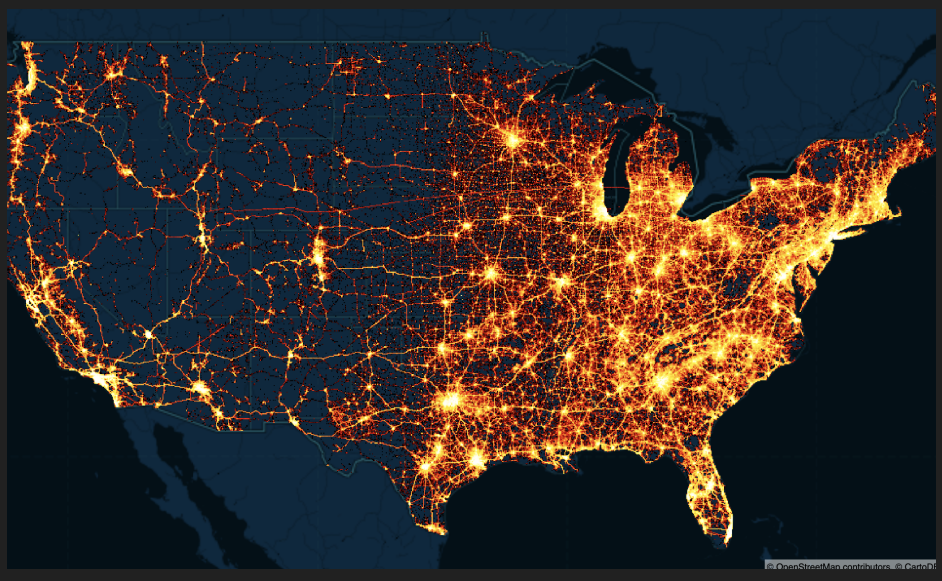
Spatial Algorithms at Scale with Spatialpandas

Dharhas Pothina
Kim Pevey
Adam Lewis

QUANSIGHT

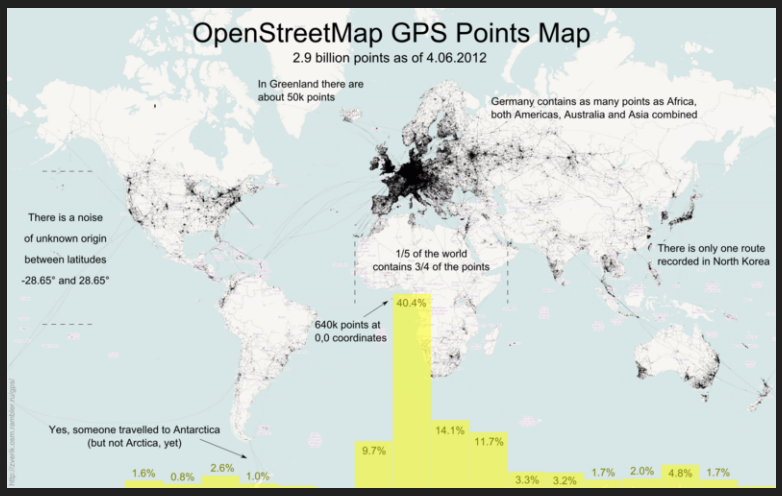
Motivation

- Large Geospatial Dataset (1 trillion rows of point data and growing)
- Data is stored in flat files
- Need to efficiently run spatially localized queries



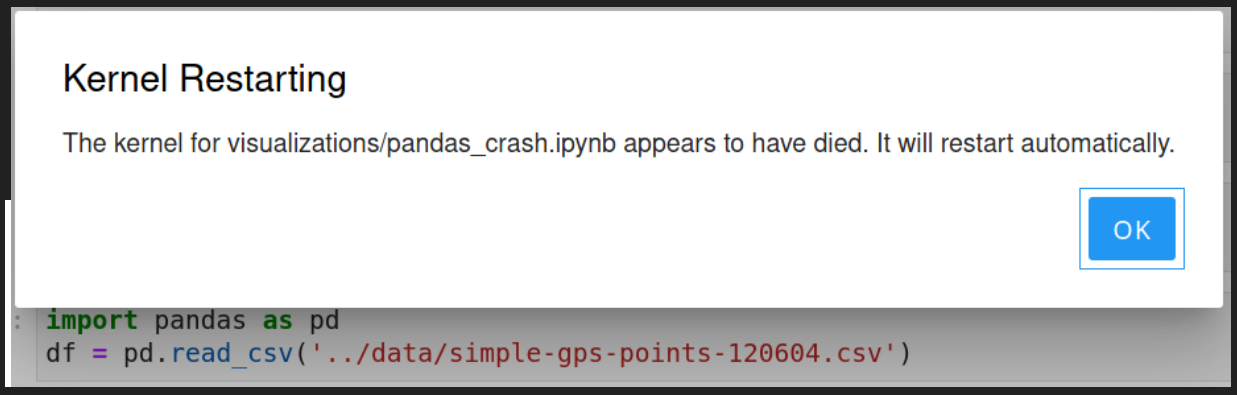
Surrogate Dataset

- All GPS data from OpenStreetMap from June 2012[1]
  - 2.9 billion latitude/longitude point pairs only
  - 73 GB Uncompressed CSV File
  - Contributed by Thousands of Users
- Contiguous US subset will be used in this presentation
  - 114 million rows
  - 3.2 GB Uncompressed CSV File



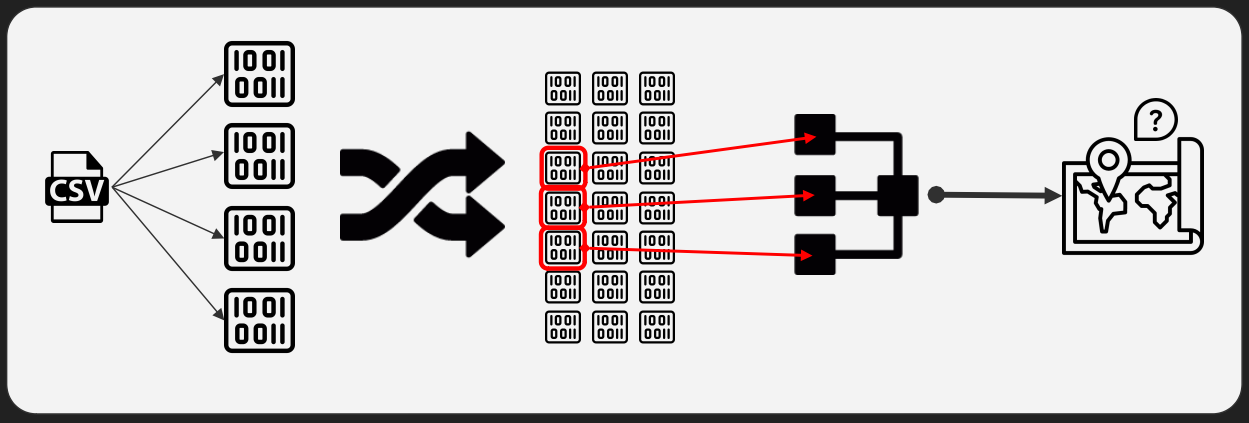
1. Freely available for download from https://planet.openstreetmap.org/gps/simple-gps-points-120604.csv.xz

Problems



1. The dataset is too big to fit in memory
2. Searching through the entire dataset is slow
   a. A naive approach touches every data point during the search
   b. Reading from disk is slow
   c. Finding if all points are within a polygon is computationally expensive

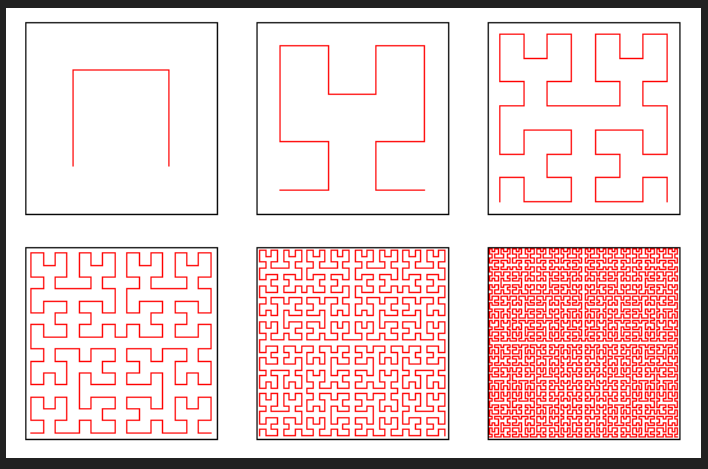
Solution



1. Use multiple smaller binary files or partitions rather than a single csv file
2. Sort the dataset so points that are spatially near are stored in the same file
3. Build a global index so only relevant files loaded based on the local region being queried
4. Conduct queries in parallel

Spatial Sort

- Need to map 2D space (lat,long) --> 1D Space
- The 1D variable must have the property that values close to another are also spatially close to each other.
- Options: **Geohash**, **Hilbert Space Filling Curve,** Google S2, Uber H3,



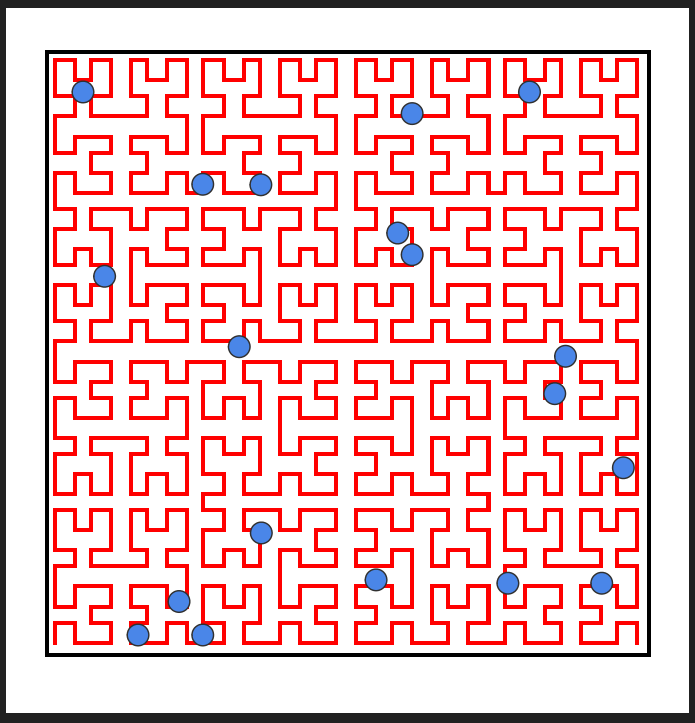
https://en.wikipedia.org/wiki/Geohash
https://en.wikipedia.org/wiki/Hilbert_curve

Geohash

# Geohash Character	Example Geohash	Cell width		Cell Height
1	g	≤ 5,000km	x	5,000km
2	gk	≤ 1,250km	x	625km
3	gk6	≤ 156km	x	156km
4	gk6k	≤ 39.1km	x	19.5km
5	gk6kp	≤ 4.89km	x	4.89km
6	gk6kpt	≤ 1.22km	x	0.61km
7	gk6kptw	≤ 153m	x	153m
8	gk6ktpw9	≤ 38.2m	x	19.1m
9	gk6ktpw9h	≤ 4.77m	x	4.77m

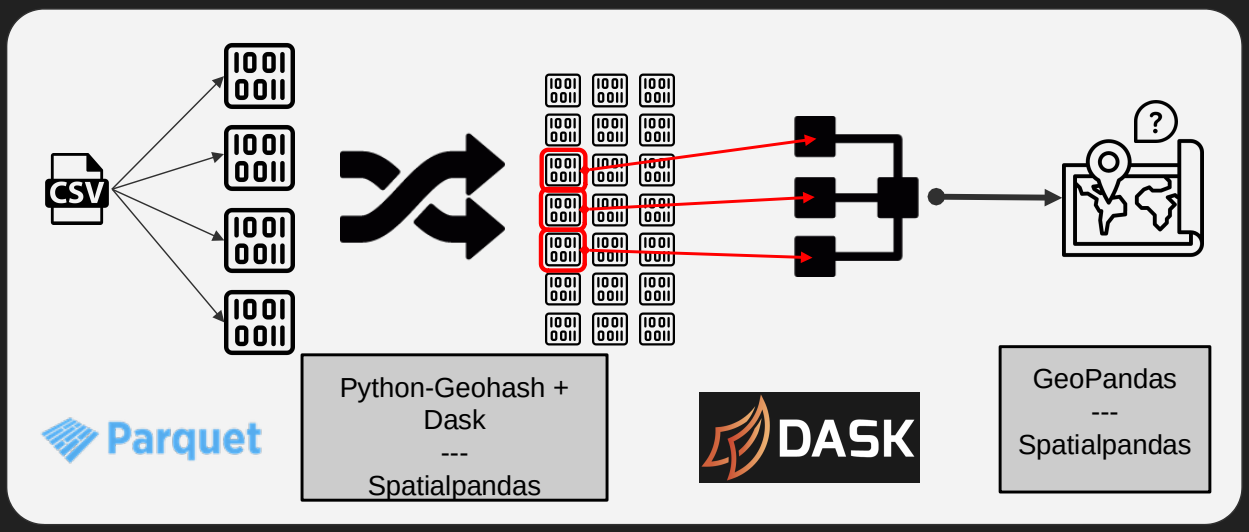
https://en.wikipedia.org/wiki/Geohash

Hilbert Curve



https://en.wikipedia.org/wiki/Hilbert_curve

Solution (Tech Stack)



1. Use multiple smaller binary files or partitions rather than a single csv file
2. Sort the dataset so points that are spatially near are stored in the same file
3. Build a global index so only relevant files loaded based on the local region being queried
4. Conduct queries in parallel

Benchmark:

Machine Specifications

- Processor: AMD Ryzen Threadripper 2970WX 24-Core Processor
- RAM: 64 GB
- Shared System
- For this comparison Dask Workers were limited to
  - 4 Workers
    - 2 Threads per Worker
    - 3 GB RAM per Worker
  - Final computation brings the filtered data into the main process

Dataset

- OpenStreetMaps Unsorted Contiguous US dataset in parquet file format.

Preprocess & Queries

- Preprocess the Data (if necessary)
- Sort the Data
- Select points from dataset that are within 1, 10, 100, 1000, 10000 random zip code polygons distributed around the US
  - Publicly available: https://www2.census.gov/geo/tiger/TIGER2019/ZCTA5/tl_2019_us_zcta510.zip

https://github.com/Quansight/scipy2020_spatial_algorithms_at_scale

1. Unsorted (Naive Approach)

- Preprocess data if necessary:
  - N/A
- Sort data if necessary
  - N/A
- Spatially Index into Data
  - Apply Geopandas's sjoin* method to each partition of GPS data in Dask

* spatial joining finds points in the dataset that are inside the zip code polygons

Unsorted Case

# Polygons	# Points	Geohash Time (s)	Sort Time (s)	Query Time (min)	# Result Points	Total Time (min)
1	113944489	0	0	41	1031	41
10	113944489			41	6551	41
100	113944489			47	203284	47
1000	Not enough memory					
10000	Not enough memory					

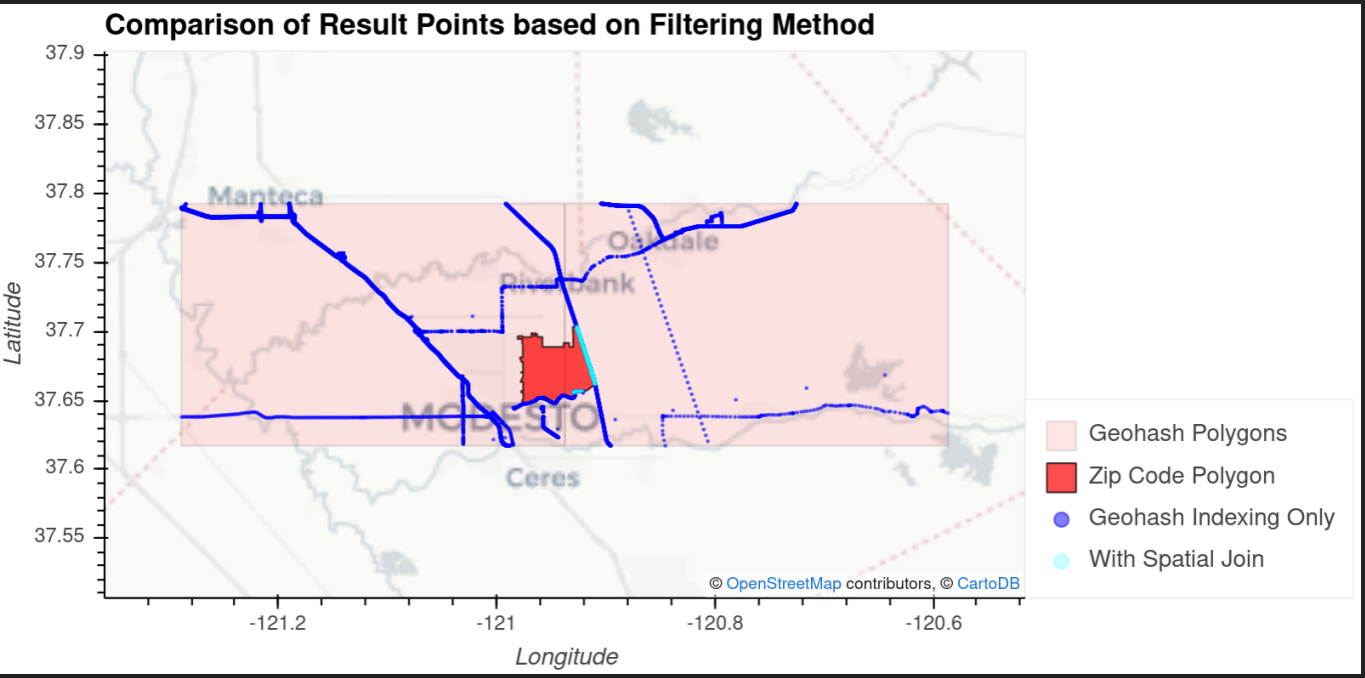
2. Sorted Geohash

- Preprocess data if necessary:
  - Compute Geohash
    - 4 character geohash used
    - Tradeoffs
      - More precise geohash, increased geohash calculation time, reduced sjoin time
- Sort data
  - Sort Geohash Lexicographically using Dask
- Spatially Index into Data
  - Calculate all Geohashes for Each Polygon
  - Index into data by polygon geohashes
    - Only need to read the partitions with data in those geohashes
  - Apply GeoPandas's sjoin method to each resulting partition of points

3. Sorted Geohash No Sjoin (Less Accurate)

- Preprocess data if necessary:
  - Compute Geohash
    - 4 character geohash used
    - Tradeoffs
      - More precise geohash, increased geohash calculation time, reduced sjoin time
- Sort data
  - Sort Geohash Lexicographically using Dask
- Spatially Index into Data
  - Calculate all Geohashes for Each Polygon
  - Index into data by polygon geohashes
    - Only need to read the partitions with data in those geohashes
  - ~~Apply GeoPandas's sjoin method to each resulting partition of points~~

Comparison of Resulting Points



Sorted Geohash Results

Sorted Geohash w/o spatial join (i.e. no point in polygon filter)

# Polygons	# Points	Geohash Time (min)	Sort Time (min)	Query Time (min)	# Result Points	Total Time (min)
1	113944489	27	3	0.05	26413	29
10	113944489			0.07	156625	29
100	113944489			0.08	5269528	29
1000	113944489			0.2	38934176	29
10000	113944489			0.4	90177639	30

Sorted Geohash with spatial join (i.e. with point in polygon filter)

# Polygons	# Points	Geohash Time (min)	Sort Time (min)	Query Time (min)	# Result Points	Total Time (min)
1	113944489	27	3	0.07	1031	29
10	113944489			0.2	6551	29
100	113944489			3	203284	32
1000	113944489			27	2403824	56

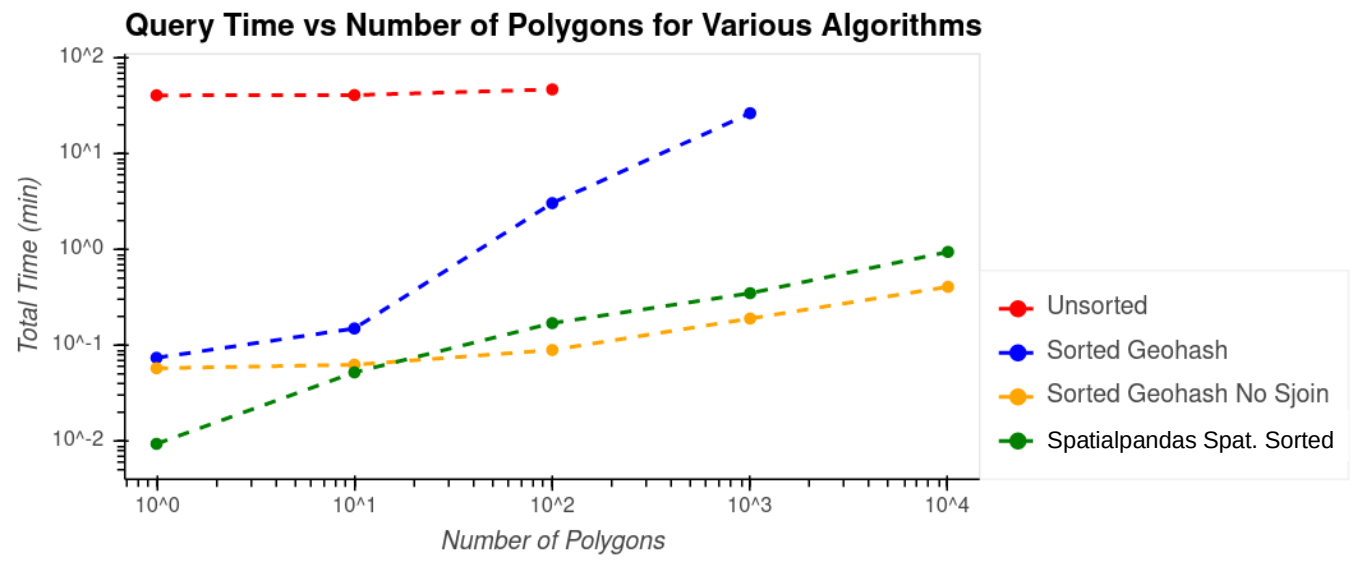
4. Spatialpandas Spatially Sorted

- Preprocess data if necessary:
  - N/A
- Sort data
  - Spatially Sort Data using Spatialpandas
    - Hilbert Curve
- Spatially Index into Data
  - Spatialpandas’s sjoin Parallelized with Dask

Spatialpandas Spatially Sorted

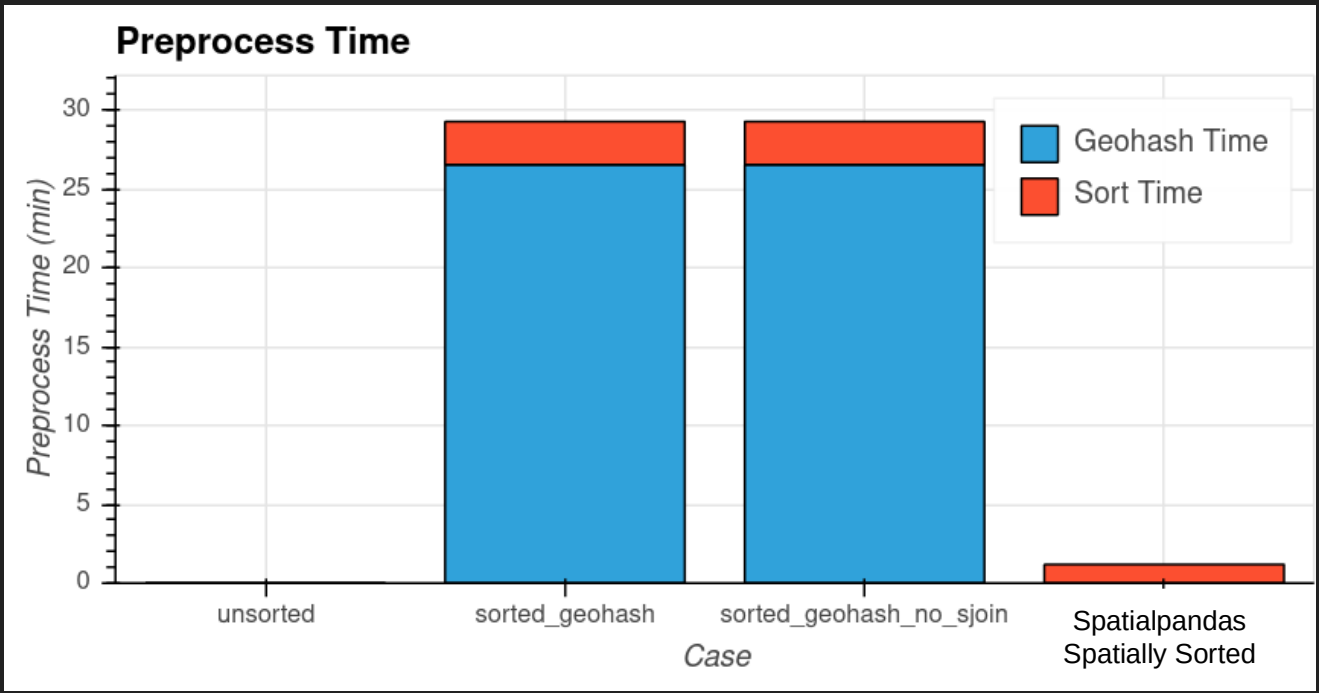
# Polygons	# Points	Geohash Time (s)	Sort Time (s)	Query Time (s)	# Result Points	Total Time (s)
1	113944489	0	71	1	1031	72
10	113944489			3	6551	74
100	113944489			10	203284	81
1000	113944489			21	2403824	92
10000	113944489			57	25877947	128

Results

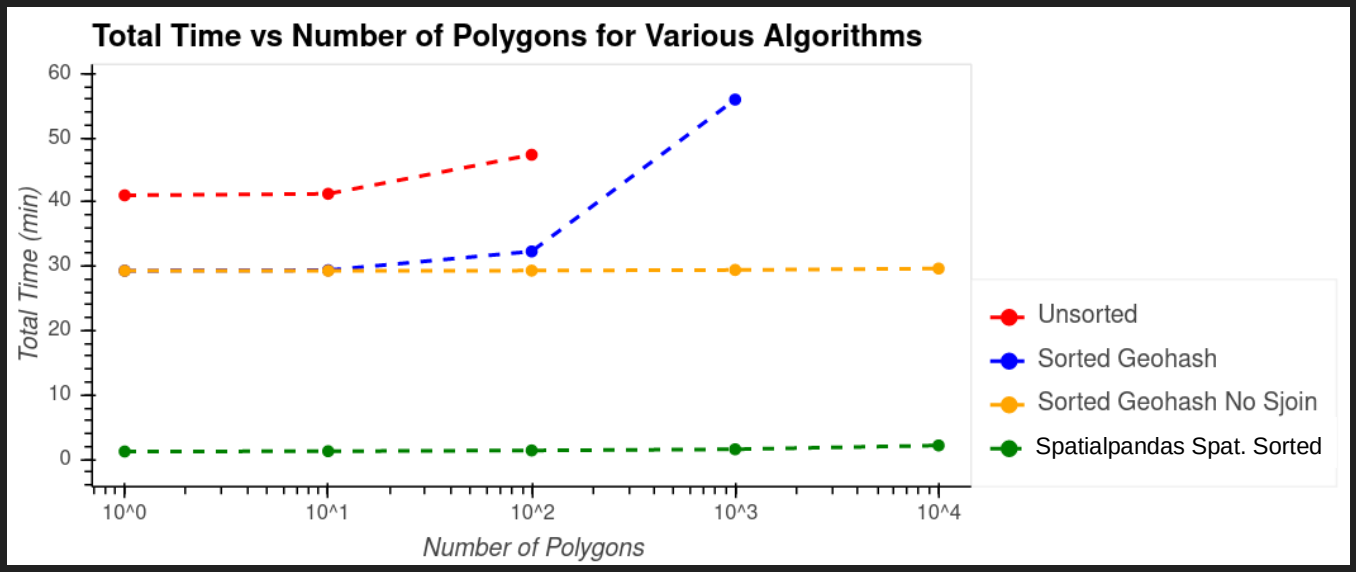


* sorted geohash w/o spatial join is fast but less accurate since it brings in more data than was desired.

Results - One Time Costs



Results Summary

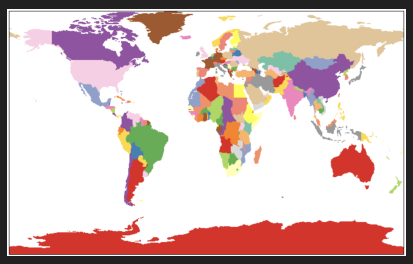


* sorted geohash w/o spatial join is fast but less accurate since it brings in more data than was desired.

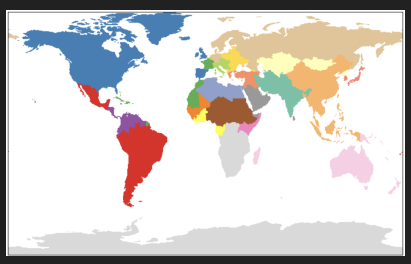
SpatialPandas https://github.com/holoviz/spatialpandas

- Advantages:
  - Easy to install pure python package
  - Vectorized geometry operations implemented with Numba
  - Integrated with Dask Dataframe and Parquet
  - Contains a vectorized and parallel numba implementation of a Hilbert-RTree.
  - Highly efficient parallel spatial joins and bounding box queries on datasets that have been spatially partitioned with spatial pandas
- Disadvantages:
  - Young project
  - Small community
  - Low development activity

Example of spatial sort of country polygons. The images are colored by partition.



Unsorted Polygons



Sorted Polygons

Thank You!

Adam Lewis
alewis@quansight.com

Kim Pevey
kcpevey@quansight.com

Dharhas Pothina
dharhas@quansight.com

Acknowledgements:

- Jon Mease & HoloViz Team
- GeoPandas Developers
- Dask Developers

Code and data for benchmarks:

https://github.com/Quansight/scipy2020_spatial_algorithms_at_scale