Performance Evaluation of GPS Trajectory Rasterization Methods

Necip Enes Gengeç[1(✉)] and Ergin Tarı[2]

[1] Graduate School of Engineering and Technology, Istanbul Technical University, Istanbul, Turkey
gengec@itu.edu.tr

[2] Department of Geomatics Engineering, Istanbul Technical University, Istanbul, Turkey
tari@itu.edu.tr

Abstract. The availability of the Global Positioning System (GPS) trajectory data is increasing along with the availability of different GPS receivers and with the increasing use of various mobility services. GPS trajectory is an important data source which is used in traffic density detection, transport mode detection, mapping data inferences with the use of different methods such as image processing and machine learning methods. While the data size increases, efficient representation of this type of data is becoming difficult to be used in these methods. A common approach is the representation of GPS trajectory information such as average speed, bearing, etc. in raster image form and applying analysis methods. In this study, we evaluate GPS trajectory data rasterization using the spatial join functions of QGIS, PostGIS+QGIS, and our iterative spatial structured grid aggregation implementation coded in the Python programming language. Our implementation is also parallelizable, and this parallelization is also included as the fourth method. According to the results of experiment carried out with an example GPS trajectory dataset, QGIS method and PostGIS+QGIS method showed relatively low performance with respect to our method using the metric of total processing time. PostGIS+QGIS method achieved the best results for spatial join though its total performance decreased quickly while test area size increases. On the other hand, both of our methods' performances decrease directly proportional to GPS point. And our methods' performance can be increased proportional to the increase with the number of processor cores and/or with multiple computing clusters.

Keywords: Rasterization · GPS trajectory · Data aggregation · Spatial join · Parallelization

1 Introduction

Availability of digital data is increasing with the increase of sensor device connectivity and with the decrease in data storage area costs. The availability of spatial data, the data that are having spatial components, is also increasing.

© Springer Nature Switzerland AG 2021
O. Gervasi et al. (Eds.): ICCSA 2021, LNCS 12949, pp. 3–17, 2021.
https://doi.org/10.1007/978-3-030-86653-2_1

Spatial data is collected and stored mostly with the use of Global Positioning System (GPS) receivers or other devices that are equipped with GPS units such as smart phones, navigation devices etc.

Collected GPS data with receivers is also varying. One type of data collected with GPS devices is called GPS trajectories and it is the collection of consecutive GPS locations during the travel time of a moving body [26]. In addition to GPS locations, additional information such as timestamp, speed, bearing of the movement, acceleration/deceleration can be recorded with GPS trajectory and/or can be derived from one another.

GPS trajectories are used in studies focusing on mapping data inference [1,11,14], traffic density detection [12] and transportation mode detection [3,13]. In these examples from literature GPS trajectories are used as is or represented in a generalized form such as embedding attributes into predetermined feature classes or converting GPS trajectories into raster images (Fig. 1) that are representing certain attributes (GPS point frequency, transportation mode) or their attributes' aggregation (average speed, maximum speed, average bearing). After the representation of GPS trajectories with embedding or rasterization, different data analysis methods can be applied to these derived data.

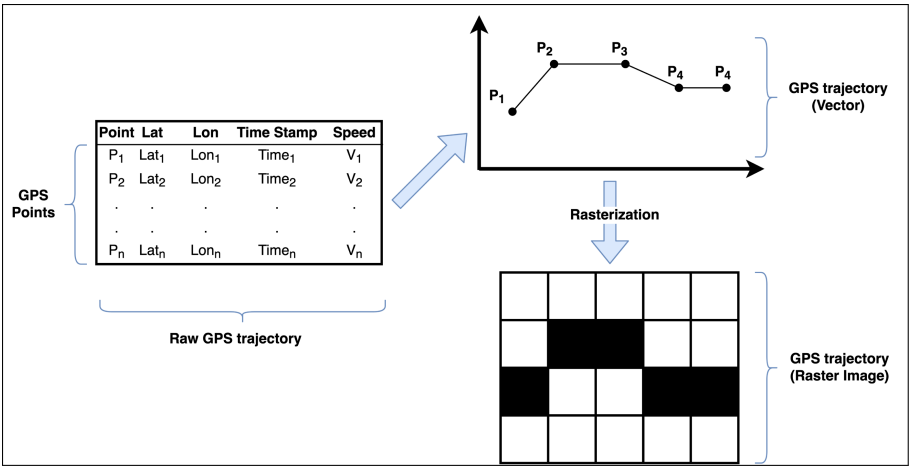

Fig. 1. A simple GPS trajectory and its rasterization.

In addition, research on GPS trajectories as GPS trajectories being the only data source, GPS trajectories fusion with satellite or aerial imagery is a new area [20]. In the context of data fusion of GPS trajectories with satellite imagery, GPS trajectories are rasterized, and it is important to obtain one to one pixel match between rasterized GPS trajectories and satellite imagery to carry out the analysis accurately.

On the other hand, due to the size of GPS trajectory data, the rasterization process can be time consuming while the data size and work area increases. This

issue is not limited only to GPS trajectory rasterization or aggregation, similar research domains such as spatial social media data analysis and other domains that are dealing with high volume point data.

In this contribution, we address the rasterization of GPS trajectories using open source Geographic Information System (GIS) tools and an algorithm coded using the Python programming language. These tools are evaluated according to their performances with an experiment which has the goal to rasterize multiple attributes of given GPS trajectories. Evaluation carried out only on the performance results of approaches for three tools in the same architecture is presented, no philosophical discussion is carried out yet.

To the best of our knowledge, this study is the only study comparing multiple open source tools and algorithms to understand their performance for aggregation of the big point data to structured grids and their rasterization. There has been multiple research for general performance comparison of QGIS with respect to GIS software like ArcGIS [8]. The parallelization is one of the options implemented in this study. There is various research on parallelization for GIS applications spreading from implementing big data tools into GIS software [6,7,25] to adaption of cluster based, distributed big data tools into GIS domain [5,21]. Also, there are significant research which discuss CPU and GPU acceleration, their special applications in GIS and achieved performance improvement [22–24]. Although previous researches may contribute to various future research directions combined with our research results, these researches neither focus on the point to structured grid aggregation and rasterization nor provide a performance comparison with respect to the widely used open source GIS tools.

2 Tools and Rasterization Process Flow

As in raw form, GPS trajectory data is a vector data while the aim of this research is to represent this data in raster image format. Because of these dependencies, tools that are required should be able to handle both vector and raster image data. Within the multiple open source GIS tools that are freely available, QGIS [19] and PostGIS [17] are used for the rasterization of GPS trajectories in this study since they are widely adopted in GIS field thanks to their robustness and abilities.

In Sect. 2.1 and 2.2, the rasterization abilities of QGIS and PostGIS will be examined with respect to given input data (work area boundaries, output pixel size, GPS trajectory data) and expected output rasterized GPS trajectory layers (frequency, average speed and maximum speed). In Sect. 2.3, our implementation will be explained. Finally, in Sect. 2.4, the process flow of methods will be summarized.

2.1 Rasterization with QGIS

As a GIS software, QGIS has different functions, tools and plugins for data analysis and data conversion. *Rasterize (Vector to Raster)* is one of these tools

offered within QGIS. This QGIS tool acts as a user interface, collects the user inputs and runs *gdal_rasterize* tool at the background. This tool is able to get an input data and burn the pixel values that are stored in the preferred attribute field of input vector data within the predefined outer boundaries. Although QGIS has the rasterize tool, this tool is only able to rasterize given values but cannot aggregate multiple values of the same attribute in the given pixels. Though, it is possible to represent pixels in vector form (structured grid) and achieve the required aggregation with spatial join tool of QGIS. After the aggregation of GPS trajectories into the structured grid, it is possible to rasterize the aggregated attributes into raster image data.

2.2 Rasterization with PostGIS

PostGIS is the spatial database add-on for the PostgreSQL [18] database management system. PostGIS is able to store and analyze spatial data in vector and raster image form that is stored in a PostgreSQL database. PostGIS has *ST_AsRaster* tool for similar to QGIS which is accepting the input though producing only given attribute values. On the other hand, similar to QGIS, it is possible to aggregate one vector layer into another using Structured Query Language (SQL) statements. Even though PostGIS does provide rasterize functionality, it is also possible to connect QGIS to PostgreSQL database and rasterize the output data that is created with PostGIS via QGIS.

2.3 Rasterization with Python

Python is a general-purpose programming language which has many internal and external libraries such as data science libraries (Pandas) and geospatial computation (GDAL, pyproj) libraries. With the use of these libraries, it is possible to analyze GPS trajectories.

To achieve the required raster images, our own Python method was created (Algorithm 1). This method gets the GPS points, coordinates of work area boundary and pixel size as inputs and calculates the raster matrix. Unlike PostGIS and QGIS, the Python method makes use of the structured grid definition (work area boundary and pixel size) of a raster image and carries out the calculation of outputs without creating a vector grid. The method determines the row and column of the pixel where each GPS point is contained. Following, it aggregates the required feature values and assigns the output raster image matrix values according to previously determined rows and columns.

The most computation intensive part of the Algorithm 1 is the *for* loop shown between row numbers 5 to 9. Python gives the ability to parallelize with the use of additional libraries such as Dask [4] and Swifter [2]. Swifter library uses Dask library at its backend. It is able to provide the processing time information and provides user the ability to choose parallel or normal computation options easily. Due to these features Swifter library is used for the experiments of this study.

Algorithm 1: Spatial join with Python.

Data: $P = \{p_1, p_2, p_3, ..., p_n\}$ where each p_i contains latitude ($p_{i,lat}$), longitude($p_{i,lon}$), speed ($p_{i,speed}$).
Output raster image top-left corner coordinate (X, Y) and pixel size (px).

Result: Output images $Image_{count}$, $Image_{speed-avg}$, $Image_{speed-max}$ in matrix form.

begin

`/* Convert input coordinates into projected cartesian coordinate system. */`

def *transformCoordinates(P_{lat}, P_{lon})*:

  Transform WGS84 to projected WGS84

return P_X, P_Y

`/* Determining row and column of each GPS point within the output raster. */`

foreach $p_i \in P$ **do**

  $p_{row} = (p_{i,X} - (p_{i,X} \bmod px) - X)/px$

  $p_{column} = (p_{i,Y} - (p_{i,Y} \bmod px) - Y)/px$

  $p_i \longleftarrow p_{row}, p_{column}$

end

`/* Aggregate GPS count, average and maximum speed values with grouping by row and column number. */`

def *aggregateValues(P_{row}, P_{column}, P_{speed})*:

  $P^{'}_{count} \longleftarrow$ count of records having same p_{row}, p_{column} where $p \in P$

  $P^{'}_{speed-avg} \longleftarrow$ average of $p_{i,speed}$ having same p_{row}, p_{column} where $p \in P$

  $P^{'}_{speed-max} \longleftarrow$ maximum of $p_{i,speed}$ having same p_{row}, p_{column} where $p \in P$

return $P^{'}$

`/* Assign pixel values by row and column of output images. */`

foreach $p^{'}_i \in P^{'}$ **do**

  $Image_{count}[p^{'}_{i,row}][p^{'}_{i,column}] \longleftarrow p^{'}_{i,count}$

  $Image_{speed-avg}[p^{'}_{i,row}][p^{'}_{i,column}] \longleftarrow p^{'}_{i,speed-avg}$

  $Image_{speed-max}[p^{'}_{i,row}][p^{'}_{i,column}] \longleftarrow p^{'}_{i,speed-max}$

end

end

2.4 Summary of Methods

According to the spatial data processing and rasterization abilities of QGIS, PostGIS and Python the process flow of the four methods was determined as in the Fig. 2.

The QGIS method creates a vector grid and transforms coordinates of GPS points into the coordinate system of the required output raster image. After creation of the vector grid and coordinate transformation, these are joined spatially. Finally, output of the spatial join is rasterized.

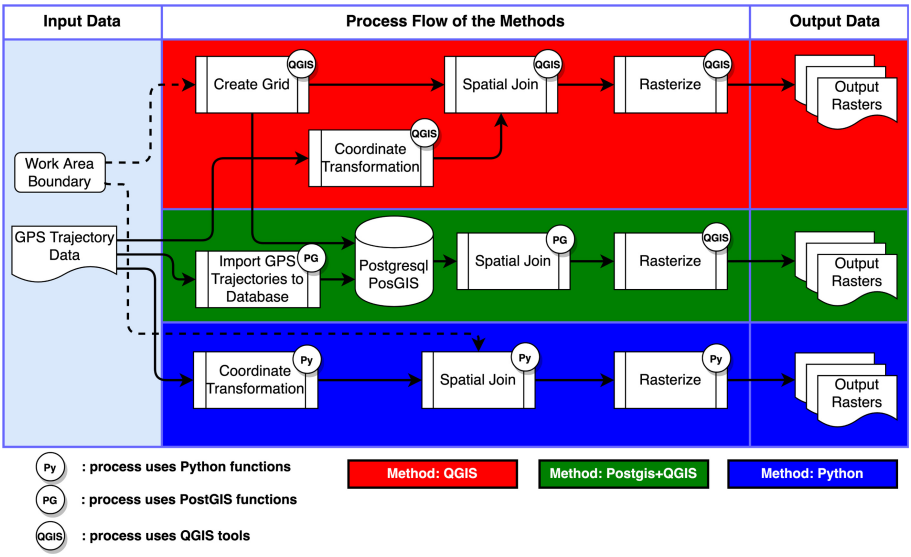

Fig. 2. Process flow of the methods; QGIS, PostGIS+QGIS and Python (Python process flow is identical for both Python and Python (Parallel) methods).

The second method is called PostGIS+QGIS because this method benefits from both PostGIS and QGIS. This method creates the vector grid with the use of QGIS. Also, GPS trajectories are required to be imported into PostgreSQL database which PostGIS is operating on. After import and grid creation, PostGIS spatially joins both data. In this method, coordinate transformation is carried out along with the spatial join. Finally, QGIS is used to rasterize the output of the spatial join.

The algorithm for the Python method is explained in Algorithm 1. This algorithm gets the GPS trajectory data directly using Python libraries and applies the coordinate transformation. After the transformation, our method calculates the output raster image matrix without the need of a vector grid and saves the output raster to the disk.

3 Setup of the Experiments

The methods defined in Sect. 2 are evaluated with the use of MTL-Trajet dataset [10]. This dataset consists of GPS trajectories collected in 2016 around Montreal, Canada. Raw data contains GPS point locations in the WGS84 Datum and timestamps. Speed of the moving objects are calculated using time difference and geodesic distances of consecutive GPS points with Geopy package [9]. These values added as an attribute to the raw GPS trajectory data. Speed value accuracy is dependent to projection and calculation method, but speed accuracy is not the main focus of this study. Because of this, achieved speed values are well enough for performance evaluation of the methods.

In the experiment, the target is to rasterize GPS point "frequency (count)", "average speed" and "maximum speed" raster images using QGIS, PostGIS+QGIS, Python and Python (Parallel) methods. In order to understand the dependencies of performance, the experiment is carried out with varying test area size and GPS point count. Figure 3 shows the test area boundaries and Table 1 summarizes the corresponding GPS count that is used in the experiments. MTL-Trajet GPS point coverage is wider than the defined test areas. The GPS points that are outside of the test area are removed from main dataset to focus only on the performance of methods for given area. Though our Python and Python (Parallel) implementations are able to ignore the GPS points which are out of given area.

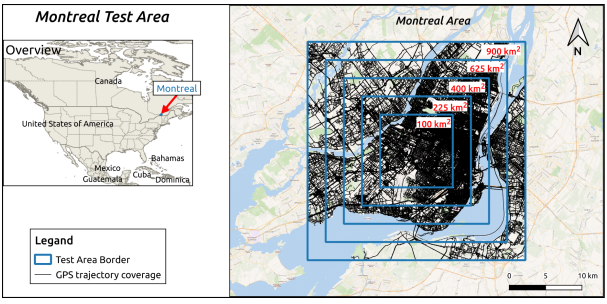

Fig. 3. MTL-Trajet dataset and test areas boundaries.

Table 1. GPS trajectory by test area that was used in the experiment.

Test Area Size (km^2)	Number of GPS points (million)
100	5
225	5, 10, 11
400	5, 10, 15
625	5, 10, 15, 18
900	5, 10, 15, 20

As explained in Sect. 1, the aim is to obtain a one to one pixel match with a given raster image. Usually the raster images that are widely used are provided in projected coordinate systems. In order to add this constraint to the experimental setup, unlike GPS trajectory data, test areas are created in projected WGS84 Datum using cartesian coordinates so that the pixel dimensions are defined in meters. According to this, GPS point coordinates must be transformed into projected WGS84 from geographic WGS84.

The aim of our research for the rasterization is to represent GPS points in raster format to use in further analysis with additional satellite imagery. Because of this, the expected output should be aligned to the satellite imagery pixel resolution. Similar studies in literature use high resolution satellite images that are having pixel resolution around 1–5 m [15,16,20]. In order to align with the literature examples the output raster image pixel size set as 5 m.

Total processing time ($t_{Totaltime}$) and spatial join time ($t_{SpatialJoin}$) are preferred as the evaluation metric. Since the process flow of each method is different, their total processing time is also varying.

Total processing time with QGIS method is calculated with

$$t_{TotalQGIS} = t^{QGIS}_{Gridcreation} + t^{QGIS}_{CoordinateTransform} + t^{QGIS}_{SpatialJoin} + t^{QGIS}_{Rasterize} \quad (1)$$

where $t^{QGIS}_{Gridcreation}$ grid creation time, $t^{QGIS}_{CoordinateTransform}$ coordinate transformation time, $t^{QGIS}_{SpatialJoin}$ spatial join time, $t^{QGIS}_{Rasterize}$ rasterization time with QGIS.

Total processing time with PostGIS+QGIS method is calculated with

$$t_{TotalPostGIS+QGIS} = t^{QGIS}_{Gridcreation} + t^{PG}_{SpatialJoin} + t^{QGIS}_{Rasterize} \quad (2)$$

where $t^{QGIS}_{Gridcreation}$ grid creation time with QGIS, $t^{PG}_{SpatialJoin}$ spatial join with PostGIS, $t^{QGIS}_{Rasterize}$ rasterization time with QGIS.

Total processing time with Python method is calculated with

$$t_{TotalPython} = t^{Py}_{CoordinateTransform} + t^{Py}_{SpatialJoin} + t^{Py}_{Rasterize} \quad (3)$$

where $t^{Py}_{CoordinateTransform}$ coordinate transformation, $t^{Py}_{SpatialJoin}$ spatial join and $t^{Py}_{Rasterize}$ rasterization using Python.

Experiments are carried out using a standard laptop which has an Intel Core I7-4600 M CPU with four 2.90 GHz cores, 16 GB RAM and Ubuntu/Linux operating system. The QGIS and PostgreSQL/PostGIS are used with their default installation settings. Spatial indexes are used for GPS point and vector grid layers in PostGIS+QGIS method. In order to avoid delays caused by memory and processor usage of another software, minimum required software was kept open while running a method.

4 Comparisons

Experiments are carried out with the methods defined in Sect. 2.4 and with the setup defined in Sect. 3. Following the experiments, output raster images are compared with the use of raster calculation. All output raster images subtracted from remaining output images one by one for given test area. This comparison aims to determine if the created raster images of each method is identical or not. If compared raster images are identical, empty raster image expected as the result of raster image subtraction. According to the comparison of the outputs of each test area, the subtraction results were empty images which proves that each method achieved the same raster images as output.

With the use of the output data from the experiments, comparison plots are created. Figures 4 and Figs. 5 show the performance comparison of methods for each test area. Firstly, as seen in Fig. 4a, Python (Parallel) and Python methods achieve best performance in terms of total processing time and followed by PostGIS+QGIS method within the 900 km^2 test area. On the other hand, PostGIS+QGIS achieves the best performance for spatial join time measure (Fig. 4b). QGIS method achieves very poor performance for each measure. Because of this, QGIS method is excluded from plots in the Figs. 5.

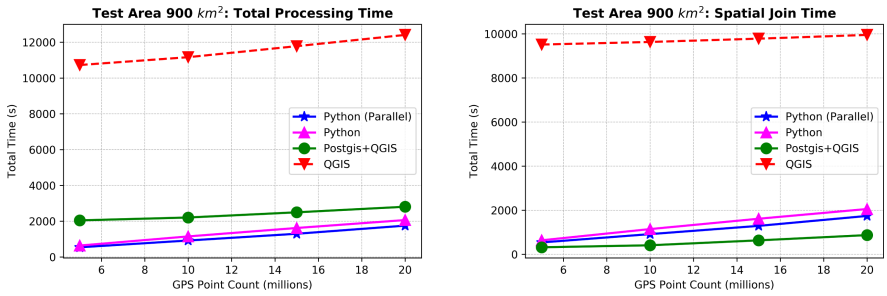

Fig. 4. (a) comparison of total processing time and **(b)** comparison of spatial join time for 900 km^2 test area.

The results for both measures are very similar for the rest of the test areas (Fig. 5). It is also visible that the Python method without parallelization is slower than PostGIS+QGIS method in small areas though performance increases with respect to the PostGIS+QGIS method while the test area size increases (Fig. 5c, 5e, 5g). In the spatial join measure, PostGIS+QGIS achieved better performance followed by the Python (Parallel) method. With the increase of GPS point count, the difference between Python methods and PostGIS+QGIS method also increases.

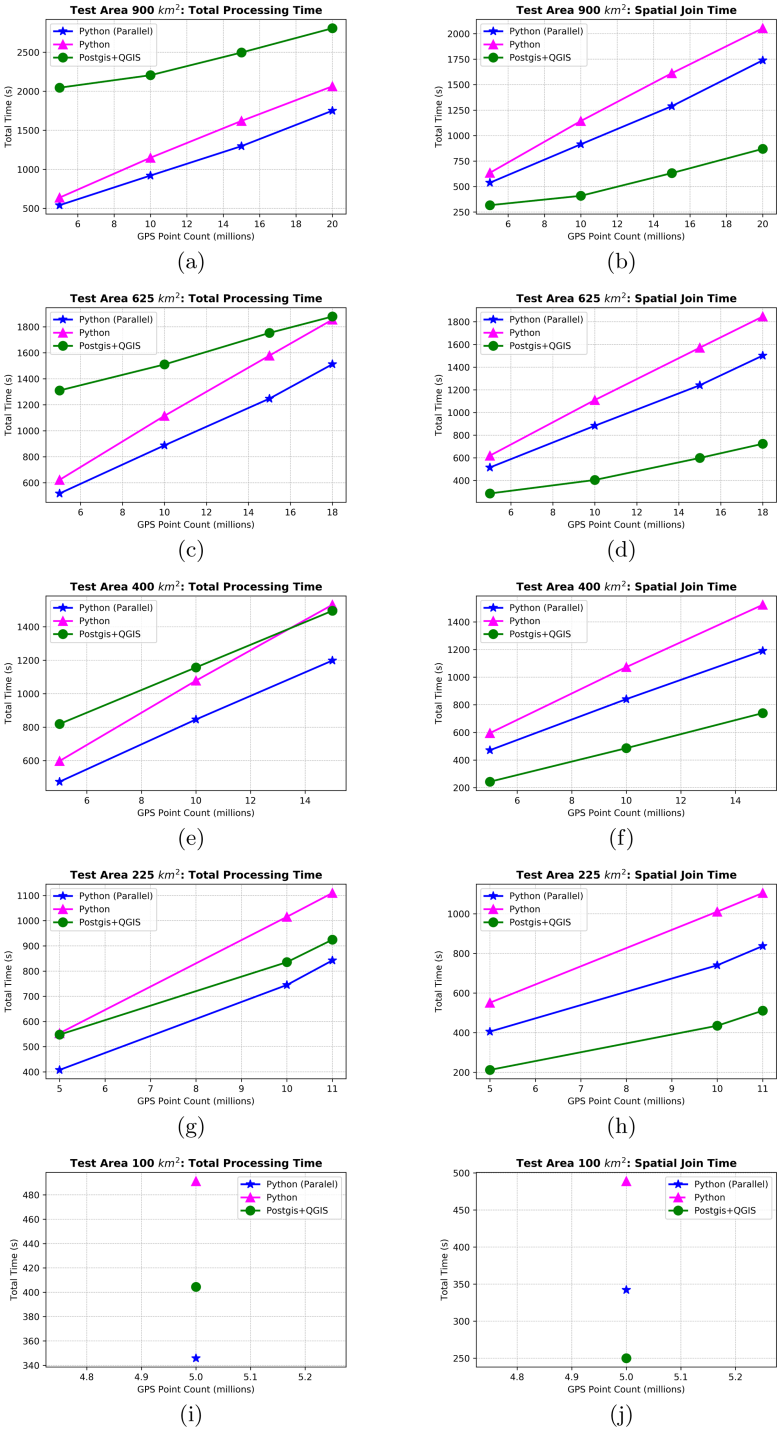

Fig. 5. Comparison of total processing time and spatial join for each test area (QGIS method's results are excluded).

There are two major reasons for the performance decrease of PostGIS+QGIS method in total processing time. The first and most important reason is the requirement of a vector grid. As summarized in Table 2, grid creation time is proportional to the test area size and increases as the test area size increases. The second reason is the importing time of the GPS trajectories to the database. Unlike the QGIS method and Python methods, GPS trajectories required to be imported to the database before starting the rest of the process for PostGIS+QGIS method. Figure 6 shows that this process is dependent on the GPS point count and not dependent on test area size.

Table 2. Grid creation time for each test area.

Test Area Size (km^2)	Time (s)
100	78
225	164
400	299
625	466
900	706

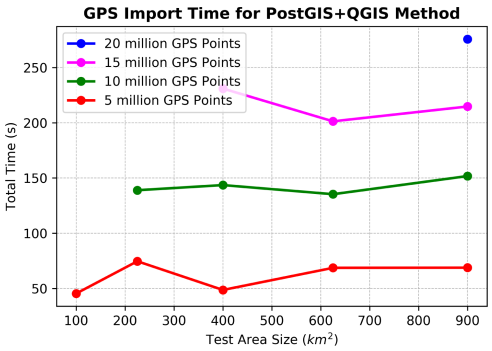

Fig. 6. Time spent for GPS trajectory data to database import for PostGIS+QGIS method.

Lastly, method results are compared internally with the performance measure by test area size when GPS count is kept constant in plots (Fig. 7). As per this comparison, total processing time of the QGIS method increases proportional to the test area size (Fig. 7d). Similar to QGIS method, PostGIS+QGIS method's total processing time also increases proportional to test area size though the increase is steeper compared to the QGIS method. On the other hand, both Python (Parallel) and Python methods show very few increases when test area size increases. Their performances are proportional to GPS point count. Python (Parallel) method is faster than Python method.

In addition to the experiments defined in Sect. 3, an additional experiment was also carried out to understand the limitations of these methods. This experiment was carried out with test area size 22120 km^2 and approximately 25 million GPS points around Montreal. Python and Python (Parallel) methods have been able to process and rasterize this area in 40 and 52 min respectively. On the other hand, QGIS and PostGIS+QGIS methods couldn't process this larger test area due to grid creation with the current hardware. Since grid creation time is proportional to test area size for both methods, in the case of a big test area, grid creation cannot be possible with the use of QGIS and at the end QGIS crashes.

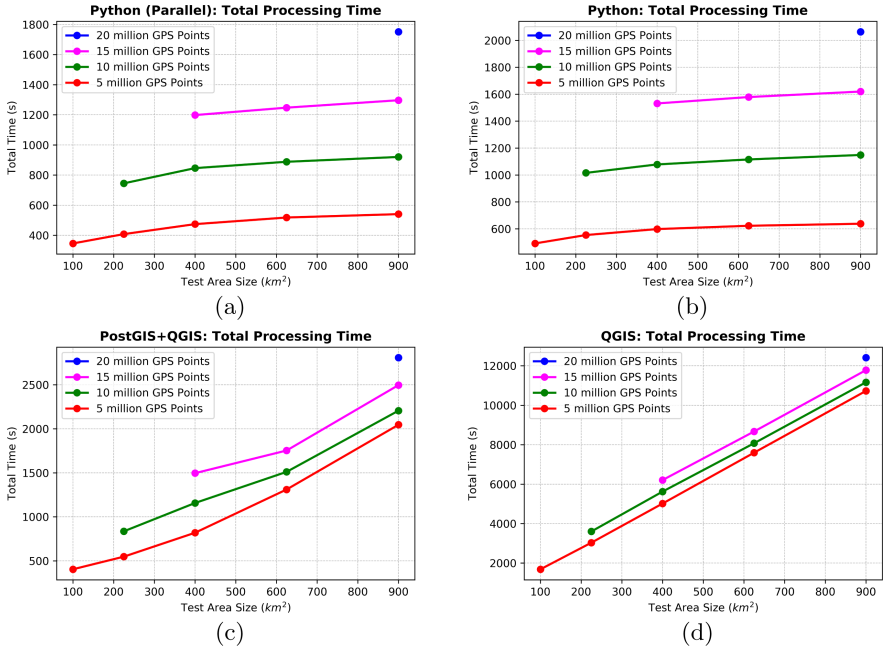

Fig. 7. GPS point count vs. test area size comparison of each method; **(a)** Python (Parallel) **(b)** Python, **(c)** PostGIS+QGIS, **(d)** QGIS.

5 Conclusions

This study evaluates the methods for rasterization of GPS trajectories. Evaluation is carried out for QGIS, PostGIS+QGIS methods and our Python and Python (Parallel) implementations. For evaluation, an experiment was carried out with varying test area and GPS trajectory size. Total processing time and spatial join time were adopted as the evaluation metric.

According to the results, the Python (Parallel) method achieves the best results among the compared methods. The Python method also showed better results with respect to QGIS and PostGIS+QGIS methods. PostGIS+QGIS

method achieves the best result for spatial join. QGIS shows the worst performance for both of the metrics.

Python and Python (Parallel) methods perform slower than PostGIS+QGIS method for spatial join metric. This issue is a result of the time for indexing operation that our implementation spent which is more than the spatial join operations carried out by PostGIS. Indexing operation only dependent to GPS point data size but PostGIS+ QGIS method performance is dependent both to GPS data size and the test are size. Also, when compared to the spent time for grid creation, this delay caused by indexing is negligible. Moreover, indexing operation is more robust than grid creation. On the contrary grid creation consumes too much memory and prone to crashes. Although the PostGIS+QGIS method achieves the best spatial join performance, due to the disadvantage of grid creation and import time required for GPS points, the total performance decreased very fast while the test area size increased. Grid creation can be considered as one-time cost though it is still a disadvantage for the possible cases of different work areas in different applications domains. Similar to the PostGIS+QGIS method, in addition to weak performance of the spatial join, QGIS also performed worse while the test area size increases.

On the other hand, the Python methods' performance is proportional to point count of the GPS trajectories. This feature is proven with additional experiment which has wider test area. As per results, Python methods can work in large areas though QGIS and PostGIS+QGIS methods fail to achieve this. Because the performance is not dependent to test area size and being suitable to parallelization, it is possible to increase performance of Python methods' with distributing computation into more processor cores and/or computation clusters.

As a conclusion, our implementation performs better than QGIS and PostGIS+QGIS methods and can be used for GPS trajectory rasterization. The use of our implementation is not limited to GPS trajectory rasterization. It is also possible to use our implementation in similar problems which require rasterization and aggregation of big point-based datasets into structured grids such as spatial social media data analysis. In addition, the integration of our implementation would increase its usage in other research domains which benefit from QGIS but requiring better performance. It is possible to integrate our implementation scripts into QGIS since support Python programming language, but further research needed to determine if libraries like Swifter are compatible with QGIS Python environment.

References

1. Ahmed, M., Wenk, C.: Constructing street networks from GPS trajectories. In: Epstein, L., Ferragina, P. (eds.) ESA 2012. LNCS, vol. 7501, pp. 60–71. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33090-2_7
2. Carpenter, J.: Swifter: A package which efficiently applies any function to a pandas dataframe or series in the fastest available manner (2020). https://github.com/jmcarpenter2/swifter. Accessed 15 Mar 2020

3. Dabiri, S., Heaslip, K.: Inferring transportation modes from GPS trajectories using a convolutional neural network. Transp. Res. Part C Emerg. Technol. **86**, 360–371 (2018). https://doi.org/10.1016/j.trc.2017.11.021
4. Dask: Dask: Scalable analytics in Python (2020). https://dask.org/. Accessed 15 Mar 2020
5. Eldawy, A., Mokbel, M.F.: SpatialHadoop: a mapreduce framework for spatial data. In: 31st IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea, April 13–17, 2015, pp. 1352–1363 (2015). https://doi.org/10.1109/ICDE.2015.7113382
6. ESRI: GIS Tools for Hadoop: Big Data Spatial Analytics for the Hadoop Framework (2020). http://esri.github.io/gis-tools-for-hadoop/. Accessed 05 May 2020
7. ESRI: Spatial Framework for Hadoop (2020). https://github.com/Esri/spatial-framework-for-hadoop. Accessed 05 May 2020
8. Friedrich, C.: Comparison of ArcGIS and QGIS for applications in sustainable spatial planning. Ph.D. thesis, uniwien (2014)
9. Geopy: Geopy: Python client for several popular geocoding web services (2020). https://geopy.readthedocs.io/en/stable/. Accessed 05 May 2020
10. Gouvernement du Québec: Déplacements MTL Trajet (2018). https://www.donneesquebec.ca/recherche/fr/dataset/vmtl-mtl-trajet. Accessed 15 Mar 2020
11. He, S., et al.: RoadRunner: improving the precision of road network inference from GPS trajectories. In: Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, pp. 3–12. ACM (2018)
12. Institute of Advanced Research in Artificial Intelligence: Traffic4cast Competition Track at NeurIPS 2019 (2020). https://www.iarai.ac.at/traffic4cast/traffic4cast-conference-2019/. Accessed 15 Mar 2020
13. Jiang, X., de Souza, E.N., Pesaranghader, A., Hu, B., Silver, D.L., Matwin, S.: TrajectoryNet: An Embedded GPS Trajectory Representation for Point-based Classification Using Recurrent Neural Networks, pp. 192–200 (2017)
14. Karagiorgou, S., Pfoser, D.: On vehicle tracking data-based road network generation. In: Proceedings of the 20th International Conference on Advances in Geographic Information Systems, pp. 89–98. SIGSPATIAL '12, ACM (2012). https://doi.org/10.1145/2424321.2424334
15. Mnih, V., Hinton, G.E.: Learning to detect roads in high-resolution aerial images. In: Daniilidis, K., Maragos, P., Paragios, N. (eds.) ECCV 2010. LNCS, vol. 6316, pp. 210–223. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15567-3_16
16. Papadomanolaki, M., Vakalopoulou, M., Zagoruyko, S., Karantzalos, K.: Benchmarking deep learning frameworks for the classification of very high resolution satellite multispectral data. ISPRS Ann. Photogrammetry Remote Sens. Spatial Inform. Sci. **3**(7), 83–88 (2016). https://doi.org/10.5194/isprs-annals-III-7-83-2016
17. PostGIS: Spatial and Geographic objects for PostgreSQL (2020). https://postgis.net/. Accessed 15 Mar 2020
18. PostgreSQL: PostgreSQL (2020). https://www.postgresql.org/. Accessed 15 Mar 2020
19. QGIS: A Free and Open Source Geographic Information System (2020). https://qgis.org/. Accessed 15 Mar 2020
20. Sun, T., Di, Z., Che, P., Liu, C., Wang, Y.: Leveraging crowdsourced GPS data for road extraction from aerial imagery. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 7509–7518 (2019)

21. Wang, F., Aji, A., Vo, H.: High performance spatial queries for spatial big data. SIGSPATIAL Special **6**(3), 11–18 (2015). https://doi.org/10.1145/2766196.2766199
22. Zacharatou, E.T., Doraiswamy, H., Ailamaki, A., Silva, C.T., Freire, J.: GPU rasterization for realtime spatial aggregation over arbitrary polygons. Proc. VLDB Endowment **11**(3), 352–365 (2017). https://doi.org/10.14778/3157794.3157803
23. Zhang, J., You, S.: CudaGIS: report on the design and realization of a massive data parallel GIS on GPUs. In: Proceedings of the 3rd ACM SIGSPATIAL International Workshop on GeoStreaming, IWGS 2012, pp. 101–108 (2012). https://doi.org/10.1145/2442968.2442981
24. Zhang, J., You, S., Gruenwald, L.: U2Stra: high-performance data management of ubiquitous urban sensing trajectories on gpgpus. In: International Conference on Information and Knowledge Management, Proceedings, pp. 5–12 (2012). https://doi.org/10.1145/2390226.2390229
25. Zhao, L., Chen, L., Ranjan, R., Choo, K.K.R., He, J.: Geographical information system parallelization for spatial big data processing: a review. Cluster Comput. **19**(1), 139–152 (2016). https://doi.org/10.1007/s10586-015-0512-2
26. Zheng, Y., Zhang, L., Xie, X., Ma, W.Y.: Mining interesting locations and travel sequences from GPS trajectories. In: Proceedings of the 18th International Conference on World Wide Web - WWW '09, pp. 791–800 (2009). https://doi.org/10.1145/1526709.1526816