A HPC Framework for Big Spatial Data Processing and Analytics

Extended Abstract

Anmol Paudel
Mathematics, Statistics and Computer Science
Marquette University
anmol.paudel@marquette.edu

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

ABSTRACT

Spatial data processing and analytics is a highly data- and compute-intensive task. Doing this in a HPC environment has remained a challenge due to scalability issues. The scalability issues usually arise due to the non-uniform nature of spatial data which makes load balancing inefficient. Existing algorithms and systems which are sequential in nature cannot be ported into a distributed system without tackling the issue of load balancing. Porting existing algorithms to a parallel system would also require finding their parallel counterpart and rewriting code in a new language like Cuda or refactoring them using pthreads. In our work, we present a framework for processing and analyzing big spatial data in a HPC environment by overcoming or diminishing the above mentioned issues. Our work includes a way to read the non-uniform spatial data into a MPI system using MPI-Vector-IO. This allows us to reduce the read time of big spatial data and get started on processing faster. ADLB is introduced as a potential solution to handling the issues of load balancing during processing of the big spatial data. Moreover, we propose using directive based programming to port the existing sequential code to run in parallel so that we can avoid rewriting or heavy refactoring. Using OpenMP will allow our code to run in parallel threads in the nodes extracting more performance when possible and using OpenACC we will be able to use any GPUs the nodes might have.

Keywords

GIS; Spatial Data; MPI IO; GPU; OpenMP; OpenACC; ADLB;

1. INTRODUCTION

Recently there has been a huge outburst in the availability of spatial data due to the abundance of GPS enabled devices and satellite imagery. Processing such large volumes of data and running analytics require a lot of time if done sequentially or in a single machine. Vector data as defined by Open GeoSpatial Consortium (OGC) are shapes that are represented with points, lines and polygons. There is a very huge variability in the size of a single shape since the size of single shape doesn't depend on the area it spans but rather on the number of vertices it has. There may be no correlation between the size of a single shape and its spatial distribution. This makes data partitioning extremely hard and require further preprocessing steps.

MPI-Vector-IO will be used to read spatial data in parallel to a spatial data aware MPI environment. It will work on distributing the raw spatial data into different nodes and then parse them into spatial datatypes. From our experiments we have found that an hour sequential operation of scanning 2.7 billion geometries in 96 GB file takes less than 2 minutes with MPI-Vector-IO using 160 MPI processes on GPFS parallel filesystem . Geometry Engine OpenSource (GEOS) is being used for geometry related computations. The library provides spatial data structures and computational geometry algorithms. The partitioned data and GEOS allows us to process spatial data in a distributed manner in parallel. However spatial data comes with its own sets of challenges to scalability due to the lack of correlation between the size and spatial distribution of the data. It can become very difficult to create an implementation for polygon based operations that have load evenly distributed among all nodes. This creates a huge imbalance issue and makes our parallel run time significantly higher due to some nodes constantly running while some nodes exit early. Given the difference in the distribution of data in different datasets the load balancing issue can only be handled effectively during runtime. ADLB utilizes a task stealing strategy to make sure that if any node exhausts its load it can borrow tasks from other nodes [1]. This allows for a more balanced distribution of tasks. However, utilization of ADLB has its overheads and the cost associated with it still needs to be explored in the context of spatial computing.

To write parallel code for CPUs or GPUs, usually an external library (like pthreads) or a new programming language (like cuda) are required. This has huge overheads due to the need to refactor the sequential code or rewrite it in a new programming language. To avoid it, we can use directive based programming to insert hints in the form of pragmas so the compilers can do the necessary parallelization by itself. This reduces the overhead needed to refactor or rewrite the existing sequential code. OpenMP is a programming library for shared memory multiprocessing programming in C/C++ or Fortan. It can be used to add compiler directives to existing sequential code to create shared memory CPU based parallelization. OpenACC is a programming library for simplifying parallel programming of heterogeneous CPU/GPU systems. It can be used to add compiler directive to existing sequential code to create GPU based parallelization.

Our aim is to be able to harness their full potential of each node in the system by using threads spawn by OpenMP. Also

if GPUs are available in the nodes, the existing sequential codes need to be augmented with OpenACC directives to allow it to exploit it. We selected the line segment intersection problem as the The line segment intersection problem is one of the most basic problem in spatial computing and all other operations for bigger problems like polygon overlay or clipping depends on results from it. The line segment intersection problem basically asks two questions - "are the lines intersecting or not?" and if they are intersecting "what is the point of intersection?" To find out the efficacy of directive based programming the parallelization of Bentley-Ottmann plane sweep algorithm using directives appeared to be an appropriate experiment. CGAL (Computational Geometry Algorithms Library) which is a state-of-the-art sequential library for computational geometry algorithms can be the sequential baseline upon which to test the parallel speedup for the code.

Before venturing with a complex algorithm like plane sweep, a naive version of the line segment intersection was coded in C++ and then pragmas were added to create separate samples of OpenMP and OpenACC based parallelization. The naive version is a one-on-one all-to-all approach of testing intersections. This was done to test if line segment intersection was a problem suitable to test parallelization or not. Synthetic data with varying size and intersection density were generated to test the parallel code and get timings.

The first set of tests as shown in Table 1 is comparison of CGAL, our naive code and OpenACC augmented naive code on very dense set of lines. The number of intersection are significantly larger than the number of lines which means that the lines are densely clustered with many of the intersecting with each other.

Table 1: CGAL, naive Sequential vs OpenACC on dense lines

Lines	Intersections	CGAL	SeqCode	OpenACC
400	17164	6.64s	0.02s	2.70s
800	75212	29.25s	0.06s	2.75s
2000	468132	180.6s	0.31s	2.60s

Key takeaways from the first table are that CGAL performance is not as good as the naive version either and it worsens as the number of dense lines increases. Also we can see that irrespective of the data size OpenACC has a base time of a little above 2.5 seconds.

Table 2 is the same comparison as Table 1 but with a sparse set of lines. There are only about ten percent of lines intersecting with each other.

Table 2: CGAL, naive Sequential vs OpenACC on sparse lines

Lines	Intersections	CGAL	SeqCode	OpenACC
10k	1095	3.96s	8.19s	2.87s
20k	2068	9.64s	35.52s	2.95s
40k	4078	17.23s	143.94s	3.48s
80k	8062	36.45s	204.94s	3.87s

Key takeaway from the second table are that CGAL performs significantly better than our naive code for sparse set of lines in sequential and the increase in sequential time is not linear with the increase in data size. OpenACC however drastically beats the sequential performance especially in larger data sizes.

Table 3 is a comparison of how our naive code when augmented with OpenMP performs as we change the number of threads for the same set of sparse lines used in Table 2.

Table 3: OpenMP with varying thread count on sparse lines

Lines	1p	2p	4p	8p	16p	32p
10k	12.53s	6.37s	3.49s	1.95s	1.11s	0.63s
20k	58.46s	28.51s	14.44s	7.84s	4.31s	2.25s
40k	235.1s	116.63s	67.87s	31.09s	17.26s	8.73s
80k	329.13s	170.33s	93.18s	42.27s	23.95s	12.3s

Key takeaways from the third table are although the time taken is not linear with the increase in data size, we can see that it linearly decreases with increase in the number of threads. This shows us that the naive approach to line segment intersection is an embarrassingly parallel problem.

This results ascertain that line segment intersection is a good problem to experiment with in terms of parallelizability. Experimentation with directive based programming on plane sweep using real geographical data should give us further direction.

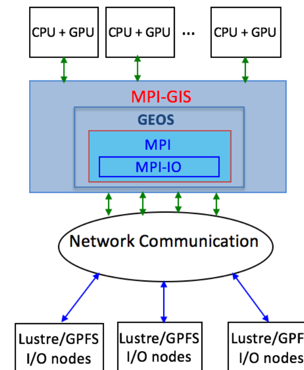

Figure 1: Design of *MPI-GIS* on a parallel I/O architecture.

Faster is better, but in which cases will the speedup really matter? - in any place where the results are time sensitive and any delay can have huge costs associated with it.

- Forecasting and predicting the trajectory of natural disasters like hurricane or flood where the variables are extremely random and every second is crucial for managing evacuation, recovery or relief efforts.
- In domains like epidemiology where early action can save thousands of lives.
- In managing ground troops, where finding alternative routes in real-time can be mission critical.
- In locating lost airplanes, where every passing moment makes it even more difficult.

2. REFERENCES

[1] E. Lusk, R. Butler, and S. C. Pieper. Evolution of a minimal parallel programming model. *The International Journal of High Performance Computing Applications*, page 1094342017703448, 2017.