Deakin University

Programming Paradigms

OnTrack Submission

Project

Submitted By:
Shane Denuka Gurusingha
sgurusingha
2020/10/16 15:14

Tutor:
Mohamed Abdelrazek

Outcome	Weight
Choose	♦♦♦♦♦
Design and Implement	♦♦♦♦♦
Evaluate	♦♦♦♦♦

Implementing MPI for plotting data on maps.

October 16, 2020

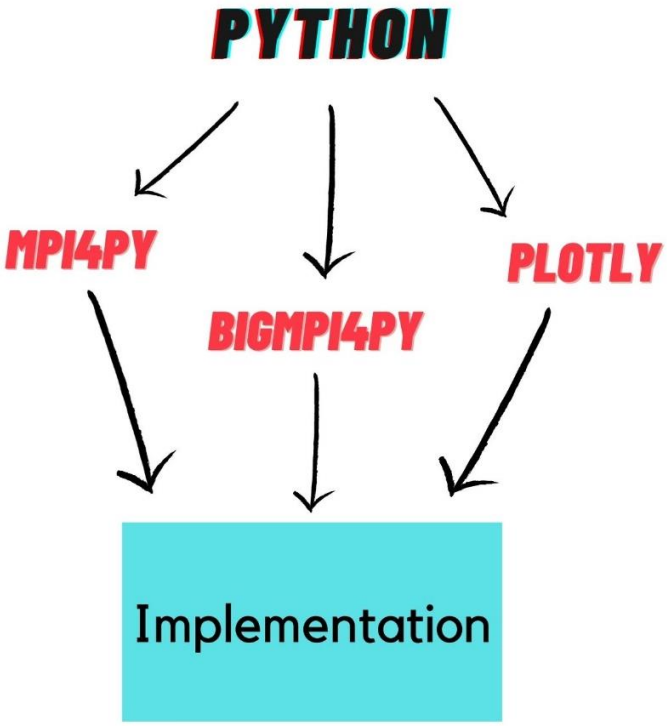
TaskM4.T1D: Project – 218666025

Project scope

For my module 4 task 1 which is the project, I was interested in exploring MPI furthermore, I have often experienced that when plotting data on maps, it becomes quite slow when the data size increases, which eventually leads to issues with overall responsiveness.

To try and tackle this problem, I decided to implement the visualization parallelly using the mpi4py and bigmpi4py which are message passing libraries for python. Finally, I wanted to evaluate its effectiveness by comparing the MPI version and the sequential version.

Design



This project was done using python mainly because of the plotly library which is an interactive open-source graphing library which allowed plotting on beautiful maps.

Moving on, the other two main libraries used was mpi4py and bigmpi4py. Mpi4py is the standard MPI library for python and bigmpi4py was a modified version which allowed to pass pandas dataframe objects. All of these was used in the implementation to create the maps.

Implementation

```
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()

if rank == 0:
    df_main_data = pd.read_csv('data - 204879.csv', low_memory=False)

else:
    df_main_data = pd.DataFrame(
        columns=['Index', 'Distance', 'Latitude', 'Longitude', 'Bearing', 'Elevation (m)', 'Curvature', 'Bend_Label',
                 'City to City', 'lat_lon', 'Sealed?',
                 'IRI', 'City', 'Traffic Light', 'Traffic Light Address', 'Rest Stops', 'Rest Stops Address',
                 'Slope (deg)', 'Gradient_Label',
                 'Speed Limit', 'State', 'elevation_diff'])

df_main_data = BM.scatter(df_main_data, comm, root=0)
```

The above picture displays the part where parallelization begins in the implementation. The *MPI.COMM_WORLD* command initializes the message passing interface. Later, *process 0* reads the data from the file and all the other processes creates an empty dataframe with the columns. Moving on, the *BM.scatter* command from the bigmpi4py library is called which scatters all the data rows equally among the participating processes.

The below image is the second main part of the implementation. After parallelly implementing all the previous steps at the end it was required to bring all the data together so that it can be visualized in one map, instead of creating a map for every process which was participating. For all the scattered data to be in one map I have gathered all the processers data to process 0 and appended it right before the html code was created, so it can successfully visualize the data in one map.

```
def render(self, fig_dict):

    fig_dict_list = comm.gather(fig_dict, root=0)

    if rank == 0:
        for i in range(1, size):
            fig_dict['data'][0]['lat'] = np.append(fig_dict['data'][0]['lat'], fig_dict_list[i]['data'][0]['lat'])
            fig_dict['data'][0]['lon'] = np.append(fig_dict['data'][0]['lon'], fig_dict_list[i]['data'][0]['lon'])

        html = to_html(
            fig_dict,
            config=self.config,
            auto_play=self.auto_play,
            include_plotlyjs=True,
            include_mathjax="cdn",
            post_script=self.post_script,
            full_html=True,
            animation_opts=self.animation_opts,
            default_width="100%",
            default_height="100%",
            validate=False,
        )
        open_html_in_browser(html, self.using, self.new, self.autoraise)
```

Challenges

- Finding a way to pass dataframe objects

Since the standard mpi4py library did not support scattering python dataframe objects, I had to find a way to get it done. After spending some time on researching, I found a modified library of mpi4py which is bigmpi4py, which allowed parallelization for big data objects. Using this library, I was simply able to scatter the dataframe successfully.

- Finding a way to visualize in one map, instead of creating new maps for each process.

```
    fig.update_layout(
        title="Bends in route Categorized into difficulty",
        mapbox_accesstoken=mapbox_access_token,
        mapbox_style="open-street-map",
        margin=dict(t=40, b=10, l=10, r=10),
        font=dict(color='white'),
        paper_bgcolor='#191A1A',
        plot_bgcolor='#191A1A',

    )

fig.show()
```

This was the biggest challenge for me, since I wanted to achieve parallelization and also have all the scattered data in one graph, it was quite tough because the above *fig.show()* was the command that displayed the maps and if it was implemented parallelly it will create new maps for each process participating. After trying to find a solution for a few days and almost giving up, I tried one more time by digging deep into plotlys fig.show() command using the debugger, I found out that there are a few more steps which is happening before the map is actually created. I decided to import those plotly functions to my implementation so that I will be able to perform parallelization until last step of creating the map.

Evaluation

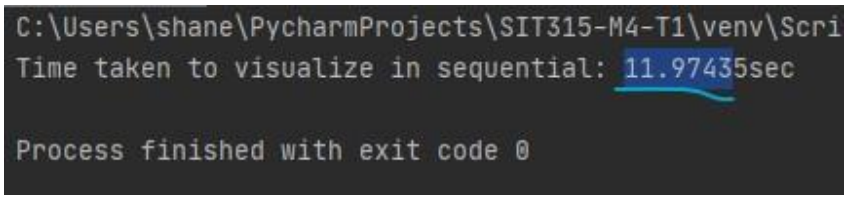
The above graph displays the evaluation between the sequential and mpi version of plotting two maps. In the graph the y-axis is the execution time in seconds and the x-axis is the amount of data which is plotted. In the graph we can clearly see that the mpi version performed better when loading up the maps than the sequential version and the performance of mpi was gradually getting better as the data size increased except for mpi with 2 processers, I believe this was because the amount of data 1 process got was still a bit too high when it was divided by 2. Moreover, it was also seen that running mpi with 4 processors was ideal for this range of data.

The below graphs shows the running times for plotting one map and two maps using plotly.

Evaluation - MPI Plotter VS Sequential Plotter					
Plotting on one mapbox map					
Type	Data				
	204879	254878	304877	353133	572831
Sequential	3.5392	4.09394	4.6773	5.2452	7.159
MPI - 2	2.2155	2.83217	3.0883	3.218	4.8119
MPI - 4	2.5236	2.6076	2.8293	3.0431	4.6785
MPI - 6	2.6636	2.85233	3.2277	3.4306	4.5192
MPI - 10	3.3108	3.5711	4.6218	4.3738	5.47
Plotting on two mapbox maps					
Type	Data				
	204879	254878	304877	353133	572831
Sequential	6.7909	8.5427	10.7891	11.9743	14.3718
MPI - 2	4.0717	5.6346	6.371	6.9051	10.8164
MPI - 4	4.2079	5.144	6.0506	6.316	8.608
MPI - 6	4.0356	5.3723	6.1345	6.3624	9.7106
MPI - 10	5.4995	5.6415765	7.0935	7.3765	9.3938

Running code – few pictures

Sequential – 2 maps – 353133 data rows



MPI – 2 maps – 353133 data rows

```
(venv) C:\Users\shane\PycharmProjects\SIT315-M4-T1>mpiexec -n 2 python mpi-plotter.py
MPI number of processes:  2 | Time taken to visualize:  6.9051117

(venv) C:\Users\shane\PycharmProjects\SIT315-M4-T1>mpiexec -n 4 python mpi-plotter.py
MPI number of processes:  4 | Time taken to visualize:  6.316039699999999

(venv) C:\Users\shane\PycharmProjects\SIT315-M4-T1>mpiexec -n 6 python mpi-plotter.py
MPI number of processes:  6 | Time taken to visualize:  6.3624364

(venv) C:\Users\shane\PycharmProjects\SIT315-M4-T1>mpiexec -n 10 python mpi-plotter.py
MPI number of processes:  10 | Time taken to visualize:  7.376501299999999
```

```
from __future__ import absolute_import, division

import inspect
import json
import os
import textwrap
import uuid
from copy import copy
from distutils.version import LooseVersion
from http.server import BaseHTTPRequestHandler, HTTPServer
from timeit import default_timer as timer

import bigmpi4py as BM
import numpy as np
import pandas as pd
import plotly.express as px
import plotly.graph_objs as go
import plotly.io as pio
import six
from mpi4py import MPI
from plotly import optional_imports
from plotly import utils
from plotly.io._base_renderers import (
    MimetypeRenderer,
    ExternalRenderer,
    PlotlyRenderer,
    NotebookRenderer,
    KaggleRenderer,
    AzureRenderer,
    ColabRenderer,
    JsonRenderer,
    PngRenderer,
    JpegRenderer,
    SvgRenderer,
    PdfRenderer,
    IFrameRenderer,
    SphinxGalleryHtmlRenderer,
    CoCalcRenderer, )
from plotly.io._utils import validate_coerce_fig_to_dict
from plotly.offline.offline import _get_jconfig, get_plotlyjs

ipython = optional_imports.get_module("IPython")
ipython_display = optional_imports.get_module("IPython.display")
nbformat = optional_imports.get_module("nbformat")

mapbox_access_token = "pk.eyJ1Ijoic2hhbmVndXJ1IiwiYSI6ImNrOTN0bzBneTA2MWkzbHFieW5rb
↪ m56b3gifQ.BuGIKlrFRyWSqOzKqL5hJQ"

#mpi initialization
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()

```

```
#process 0 gets data from file
if rank == 0:
    df_main_data = pd.read_csv('data - 204879.csv', low_memory=False)

else:
    df_main_data = pd.DataFrame(
        columns=['Index', 'Distance', 'Latitude', 'Longitude', 'Bearing',
        ↪  'Elevation (m)', 'Curvature', 'Bend_Label',
                 'City to City', 'lat_lon', 'Sealed?',
                 'IRI', 'City', 'Traffic Light', 'Traffic Light Address', 'Rest
                 ↪  Stops', 'Rest Stops Address',
                 'Slope (deg)', 'Gradient_Label',
                 'Speed Limit', 'State', 'elevation_diff'])

#process 0 scatters data to all participating processes
df_main_data = BM.scatter(df_main_data, comm, root=0)

# print("Rank: I got data:", df_main_data, rank)

#creating the map functions

def createSurfaceMap(filter=None):
    if filter is 'Select City':
        filteredCity = df_main_data
    elif filter is 'entire_route':
        filteredCity = df_main_data
    else:
        filteredCity = df_main_data[df_main_data['City to City'] == filter]

    fig = px.scatter_mapbox(filteredCity,
                            range_color=[0.5, 13.5],
                            lat="Latitude",
                            lon="Longitude",
                            hover_data=['IRI'],
                            color="IRI",
                            zoom=3,
                            )

    fig.update_layout(
        mapbox_accesstoken=mapbox_access_token,
        mapbox_style="open-street-map",
        margin=dict(t=40, b=10, l=10, r=10),
        font=dict(color='white'),
        paper_bgcolor='#191A1A',
        plot_bgcolor='#191A1A',
        title='Road surface in the route'
    )

    show(fig, renderer=None, validate=True)


def createBendsMap(filter=None):
    if filter is None:
```

```
        filteredCity = df_main_data
    elif filter is 'entire_route':
        filteredCity = df_main_data
    else:
        filteredCity = df_main_data[df_main_data['Bend_Label'] == filter]

    if (len(filteredCity) == 0):
        fig = go.Figure([go.Scattermapbox(
        )
        ])
        fig.update_layout(
            title="Bends in route Categorized into difficulty",
            mapbox_accesstoken=mapbox_access_token,
            mapbox_style="open-street-map",
            margin=dict(t=40, b=10, l=10, r=10),
            font=dict(color='white'),
            paper_bgcolor='#191A1A',
            plot_bgcolor='#191A1A',
            mapbox=dict(
                center=dict(
                    lat=-25.2744,
                    lon=133.7751
                ),
                zoom=3,
            )

        )

    else:

        fig = px.scatter_mapbox(filteredCity,
                                lat="Latitude",
                                lon="Longitude",
                                hover_data=['Curvature'],
                                color="Bend_Label",
                                zoom=4,
                                )

        fig.update_layout(
            title="Bends in route Categorized into difficulty",
            mapbox_accesstoken=mapbox_access_token,
            mapbox_style="open-street-map",
            margin=dict(t=40, b=10, l=10, r=10),
            font=dict(color='white'),
            paper_bgcolor='#191A1A',
            plot_bgcolor='#191A1A',

        )

        show(fig, renderer=None, validate=True)

"""

```

```
157 Below code is the functions from the Plotly library
158
159 """
160 # Build script to set global PlotlyConfig object. This must execute before
161 # plotly.js is loaded.
162 _window_plotly_config = """\
163 <script type="text/javascript">\
164 window.PlotlyConfig = {MathJaxConfig: 'local'};\
165 </script>"""
166
167 _mathjax_config = """\
168 <script type="text/javascript">\
169 if (window.MathJax) {MathJax.Hub.Config({SVG: {font: "STIX-Web"}});}\
170 </script>"""
171
172
173 def to_html(
174         fig,
175         config=None,
176         auto_play=True,
177         include_plotlyjs=True,
178         include_mathjax=False,
179         post_script=None,
180         full_html=True,
181         animation_opts=None,
182         default_width="100%",
183         default_height="100%",
184         validate=True,
185 ):
186     # ## Validate figure ##
187     fig_dict = validate_coerce_fig_to_dict(fig, validate)
188
189     # ## Generate div id ##
190     plotdivid = str(uuid.uuid4())
191
192     # ## Serialize figure ##
193
194     jdata = json.dumps(
195         fig_dict.get("data", []), cls=utils.PlotlyJSONEncoder, sort_keys=True
196     )
197
198     jlayout = json.dumps(
199         fig_dict.get("layout", {}), cls=utils.PlotlyJSONEncoder, sort_keys=True
200     )
201
202     if fig_dict.get("frames", None):
203         jframes = json.dumps(fig_dict.get("frames", []),
          ↪  cls=utils.PlotlyJSONEncoder)
204     else:
205         jframes = None
206
207     # ## Serialize figure config ##
208     config = _get_jconfig(config)
```

```
209
210     # Set responsive
211     config.setdefault("responsive", True)
212
213     # Get div width/height
214     layout_dict = fig_dict.get("layout", {})
215     template_dict = fig_dict.get("layout", {}).get("template", {}).get("layout", {})
216
217     div_width = layout_dict.get("width", template_dict.get("width", default_width))
218     div_height = layout_dict.get("height", template_dict.get("height",
        ↪ default_height))
219
220     # Add 'px' suffix to numeric widths
221     try:
222         float(div_width)
223     except (ValueError, TypeError):
224         pass
225     else:
226         div_width = str(div_width) + "px"
227
228     try:
229         float(div_height)
230     except (ValueError, TypeError):
231         pass
232     else:
233         div_height = str(div_height) + "px"
234
235     # ## Get platform URL ##
236     if config.get("showLink", False) or config.get("showSendToCloud", False):
237         # Figure is going to include a Chart Studio link or send-to-cloud button,
238         # So we need to configure the PLOTLYENV.BASE_URL property
239         base_url_line = """
240                     window.PLOTLYENV.BASE_URL='{plotly_platform_url}';\
241 """.format(
242             plotly_platform_url=config.get("plotlyServerURL", "https://plot.ly")
243         )
244     else:
245         # Figure is not going to include a Chart Studio link or send-to-cloud
            ↪ button,
246         # In this case we don't want https://plot.ly to show up anywhere in the HTML
247         # output
248         config.pop("plotlyServerURL", None)
249         config.pop("linkText", None)
250         config.pop("showLink", None)
251         base_url_line = ""
252
253     # ## Build script body ##
254     # This is the part that actually calls Plotly.js
255
256     # build post script snippet(s)
257     then_post_script = ""
258     if post_script:
259         if not isinstance(post_script, (list, tuple)):
```

```
            post_script = [post_script]
        for ps in post_script:
            then_post_script += """.then(function(){{
                                {post_script}
                            }})""".format(
                post_script=ps.replace("{plot_id}", plotdivid)
            )

    then_addframes = ""
    then_animate = ""
    if jframes:
        then_addframes = """.then(function(){{
                            Plotly.addFrames('{id}', {frames});
                        }})""".format(
            id=plotdivid, frames=jframes
        )

        if auto_play:
            if animation_opts:
                animation_opts_arg = ", " + json.dumps(animation_opts)
            else:
                animation_opts_arg = ""
            then_animate = """.then(function(){{
                            Plotly.animate('{id}', null{animation_opts});
                        }})""".format(
                id=plotdivid, animation_opts=animation_opts_arg
            )

    # Serialize config dict to JSON
    jconfig = json.dumps(config)

    script = """\
                    if (document.getElementById("{id}")) {{\
                        Plotly.newPlot(\
                            "{id}",\
                            {data},\
                            {layout},\
                            {config}\
                        ){then_addframes}{then_animate}{then_post_script}\
                    }}""".format(
        id=plotdivid,
        data=jdata,
        layout=jlayout,
        config=jconfig,
        then_addframes=then_addframes,
        then_animate=then_animate,
        then_post_script=then_post_script,
    )

    # ## Handle loading/initializing plotly.js ##
    include_plotlyjs_orig = include_plotlyjs
    if isinstance(include_plotlyjs, six.string_types):
        include_plotlyjs = include_plotlyjs.lower()
```

```
313
314     # Start/end of requirejs block (if any)
315     require_start = ""
316     require_end = ""
317
318     # Init and load
319     load_plotlyjs = ""
320
321     # Init plotlyjs. This block needs to run before plotly.js is loaded in
322     # order for MathJax configuration to work properly
323     if include_plotlyjs == "require":
324         require_start = 'require(["plotly"], function(Plotly) {'
325         require_end = "});"
326
327     elif include_plotlyjs == "cdn":
328         load_plotlyjs = """\
329             {win_config}
330             <script src="https://cdn.plot.ly/plotly-latest.min.js"></script>\
331         """.format(
332             win_config=_window_plotly_config
333         )
334
335     elif include_plotlyjs == "directory":
336         load_plotlyjs = """\
337             {win_config}
338             <script src="plotly.min.js"></script>\
339         """.format(
340             win_config=_window_plotly_config
341         )
342
343     elif isinstance(include_plotlyjs, six.string_types) and
     ↪  include_plotlyjs.endswith(
344             ".js"
345     ):
346         load_plotlyjs = """\
347             {win_config}
348             <script src="{url}"></script>\
349         """.format(
350             win_config=_window_plotly_config, url=include_plotlyjs_orig
351         )
352
353     elif include_plotlyjs:
354         load_plotlyjs = """\
355             {win_config}
356             <script type="text/javascript">{plotlyjs}</script>\
357         """.format(
358             win_config=_window_plotly_config, plotlyjs=get_plotlyjs()
359         )
360
361     # ## Handle loading/initializing MathJax ##
362     include_mathjax_orig = include_mathjax
363     if isinstance(include_mathjax, six.string_types):
364         include_mathjax = include_mathjax.lower()
```

```
365
366     mathjax_template = """\
367         <script src="{url}?config=TeX-AMS-MML_SVG"></script>"""
368
369     if include_mathjax == "cdn":
370         mathjax_script = (
371                 mathjax_template.format(
372                     url=(
373                         "https://cdnjs.cloudflare.com"
                             ↪  "/ajax/libs/mathjax/2.7.5/MathJax.js"
374                     )
375                 )
376                 + _mathjax_config
377         )
378
379     elif isinstance(include_mathjax, six.string_types) and include_mathjax.endswith(
380             ".js"
381     ):
382
383         mathjax_script = (
384                 mathjax_template.format(url=include_mathjax_orig) + _mathjax_config
385         )
386     elif not include_mathjax:
387         mathjax_script = ""
388     else:
389         raise ValueError(
390             """\
391 Invalid value of type {typ} received as the include_mathjax argument
392     Received value: {val}
393
394 include_mathjax may be specified as False, 'cdn', or a string ending with '.js'
395     """.format(
396                 typ=type(include_mathjax), val=repr(include_mathjax)
397             )
398         )
399
400     plotly_html_div = """\
401     <div>\
402             {mathjax_script}\
403             {load_plotlyjs}\
404                 <div id="{id}" class="plotly-graph-div" \
405     style="height:{height}; width:{width};"></div>\
406                 <script type="text/javascript">\
407                     {require_start}\
408                         window.PLOTLYENV=window.PLOTLYENV || {{}};{base_url_line}\
409                         {script};\
410                     {require_end}\
411                 </script>\
412             </div>""".format(
413         mathjax_script=mathjax_script,
414         load_plotlyjs=load_plotlyjs,
415         id=plotdivid,
416         width=div_width,
```

```
        height=div_height,
        base_url_line=base_url_line,
        require_start=require_start,
        script=script,
        require_end=require_end,
    ).strip()

    if full_html:
        return """\
    <html>
    <head><meta charset="utf-8" /></head>
    <body>
        {div}
    </body>
    </html>""".format(
            div=plotly_html_div
        )
    else:
        return plotly_html_div


def write_html(
        fig,
        file,
        config=None,
        auto_play=True,
        include_plotlyjs=True,
        include_mathjax=False,
        post_script=None,
        full_html=True,
        animation_opts=None,
        validate=True,
        default_width="100%",
        default_height="100%",
        auto_open=False,
):
    # Build HTML string
    html_str = to_html(
        fig,
        config=config,
        auto_play=auto_play,
        include_plotlyjs=include_plotlyjs,
        include_mathjax=include_mathjax,
        post_script=post_script,
        full_html=full_html,
        animation_opts=animation_opts,
        default_width=default_width,
        default_height=default_height,
        validate=validate,
    )

    # Check if file is a string
    file_is_str = isinstance(file, six.string_types)
```

```
470
471         # Write HTML string
472         if file_is_str:
473             with open(file, "w") as f:
474                 f.write(html_str)
475         else:
476             file.write(html_str)
477
478         # Check if we should copy plotly.min.js to output directory
479         if file_is_str and full_html and include_plotlyjs == "directory":
480             bundle_path = os.path.join(os.path.dirname(file), "plotly.min.js")
481
482             if not os.path.exists(bundle_path):
483                 with open(bundle_path, "w") as f:
484                     f.write(get_plotlyjs())
485
486         # Handle auto_open
487         if file_is_str and full_html and auto_open:
488             url = "file://" + os.path.abspath(file)
489             webbrowser.open(url)
490
491
492 class BaseRenderer(object):
493     """
494     Base class for all renderers
495     """
496
497     def activate(self):
498         pass
499
500     def __repr__(self):
501         try:
502             init_sig = inspect.signature(self.__init__)
503             init_args = list(init_sig.parameters.keys())
504         except AttributeError:
505             # Python 2.7
506             argspec = inspect.getargspec(self.__init__)
507             init_args = [a for a in argspec.args if a != "self"]
508
509         return "{cls}({attrs})\n{doc}".format(
510             cls=self.__class__.__name__,
511             attrs=", ".join("{}={!r}".format(k, self.__dict__[k]) for k in
                ↪  init_args),
512             doc=self.__doc__,
513         )
514
515     def __hash__(self):
516         # Constructor args fully define uniqueness
517         return hash(repr(self))
518
519
520 class HtmlRenderer(MimetypeRenderer):
521     """
```

```
    Base class for all HTML mime type renderers

    mime type: 'text/html'
    """

    def __init__(
            self,
            connected=False,
            full_html=False,
            requirejs=True,
            global_init=False,
            config=None,
            auto_play=False,
            post_script=None,
            animation_opts=None,
    ):

        self.config = dict(config) if config else {}
        self.auto_play = auto_play
        self.connected = connected
        self.global_init = global_init
        self.requirejs = requirejs
        self.full_html = full_html
        self.animation_opts = animation_opts
        self.post_script = post_script

    def to_mimebundle(self, fig_dict):

        from plotly.io import to_html

        if self.requirejs:
            include_plotlyjs = "require"
            include_mathjax = False
        elif self.connected:
            include_plotlyjs = "cdn"
            include_mathjax = "cdn"
        else:
            include_plotlyjs = True
            include_mathjax = "cdn"

        # build post script
        post_script = [
            """
var gd = document.getElementById('{plot_id}');
var x = new MutationObserver(function (mutations, observer) {{
        var display = window.getComputedStyle(gd).display;
        if (!display || display === 'none') {{
            console.log([gd, 'removed!']);
            Plotly.purge(gd);
            observer.disconnect();
        }}
}});

```

```
// Listen for the removal of the full notebook cells
var notebookContainer = gd.closest('#notebook-container');
if (notebookContainer) {{
    x.observe(notebookContainer, {childList: true});
}}

// Listen for the clearing of the current output cell
var outputEl = gd.closest('.output');
if (outputEl) {{
    x.observe(outputEl, {childList: true});
}}
"""
        ]

        # Add user defined post script
        if self.post_script:
            if not isinstance(self.post_script, (list, tuple)):
                post_script.append(self.post_script)
            else:
                post_script.extend(self.post_script)

        html = to_html(
            fig_dict,
            config=self.config,
            auto_play=self.auto_play,
            include_plotlyjs=include_plotlyjs,
            include_mathjax=include_mathjax,
            post_script=post_script,
            full_html=self.full_html,
            animation_opts=self.animation_opts,
            default_width="100%",
            default_height=525,
            validate=False,
        )

        return {"text/html": html}


class ExternalRenderer(BaseRenderer):


    def render(self, fig):
        raise NotImplementedError()


def open_html_in_browser(html, using=None, new=0, autoraise=True):

    if isinstance(html, six.string_types):
        html = html.encode("utf8")

    class OneShotRequestHandler(BaseHTTPRequestHandler):
        def do_GET(self):
            self.send_response(200)
```

```
            self.send_header("Content-type", "text/html")
            self.end_headers()

            bufferSize = 1024 * 1024
            for i in range(0, len(html), bufferSize):
                self.wfile.write(html[i: i + bufferSize])

        def log_message(self, format, *args):
            # Silence stderr logging
            pass

    server = HTTPServer(("127.0.0.1", 0), OneShotRequestHandler)
    webbrowser.get(using).open(
        "http://127.0.0.1:%s" % server.server_port, new=new, autoraise=autoraise
    )

    server.handle_request()


class BrowserRenderer(ExternalRenderer):

    def __init__(
            self,
            config=None,
            auto_play=False,
            using=None,
            new=0,
            autoraise=True,
            post_script=None,
            animation_opts=None,
    ):

        self.config = config
        self.auto_play = auto_play
        self.using = using
        self.new = new
        self.autoraise = autoraise
        self.post_script = post_script
        self.animation_opts = animation_opts

    def render(self, fig_dict):

        fig_dict_list = comm.gather(fig_dict, root=0)

        if rank == 0:
            for i in range(1, size):
                fig_dict['data'][0]['lat'] = np.append(fig_dict['data'][0]['lat'],
                ↪  fig_dict_list[i]['data'][0]['lat'])
                fig_dict['data'][0]['lon'] = np.append(fig_dict['data'][0]['lon'],
                ↪  fig_dict_list[i]['data'][0]['lon'])

            html = to_html(
                fig_dict,
```

```
                config=self.config,
                auto_play=self.auto_play,
                include_plotlyjs=True,
                include_mathjax="cdn",
                post_script=self.post_script,
                full_html=True,
                animation_opts=self.animation_opts,
                default_width="100%",
                default_height="100%",
                validate=False,
            )
            open_html_in_browser(html, self.using, self.new, self.autoraise)


class DatabricksRenderer(ExternalRenderer):
    def __init__(
            self,
            config=None,
            auto_play=False,
            post_script=None,
            animation_opts=None,
            include_plotlyjs="cdn",
    ):

        self.config = config
        self.auto_play = auto_play
        self.post_script = post_script
        self.animation_opts = animation_opts
        self.include_plotlyjs = include_plotlyjs
        self._displayHTML = None

    @property
    def displayHTML(self):
        import inspect

        if self._displayHTML is None:
            for frame in inspect.getouterframes(inspect.currentframe()):
                global_names = set(frame.frame.f_globals)
                # Check for displayHTML plus a few others to reduce chance of a
                 ↪  false
                # hit.
                if all(v in global_names for v in ["displayHTML", "display",
                 ↪  "spark"]):
                    self._displayHTML = frame.frame.f_globals["displayHTML"]
                    break

            if self._displayHTML is None:
                raise EnvironmentError(
                    """
Unable to detect the Databricks displayHTML function. The 'databricks' renderer is
 ↪  only
supported when called from within the Databricks notebook environment."""
                )
```

```
729
730         return self._displayHTML
731
732     def render(self, fig_dict):
733         from plotly.io import to_html
734
735         html = to_html(
736             fig_dict,
737             config=self.config,
738             auto_play=self.auto_play,
739             include_plotlyjs=self.include_plotlyjs,
740             include_mathjax="cdn",
741             post_script=self.post_script,
742             full_html=True,
743             animation_opts=self.animation_opts,
744             default_width="100%",
745             default_height="100%",
746             validate=False,
747         )
748
749         # displayHTML is a Databricks notebook built-in function
750         self.displayHTML(html)
751
752
753 class SphinxGalleryHtmlRenderer(HtmlRenderer):
754     def __init__(
755             self,
756             connected=True,
757             config=None,
758             auto_play=False,
759             post_script=None,
760             animation_opts=None,
761     ):
762         super(SphinxGalleryHtmlRenderer, self).__init__(
763             connected=connected,
764             full_html=False,
765             requirejs=False,
766             global_init=False,
767             config=config,
768             auto_play=auto_play,
769             post_script=post_script,
770             animation_opts=animation_opts,
771         )
772
773     def to_mimebundle(self, fig_dict):
774
775         from plotly.io import to_html
776
777         if self.requirejs:
778             include_plotlyjs = "require"
779             include_mathjax = False
780         elif self.connected:
781             include_plotlyjs = "cdn"
```

```
            include_mathjax = "cdn"
        else:
            include_plotlyjs = True
            include_mathjax = "cdn"

        html = to_html(
            fig_dict,
            config=self.config,
            auto_play=self.auto_play,
            include_plotlyjs=include_plotlyjs,
            include_mathjax=include_mathjax,
            full_html=self.full_html,
            animation_opts=self.animation_opts,
            default_width="100%",
            default_height=525,
            validate=False,
        )

        return {"text/html": html}


class SphinxGalleryOrcaRenderer(ExternalRenderer):
    def render(self, fig_dict):
        stack = inspect.stack()
        # Name of script from which plot function was called is retrieved
        try:
            filename = stack[3].filename  # let's hope this is robust...
        except:  # python 2
            filename = stack[3][1]
        filename_root, _ = os.path.splitext(filename)


class RenderersConfig(object):
    """
    Singleton object containing the current renderer configurations
    """

    def __init__(self):
        self._renderers = {}
        self._default_name = None
        self._default_renderers = []
        self._render_on_display = False
        self._to_activate = []

    # ### Magic methods ###
    # Make this act as a dict of renderers
    def __len__(self):
        return len(self._renderers)

    def __contains__(self, item):
        return item in self._renderers

    def __iter__(self):
```

```
        return iter(self._renderers)

    def __getitem__(self, item):
        renderer = self._renderers[item]
        return renderer

    def __setitem__(self, key, value):
        if not isinstance(value, (MimetypeRenderer, ExternalRenderer)):
            raise ValueError(
                """\
Renderer must be a subclass of MimetypeRenderer or ExternalRenderer.
    Received value with type: {typ}""".format(
                    typ=type(value)
                )
            )

        self._renderers[key] = value

    def __delitem__(self, key):
        # Remove template
        del self._renderers[key]

        # Check if we need to remove it as the default
        if self._default == key:
            self._default = None

    def keys(self):
        return self._renderers.keys()

    def items(self):
        return self._renderers.items()

    def update(self, d={}, **kwargs):

        for k, v in dict(d, **kwargs).items():
            self[k] = v

    # ### Properties ###
    @property
    def default(self):
        """
        The default renderer, or None if no there is no default

        If not None, the default renderer is used to render
        figures when the `plotly.io.show` function is called on a Figure.

        If `plotly.io.renderers.render_on_display` is True, then the default
        renderer will also be used to display Figures automatically when
        displayed in the Jupyter Notebook

        Multiple renderers may be registered by separating their names with
        '+' characters. For example, to specify rendering compatible with
        the classic Jupyter Notebook, JupyterLab, and PDF export:
```

```
888
889             >>> import plotly.io as pio
890             >>> pio.renderers.default = 'notebook+jupyterlab+pdf'
891
892             The names of available renderers may be retrieved with:
893
894             >>> import plotly.io as pio
895             >>> list(pio.renderers)
896
897             Returns
898             -------
899             str
900             """
901             return self._default_name
902
903         @default.setter
904         def default(self, value):
905             # Handle None
906             if not value:
907                 # _default_name should always be a string so we can do
908                 # pio.renderers.default.split('+')
909                 self._default_name = ""
910                 self._default_renderers = []
911                 return
912
913             # Store defaults name and list of renderer(s)
914             renderer_names = self._validate_coerce_renderers(value)
915             self._default_name = value
916             self._default_renderers = [self[name] for name in renderer_names]
917
918             # Register renderers for activation before their next use
919             self._to_activate = list(self._default_renderers)
920
921         @property
922         def render_on_display(self):
923
924             return self._render_on_display
925
926         @render_on_display.setter
927         def render_on_display(self, val):
928             self._render_on_display = bool(val)
929
930         def _activate_pending_renderers(self, cls=object):
931
932             to_activate_with_cls = [
933                 r for r in self._to_activate if cls and isinstance(r, cls)
934             ]
935
936             while to_activate_with_cls:
937                 # Activate renderers from left to right so that right-most
938                 # renderers take precedence
939                 renderer = to_activate_with_cls.pop(0)
940                 renderer.activate()
```

```
941
942         self._to_activate = [
943             r for r in self._to_activate if not (cls and isinstance(r, cls))
944         ]
945
946     def _validate_coerce_renderers(self, renderers_string):
947
948         # Validate value
949         if not isinstance(renderers_string, six.string_types):
950             raise ValueError("Renderer must be specified as a string")
951
952         renderer_names = renderers_string.split("+")
953         invalid = [name for name in renderer_names if name not in self]
954         if invalid:
955             raise ValueError(
956                 """
957 Invalid named renderer(s) received: {}""".format(
958                     str(invalid)
959                 )
960             )
961
962         return renderer_names
963
964     def __repr__(self):
965         return """\
966 Renderers configuration
967 -----------------------
968     Default renderer: {default}
969     Available renderers:
970 {available}
971 """.format(
972             default=repr(self.default), available=self._available_renderers_str()
973         )
974
975     def _available_renderers_str(self):
976
977         available = "\n".join(
978             textwrap.wrap(
979                 repr(list(self)),
980                 width=79 - 8,
981                 initial_indent=" " * 8,
982                 subsequent_indent=" " * 9,
983             )
984         )
985         return available
986
987     def _build_mime_bundle(self, fig_dict, renderers_string=None, **kwargs):
988
989         if renderers_string:
990             renderer_names = self._validate_coerce_renderers(renderers_string)
991             renderers_list = [self[name] for name in renderer_names]
992
993             # Activate these non-default renderers
```

```
            for renderer in renderers_list:
                if isinstance(renderer, MimetypeRenderer):
                    renderer.activate()
        else:
            # Activate any pending default renderers
            self._activate_pending_renderers(cls=MimetypeRenderer)
            renderers_list = self._default_renderers

        bundle = {}
        for renderer in renderers_list:
            if isinstance(renderer, MimetypeRenderer):
                renderer = copy(renderer)
                for k, v in kwargs.items():
                    if hasattr(renderer, k):
                        setattr(renderer, k, v)

                bundle.update(renderer.to_mimebundle(fig_dict))

        return bundle

    def _perform_external_rendering(self, fig_dict, renderers_string=None,
    ↪ **kwargs):

        if renderers_string:
            renderer_names = self._validate_coerce_renderers(renderers_string)
            renderers_list = [self[name] for name in renderer_names]

            # Activate these non-default renderers
            for renderer in renderers_list:
                if isinstance(renderer, ExternalRenderer):
                    renderer.activate()
        else:
            self._activate_pending_renderers(cls=ExternalRenderer)
            renderers_list = self._default_renderers

        for renderer in renderers_list:
            if isinstance(renderer, ExternalRenderer):
                renderer = copy(renderer)
                for k, v in kwargs.items():
                    if hasattr(renderer, k):
                        setattr(renderer, k, v)

                renderer.render(fig_dict)


# Make renderers a singleton object
# ---------------------------------
renderers = RenderersConfig()
del RenderersConfig


def show(fig, renderer=None, validate=True, **kwargs):

```

```
    fig_dict = validate_coerce_fig_to_dict(fig, validate)

    # Mimetype renderers
    bundle = renderers._build_mime_bundle(fig_dict, renderers_string=renderer,
    ↪  **kwargs)
    if bundle:
        if not ipython_display:
            raise ValueError(
                "Mime type rendering requires ipython but it is not installed"
            )

        if not nbformat or LooseVersion(nbformat.__version__) <
        ↪  LooseVersion("4.2.0"):
            raise ValueError(
                "Mime type rendering requires nbformat>=4.2.0 but it is not
                ↪  installed"
            )

        ipython_display.display(bundle, raw=True)
    renderers._perform_external_rendering(fig_dict, renderers_string=renderer,
    ↪  **kwargs)


# Register renderers
# ------------------

# Plotly mime type
plotly_renderer = PlotlyRenderer()
renderers["plotly_mimetype"] = plotly_renderer
renderers["jupyterlab"] = plotly_renderer
renderers["nteract"] = plotly_renderer
renderers["vscode"] = plotly_renderer

# HTML-based
config = {}
renderers["notebook"] = NotebookRenderer(config=config)
renderers["notebook_connected"] = NotebookRenderer(config=config, connected=True)
renderers["kaggle"] = KaggleRenderer(config=config)
renderers["azure"] = AzureRenderer(config=config)
renderers["colab"] = ColabRenderer(config=config)
renderers["cocalc"] = CoCalcRenderer()
renderers["databricks"] = DatabricksRenderer()

# JSON
renderers["json"] = JsonRenderer()

# Static Image
img_kwargs = dict(height=450, width=700)
renderers["png"] = PngRenderer(**img_kwargs)
jpeg_renderer = JpegRenderer(**img_kwargs)
renderers["jpeg"] = jpeg_renderer
renderers["jpg"] = jpeg_renderer
renderers["svg"] = SvgRenderer(**img_kwargs)
```

```
renderers["pdf"] = PdfRenderer(**img_kwargs)

# External
renderers["browser"] = BrowserRenderer(config=config)
renderers["firefox"] = BrowserRenderer(config=config, using="firefox")
renderers["chrome"] = BrowserRenderer(config=config, using="chrome")
renderers["chromium"] = BrowserRenderer(config=config, using="chromium")
renderers["iframe"] = IFrameRenderer(config=config, include_plotlyjs=True)
renderers["iframe_connected"] = IFrameRenderer(config=config,
    include_plotlyjs="cdn")
renderers["sphinx_gallery"] = SphinxGalleryHtmlRenderer()
renderers["sphinx_gallery_png"] = SphinxGalleryOrcaRenderer()

# Set default renderer
# --------------------
# Version 4 renderer configuration
default_renderer = None

# Handle the PLOTLY_RENDERER environment variable
env_renderer = os.environ.get("PLOTLY_RENDERER", None)
if env_renderer:
    try:
        renderers._validate_coerce_renderers(env_renderer)
    except ValueError:
        raise ValueError(
            """
Invalid named renderer(s) specified in the 'PLOTLY_RENDERER'
environment variable: {env_renderer}""".format(
                env_renderer=env_renderer
            )
        )

    default_renderer = env_renderer
elif ipython and ipython.get_ipython():
    # Try to detect environment so that we can enable a useful
    # default renderer
    if not default_renderer:
        try:
            import google.colab

            default_renderer = "colab"
        except ImportError:
            pass

    # Check if we're running in a Kaggle notebook
    if not default_renderer and os.path.exists("/kaggle/input"):
        default_renderer = "kaggle"

    # Check if we're running in an Azure Notebook
    if not default_renderer and "AZURE_NOTEBOOKS_HOST" in os.environ:
        default_renderer = "azure"

    # Check if we're running in VSCode
```

```
    if not default_renderer and "VSCODE_PID" in os.environ:
        default_renderer = "vscode"

    # Check if we're running in nteract
    if not default_renderer and "NTERACT_EXE" in os.environ:
        default_renderer = "nteract"

    # Check if we're running in CoCalc
    if not default_renderer and "COCALC_PROJECT_ID" in os.environ:
        default_renderer = "cocalc"

    if not default_renderer and "DATABRICKS_RUNTIME_VERSION" in os.environ:
        default_renderer = "databricks"

    # Check if we're running in spyder and orca is installed
    if not default_renderer and "SPYDER_ARGS" in os.environ:
        try:
            from plotly.io.orca import validate_executable

            validate_executable()
            default_renderer = "svg"
        except ValueError:
            # orca not found
            pass

    # Check if we're running in ipython terminal
    if not default_renderer and (
            ipython.get_ipython().__class__.__name__ == "TerminalInteractiveShell"
    ):
        default_renderer = "browser"

    # Fallback to renderer combination that will work automatically
    # in the classic notebook (offline), jupyterlab, nteract, vscode, and
    # nbconvert HTML export.
    if not default_renderer:
        default_renderer = "plotly_mimetype+notebook"
else:
    # If ipython isn't available, try to display figures in the default
    # browser
    import webbrowser

    try:
        webbrowser.get()
        default_renderer = "browser"
    except webbrowser.Error:
        # Default browser could not be loaded
        pass

renderers.render_on_display = True
renderers.default = default_renderer

"""

```

```
1200 Creating maps and calculating execution time
1201
1202 """
1203
1204 if rank == 0:
1205     start = timer()
1206
1207 createSurfaceMap(filter='entire_route')
1208 createBendsMap(filter='entire_route')
1209
1210 if rank == 0:
1211     end = timer()
1212     exec = end - start
1213     print("MPI number of processes: ", size, "| Time taken to visualize: ", exec)
```

TaskM4.T1D: Project – 218666025

Demo

https://youtu.be/0KKVhZB6k-o