MPI-GIS : High Performance Computing and IO for Spatial Overlay and Join

Satish Puri
Mathematics, Statistics and Computer Science
Marquette University
satish.puri@marquette.edu

Danial Aghajarian
Computer Science Department
Georgia State University
daghajarian@cs.gsu.edu

Sushil Prasad
Computer Science Department
Georgia State University
sprasad@gsu.edu

ABSTRACT

Geo-spatial datasets are large and the related computations and analytics are computationally intensive. For certain GIS and Spatial Database applications, spatial overlay and join on two or more layers of geo-spatial data may be necessary. However, using sequential paradigm to process them is time-consuming. For instance, it takes roughly 20 hours to compute the spatial join of a polyline table with 73M records representing the contiguous USA with itself on an Amazon EC2 instance [4]. For these large datasets, I/O, spatial indexing, and geometric refinement phase are time consuming. These operations involve irregular IO due to varying number of vertices in different shapes and irregular computations with no well-defined communication pattern due to irregular spatial and/or temporal task or data distributions. These irregularities makes parallelization, partitioning, and load balancing more challenging. Since, these operations require a lot of floating point operations, harnessing GPU is also essential [3].

We have undertaken parallelization of polygon clipping and overlay algorithms, and spatial join using GPU and MPI on a cluster of nodes in the ROGER cluster [1]. Here, we briefly describe our MPI-GIS system and highlight our work on parallel I/O for OpenStreetMap data [2] using MPI I/O and GPU based spatial join. The IO and parsing timing of a 92 GB WKT file containing 263 million polygons and polylines takes about 76 seconds using 90 MPI processes. Spatial join on this dataset with another dataset containing about 200K polygons takes about 18 minutes only using a cluster with 14 nodes each having 20 processing cores. We also present parallel algorithm for *Intersection* of polygons which is an elementary operations in polygon overlay. Its time complexity is $O((n+k)\log n)$. This is an improvement over our earlier work [5, 6].

Keywords

GIS; Polygon Overlay; MPI IO; GPU; CUDA

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2016 ACM. ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

1. INTRODUCTION

Scalable vector data computation for massive datasets has been a challenge in Geographic Information System (GIS). Spatial Join and Overlay are complex operations in spatial databases and GIS respectively. Both operations need to combine two or more sets of spatial objects like polylines, polygons, etc based on spatial relationships. Given a map of United States representing population distribution and another map representing the area affected by hurricane Sandy, one can use spatial overlay to answer queries such as "What is the optimal location for a rescue shelter?" A typical example of a spatial join is "Find all pair of rivers and cities that intersect."

First, we formally define polygon overlay and spatial join. Polygon overlay is a type of spatial overlay with two or more sets of polygons as input, also known as layers. The input to binary map overlay are two map layers $L_1 = [p_1, p_2, .. , p_n]$ and $L_2 = [q_1, q_2, .. , q_m]$ where p_i and q_i are polygons represented as x,y co-ordinates of vertices. The output of the overlay operation is a third layer $L_3 = L_1 \times L_2 = [o_1, o_2, ... , o_k]$ represented by k output polygons and this output depends on the overlay operator denoted as $\times$.

In contrast to the overlay operation, spatial join takes two sets of spatial records L_1 and L_2 and a spatial join predicate θ (e.g. overlaps, intersects, etc) as input, and returns the set of all pairs (p, q) where $r \in L_1$, $q \in L_2$, and θ is true for (r, s), but not the output polygons themselves. In spatial join, only intersection testing is needed, whereas in polygon overlay, the new polygon's contour needs to be produced. As such, polygon overlay is more compute-intensive than spatial join.

Both operations follow filter-and-refine methodology to produce output. In the filter step, approximation of each polygon, e.g, the minimum bounding rectangle, is used to remove polygons that can not be part of the output. In the refinement step, each pair of polygons passing the filter step is examined according the spatial join condition.

In section 2, we discuss our distributed memory based MPI-CUDA-GIS system which takes advantage of parallel IO and accelerators like GPUS. In section 3, we present our novel segment tree based parallel algorithm for set operations on polygons like intersection and union.

2. MPI-GIS

We have engineered a hybrid MPI/GPU system over a cluster of nodes with a parallel, open software architecture for traditional polygon overlay analysis. *MPI-GIS* achieves 44X speedup while processing about 600K polygons in two

Datasets	File Size (GB)	#Polygons, Polylines (Million)	Sequential I/O time (sec)	Parallel I/O time (sec)	# of MPI Processes
Road Network	137	717	Runs for extended period	47	160
All Map Objects	92	263	Runs for extended period	76.6	90
Roads	24	72	786.34	23.28	75
Lakes	9	8.4	328.63	14.72	45

Figure 1: Sequential and Parallel I/O time comparison using three datasets.

real-world GIS shapefiles 1) USA Detailed Water Bodies and 2) USA Block Group Boundaries within 20 seconds on a 32-node (8 cores each) cluster [6]. The size of these datasets is less than 4 GB. In order to improve the IO scalability for very large datasets as shown in Figure 1, we have incorporated MPI-IO into MPI-GIS.

Parallel I/O in MPI-GIS for vector data: Although, some people use parallel netCDF (PnetCDF) which is a parallel I/O library for accessing NetCDF files, we can not use this library since PnetCDF works for files that are represented in raster format. It cannot handle vector data directly. As such, for large geo-spatial data in vector format, we have developed a parallel I/O library using MPI-I/O on top of GPFS parallel filesystem which can parse the polygon/polyline data into shape objects. This library enables MPI-GIS to handle large vector datasets of hundred gigabytes size extracted from OpenStreetMap as shown in ref-fig:datasets. The basic idea is to partition the entire file into fixed size blocks using MPI functions. The geometric shapes that span across block boundaries are handled after file partitioning. Then, each process parses the file concurrently and builds r-tree/quad-tree index on the shapes. Finally, spatial overlay and join is performed using the index. It took 84 seconds to read, parse, index, and perform R-tree query on the Road Network dataset as shown in Figure 1 against 200K real-world rectangles using 16 compute nodes.

Figure 2 below shows the I/O time with different number of MPI processes. The experiments are done using ROGER cluster at NCSA.

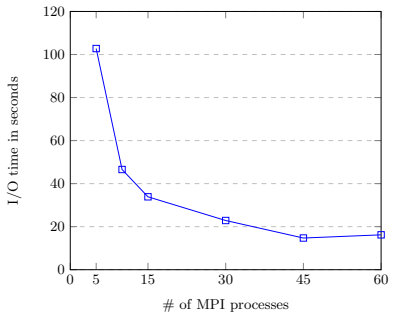

Figure 2: Scalability of parallel I/O for Lakes dataset.

An End-to-End Polygonal Spatial Join on GPU: We have developed an end-to-end software system, that is able to handle spatial join over non-indexed polygonal datasets more than 3 GB file size comprising over 600,000 polygons on a single GPU within 8 seconds. Our system performs a two-step filtering phase: 1) A sort-based Minimum Bounding Rectangle (MBR) filtering step (SMF) detects potentially overlapping polygon pairs up to 20 times faster than the optimized GEOS library routine. 2) A linear time Common MBR filtering (CMF) step (based on the overlapping area of two given MBRs) is employed which not only eliminates two-third of the candidate polygon pairs but also reduces the number of edges to be considered in the refinement phase by almost 40-fold on an average based on our experimental results with real datasets. Furthermore, for the refinement phase, we implement efficient parallel point-in-polygon and edge-intersection tests over GPU. Finally, our experimental results with three different real datasets, show up to 39-fold end-to-end speedup versus optimized sequential routines of GEOS C++ library as well as PostgreSQL spatial database with PostGIS.

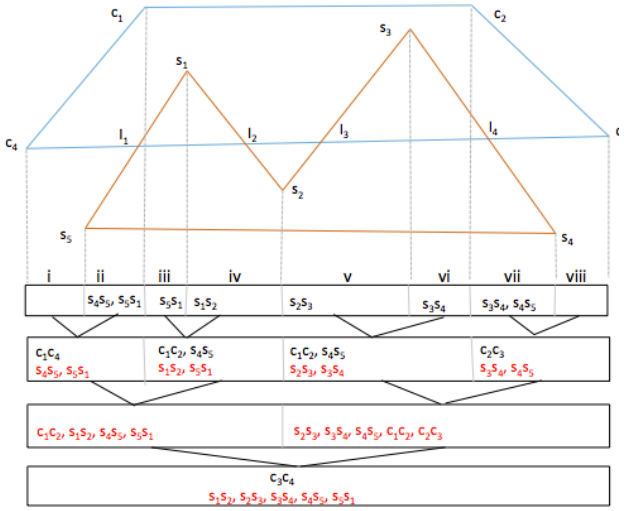

Figure 3: Using segment tree for finding intersection of polygon S ($s_1, s_2, .., s_5$) and C ($c_1, c_2, .., c_4$). Edges in black letters show cover lists and edges in red letters show end lists. I_1, I_2, I_3, I_4 are intersections.

3. PARALLEL POLYGON CLIPPING

Set operations like Intersection and Union on polygons are one of the complex operations in computational geometry. Some of the U.S. State boundaries consist of about 100,000 polygonal vertices. For two polygons with n vertices, the number of intersections can be $O(n^2)$. Sequential algorithms for this problem are in abundance in literature but there are very few parallel algorithms solving it in its most general form. Using segment tree data structure as shown in Figure 3, we have developed the first output-sensitive CREW PRAM algorithm, which can perform polygon intersection and union in $O(\log n)$ time using $O(n + k)$ processors by parallelizing Vatti's algorithm, where n is the number of vertices and k is the number of intersections. Our previous algorithm required k' additional processors for handling additional temporary vertices introduced due to the partitioning of polygons [5].

4. REFERENCES

[1] NCSA ROGER Cluster. https://wiki.ncsa.illinois.edu/display/ROGER/ROGER+Technical+Summary.

[2] OpenstreetMap Datasets. http://spatialhadoop.cs.umn.edu/datasets.html.

[3] S. K. Prasad, M. McDermott, S. Puri, D. Shah, D. Aghajarian, S. Shekhar, and X. Zhou. A vision for gpu-accelerated parallel computation on geo-spatial datasets. *SIGSPATIAL Special*, 6(3):19–26, 2015.

[4] S. Puri, D. Agarwal, and S. K. Prasad. Polygonal overlay computation on cloud, hadoop, and mpi. pages 1–9, 2015.

[5] S. Puri and S. K. Prasad. Output-sensitive parallel algorithm for polygon clipping. In *International Conference on Parallel Processing (ICPP 2014)*. IEEE, 2014.

[6] S. Puri and S. K. Prasad. A parallel algorithm for clipping polygons with improved bounds and a distributed overlay processing system using MPI. In *Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium on*, pages 576–585. IEEE, 2015.