Spatial Data Decomposition and Load Balancing on HPC Platforms

Jie Yang
MSCS department
Marquette University
Milwaukee, Wisconsin
jie.yang@marquette.edu

Anmol Paudel
MSCS department
Marquette University
Milwaukee, Wisconsin
anmol.paudel@marquette.edu

Satish Puri
MSCS department
Marquette University
Milwaukee, Wisconsin
satish.puri@marquette.edu

ABSTRACT

We are in the era of Spatial Big Data. Due to the developments of topographic techniques, clear satellite imagery, and various means for collecting information, geospatial datasets are growing in volume, complexity and heterogeneity. For example, OpenStreetMap data for the whole world is about 1 TB and NASA world climate datasets are about 17 TB. Spatial data volume and variety makes spatial computations both data-intensive and compute-intensive. Due to the irregular distribution of spatial data, domain decomposition becomes challenging. In this work, we present spatial data partitioning technique that takes into account spatial join cost. In addition, we present spatial join computation using Asynchronous Dynamic Load Balancing (ADLB) library. ADLB is a software library designed to help rapidly build scalable parallel programs using MPI. We evaluated the performance of ADLB-based MPI-GIS implementation. In our existing work, spatial data movement cost from ADLB server to worker MPI processes limited the scalability of MPI-GIS.

CCS CONCEPTS

• **Computing methodologies** → **Parallel algorithms**; • **Information systems** → *Geographic information systems*;

KEYWORDS

Message Passing Interface, Parallel IO, HPC, Spatial Join, Spatial Data

ACM Reference Format:
Jie Yang, Anmol Paudel, and Satish Puri. 2019. Spatial Data Decomposition and Load Balancing on HPC Platforms. In *Practice and Experience in Advanced Research Computing (PEARC '19), July 28-August 1, 2019, Chicago, IL, USA.* ACM, New York, NY, USA, 4 pages.

1 INTRODUCTION

With the increasing volume and complexity of spatial data, there is an increasing demand for efficient geo-spatial techniques for parallelizing spatial computations. This paper presents modeling of spatial join complexity, challenges encountered in spatial data partitioning, and experience of using Asynchronous Dynamic Load-balancing library (ADLB) [3] to build MPI-based GIS [4, 5]. Much of our research on big spatial data has been done on a supercomputer named Bridges at the Pittsburgh Supercomputing Center. Our implementations use Geometry Engine OpenSource (GEOS) library which provides 1) spatial data indices such as R-Tree, 2) computational geometry algorithms, and 3) parser for geometric data.

The datasets used in this paper are in Well-Known Text (WKT) format, which records geometry objects on a map as a text markup language. For example, a polygon with 3 vertices is represented as POLYGON((10 20, 30 40, 50 60, 10 20)). A geometry collection can be represented as GEOMETRYCOLLECTION(POINT((12 17)), LINESTRING((3 3, -10 10))).

Organization of this paper is as follows. In section 2, we model polygon intersection costs and reveal one of the causes of load-imbalance in parallel spatial join implementations. In section 3 and 4, we present two contributions of this paper which are 1) spatial join cost-based partitioning and 2) asynchronous dynamic load balancing for geospatial computations. Compared to our earlier work on MPI-based spatial computations, these sections describe new research contributions [2, 4–6].

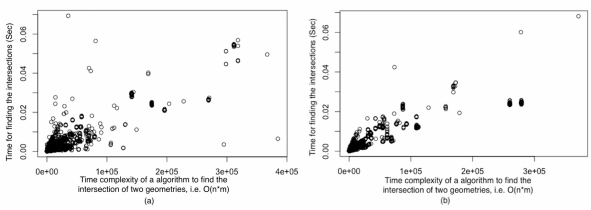

Figure 1: The execution time of polygon intersection using GEOS. In (a), all geometry collections are kept intact; in (b), all geometry collections are broken down into single geometries.

2 MODELING THE COSTS OF SPATIAL COMPUTATIONS

The number of vertices in a polygon/polyline shape varies widely. The number of vertices in the largest feature is about five orders of magnitude higher than smaller features that contain few vertices. Spatial operations like polygon overlay and spatial join where two layers of GIS data are merged together to produce a third layer as

PEARC 2019, July 28–Aug 01, 2019, Chicago, IL, USA

ACM ISBN 978-1-4503-6510-9/18/08…$15.00

DOI: 10.1145/3332186.3333266

output are specially challenging to model. Execution time depends on the degree of overlap between the two input layers. Elementary operations that include cross-layer geometric intersections need to partitioned across MPI processes in balanced way in order to use the processors efficiently. The worst case time complexity to check if two geometric shapes with n and m vertices overlap or not is $O(n * m)$. However, a tighter bound on time complexity is dictated by the number of actual segment-intersection points which is a variable quantity depending on input shapes. This makes modeling task harder.

We performed intersection operation on pairs of geometries which are taken from Lakes (8.4 million polygons) and Sports (1.8 million polygons) datasets [1]. The execution times for different pairs of geometries are distributed as shown in Figure 1 (a). After analysis, we figured out that geometry collections cause the distribution to be more scattered. We performed another intersection operation on the same pairs of geometries with all geometry collections being split to single geometries. After geometry collections are divided, the intersection time shows a better correlation with the theoretical time complexity shown as Figure 1 (b).

3 ADAPTIVE SPATIAL DATA PARTITIONING

For domain decomposition, here we study the partitioning of a layer of spatial data containing polygons/polylines. Similar to join operation in databases, we have spatial join operation that is used in spatial databases and Geographic Information System (GIS). Spatial join finds all-to-all topological relationship between two geometry layers based on whether two shapes overlap or not. With partitioned data, not only the join task is divided into many sub-tasks, but also the spatial query for a single geometry becomes more efficient.

We have embedded our computational cost model inherent in spatial join algorithms to do better partitioning on top of adaptive grid partitioning. We split one grid cell into four grid cells if the cost exceeds a threshold value. For both Quadtree partitioning and uniform grid partitioning, two spatial datasets - Lakes (8.4 M polygons) and Sports (1.8 M polygons), are partitioned into 8192 grid cells. MPI-GIS implementation performs the join tasks that are scheduled in round robin manner to check the quality of different partitioning techniques.

The implementation ran on regular Bridges computing nodes with two E5-2695 v3 CPUs, i.e. 28 cores per node. Figure 2 shows the performances of two partitioning techniques by comparing the maximum execution times and the minimum execution times of the MPI-GIS program. Maximum execution time determines the overall job completion time. The maximum execution times for the data partitioned based on Quadtree partitioning are 20% to 35% lower than the maximum execution times for the data partitioned based on uniform partitioning. With more processes, the minimum execution times are closer to 0 for both partitioning methods.

4 DYNAMIC LOAD-BALANCING

First, we describe our system based on Asynchronous Dynamic Load Balancing Library (ADLB) library [3]. We chose this library because it allows multiple MPI processes to be server process using master-slave strategy. Moreover, it supports asynchronous communication and work stealing among MPI processes. Among P

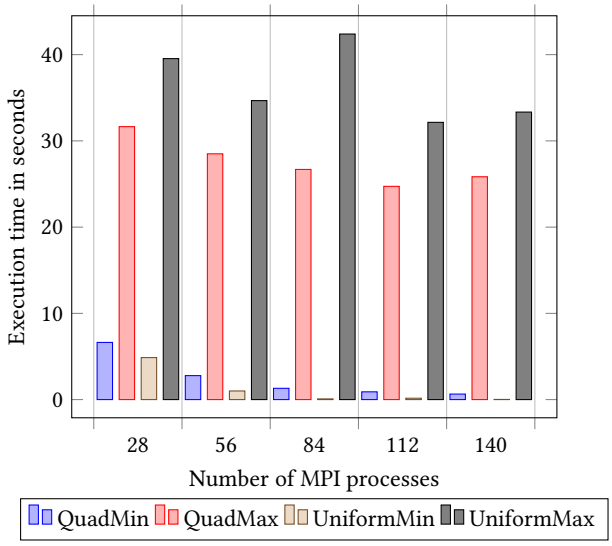

Figure 2: Spatial Join time using Quadtree and Uniform grid partitioning. Maximum and minimum spatial join execution time is shown for different MPI processes using Lakes and Sports.

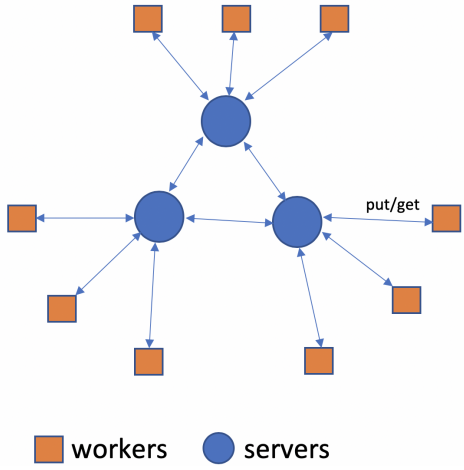

Figure 3: Architecture of ADLB library. The server(s) maintain distributed shared queue. Put and Get operations are used by the workers to add tasks and retrieve tasks to/from the queue. Data can migrate to arbitrary servers depending on the load.

MPI processes, we employ few of them as ADLB servers and the rest are employed as workers. Figure 3 shows the architecture of ADLB. The workers read small input layer entirely in memory and build an R-tree index with geometries in the file. However, only a file split from the larger layer is read in parallel by the workers. These geometries are parsed and enqueued in the distributed shared queue provided by ADLB. The server processes are responsible for load-balancing and do not participate in spatial computations. The

ADLB Server	ADLB Worker
1. Allocate memory for distributed queue. 2. Maintain queue: communicate among servers to distribute polygons dynamically for load balancing. 3. Wait for task completion by ADLB workers.	1. Read polygons from small file and build an Rtree index *indx* using minimum bounding rectangles of polygons. 2. Read a file split from the large file of size N bytes. File split size f = $\frac{N}{\#workers}$ 3. Add tasks to the queue *For each polygon p in f* *ADLB_Put(p);* 4. Retrieve task from the queue until it is empty *while(!q.isEmpty()) {* *string s = ADLB_Get();* *list<polygon> *pList = parse(s);* *perform spatial join(pList, indx);* *}*

Figure 4: Spatial Join Algorithm using ADLB library. The steps followed by each server and worker process is shown. The input to this algorithm is a large spatial data file that gets split across workers and a smaller file.

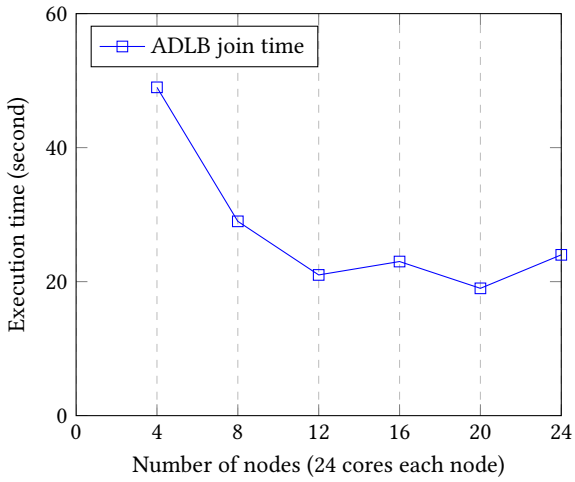

Figure 5: Execution time of ADLB-based spatial join system using two layers - 1) Sports (1.8 million polygons) and 2) Roads (72 million polylines).

workers put all the polygons read from the larger file split in the queue. Finally, geometries are retrieved from the queue and spatial join is carried out. Figure 4 shows the multi-server, multi-worker algorithm in more detail. In this algorithm, we follow a data-parallel approach, where we partition the large file into smaller file splits. However, there is no spatial partitioning done.

As we can see in Figure 5, the spatial join time decreases as the number of workers increases upto twelve compute nodes. In Figure 6, the overhead of adding tasks by the workers is shown. The task here consists of polygons from the large layer. The size of the polygons/polylines vary from few bytes to few megabytes. In this experiment, as we increase the number of workers, we also increase the number of servers from 10 to 20. We can observe a sudden slowdown in task creation phase from Figure 6. This is due to the overhead caused by queue maintenance in ADLB. Beyond 240 MPI processes, ADLB put operations take longer time to finish even though the number of put operations decreases per process.

Inspite of the queue maintenance overhead, we observed that ADLB does a good job of load balancing the tasks among workers. However, the ADLB-based system takes almost double time compared to the single-server multi-worker MPI-GIS system that does not use ADLB (more details in the next paragraph). We found that for ADLB-based spatial join implementation, transferring and parsing data by *put* and *get* operations on the distributed queue are the overheads that increase the end-to-end time considerably.

Load-balancing with partitioned data: In order to improve the performance of parallel spatial computation in MPI-GIS, instead of working directly with original datasets, now we first partition the data adaptively among grid cells as described in the earlier section and then use our own dynamic load balancing implementation. If user selects 8192 grid cells for spatial partitioning, then the original file is broken down into 8192 files i.e. one file for each grid cell data. This version uses a single-sever multi-worker strategy to implement parallel spatial join. A single server works well in this case

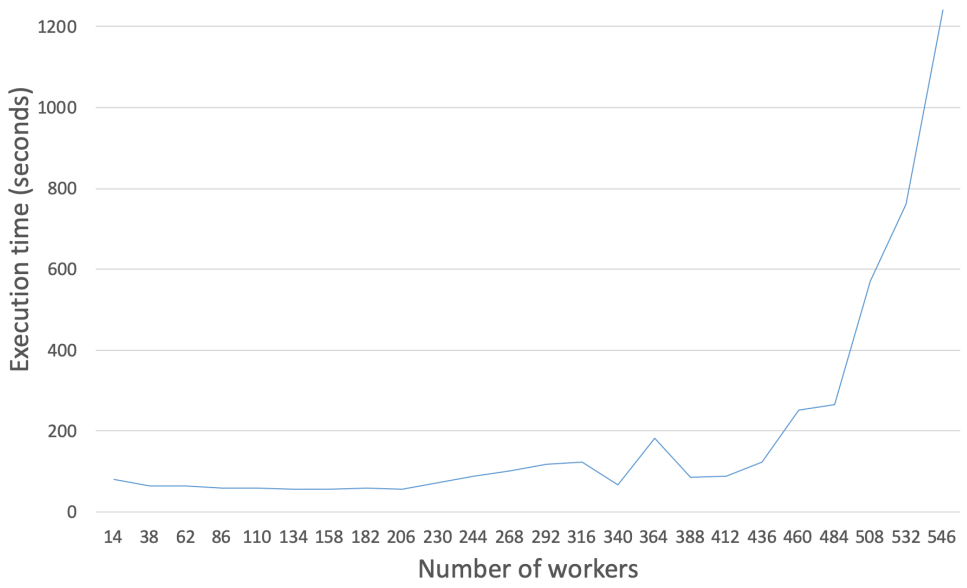

Figure 6: Time taken due to workers adding tasks to the shared ADLB queue using put function.

because the message size is small and due to the coarse granularity of partitions, the server does not become a point of contention.

In this version, a task is defined as a pair of cross-layer and overlapping grid cells. The server sorts these tasks in descending order of computational cost generated by our model and schedules the remaining tasks among workers as they become idle. The program has been tested using two spatial datasets: Roads (72 million polylines stored) and Sports (1.8 million polygons). The WKT file for Roads is 24 GB and the WKT file for sports is 590 MB. We found this implementation that leverages the partitioned data to be faster than ADLB-based implementation.

ACKNOWLEDGMENTS

This work is partly supported by the National Science Foundation Grant No. 1756000.

REFERENCES

[1] [n. d.]. SpatialHadoop, http://spatialhadoop.cs.umn.edu. Website. ([n. d.]). http://spatialhadoop.cs.umn.edu/

[2] Dinesh Agarwal, Satish Puri, Xi He, and Sushil K Prasad. 2012. A system for GIS polygonal overlay computation on linux cluster-an experience and performance report. In *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*. IEEE, 1433–1439.

[3] Ewing L Lusk, Steve C Pieper, Ralph M Butler, et al. 2010. More scalability, less pain: A simple programming model and its implementation for extreme computing. *SciDAC Review* 17, 1 (2010), 30–37.

[4] Satish Puri. 2019. SpatialMPI: Message Passing Interface for GIS Applications. *Geographic Information Science & Technology Body of Knowledge* 2019, Q2 (2019).

[5] Satish Puri, Anmol Paudel, and Sushil K Prasad. 2018. MPI-Vector-IO: Parallel I/O and Partitioning for Geospatial Vector Data. In *Proceedings of the 47th International Conference on Parallel Processing, ICPP*. 13.

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