to ArcGIS 10.1 and relative speedup of 22x using Hadoop MapReduce platform [1,2].

Algorithm 1 Algorithm Parallel Polygon Overlay

Input: \( V = \{E_1, \ldots, E_n\} \) and \( A = \{E_1, \ldots, E_n\} \).

Step 1: Sort the vertices in \( V \) by their x coordinates.

Step 2: Partition \( E \) into \( k \) subsets \( E_1, \ldots, E_k \), each edge where \( E_i \) belongs to \( n \) scanbeam.

Step 3: For all \( E_i \) in \( E \) in parallel do

Get minimal coordinate of lower(y) and upper(y) for line \( E_i \).

Step 4: \( P_i \rightarrow \) Intersection(y, x)

Step 5: \( P_i \rightarrow \) ProcessEdge(E, (x, y))

end for

Step 6: \( R = P_1 \cap \ldots \cap P_k \)

V. Different Architectures for Distributed Overlay System

VI. Experimental Results

Fig. 1 Intersection of two polygonal maps

Fig. 2 Overlap Graph for polygons from two map layers

Fig. 3 and 4 shows load distribution for two sets of polygonal data

Fig. 5 Here, input is a pair of overlapping polygons. The polygon edges are divided into scanbeams and each scanbeam is processed in parallel. (Output zero or more polygon(s)).

Fig. 6 MPI-GIS with PhDread and CUDA based overlay

Fig. 7 Two level of spatial indexing. Grid-based partitioning and R-tree based indexing. Polygons are mapped based on grid cells. Each reducer performs overlay processing for one grid cell.

Fig. 8 Our MPI based Overlay system based on dynamic load-balancing and R-tree.

Fig. 9 Speedup of the three versions using data set I having 4K*750K polygons for three versions a) Chaining of two MapReduce jobs b) Grid-Row based with map and reduce phase c) Distributed Cache based with a single map phase only

Fig. 10 Speedup of the three versions using data set II having 300K*750K polygons for three versions a) Chaining of two MapReduce jobs b) Grid-Row based with map and reduce phase c) Distributed Cache based with a single map phase only

Fig. 11 Scalability of MPI-GIS using USA shapefile data set

Fig. 12 Absolute speedup of 15x for MPI based overlay system using 8K*400K (data set III) polygons using 80 CPU cores

VII. Future Work

1. GPU Implementation of Segment tree and TPR tree.

2. Multi-layer overlay algorithm

3. Load balanced overlay processing using MapReduce based CPU-GPU cluster.

VIII. References


