Parallel routing through GPUs

Cillian Berragan

20th January 2022

GPUs present the opportunity to execute highly parallel processes an order of magnitude faster than traditional CPU bound solutions. While GPUs are already widely used in deep learning applications, more focus is being given to their use in traditional data processing when large amounts of data are involved. This article demonstrates how GPUs may be used to parallelise road network routing using the [cugraph](https://rapids.ai/) library, speeding up analysis times exponentially over the commonly used [networkx](https://networkx.org/) python library.

To test our routing, we will be considering the drive-time distance for a collection of postcodes to their nearest GP practice[[1]](#footnote-22). First some functions and classes are loaded in from our base library named ahah, cudf provides GPU accelerated data frames, and pandas provides normal CPU bound data frames.

import functools  
import time  
from typing import Union  
  
import cudf  
import pandas as pd  
import matplotlib.pyplot as plt  
  
from ahah.common.utils import Config  
from ahah.process\_routing import nearest\_nodes  
from ahah.routing import Routing  
from ahah.routing\_cpu import CPURouting

The road network data and postcodes have already been pre-processed so may be directly loaded into cudf GPU-accelerated data frames through the get\_data function. This function takes in a single postcode, a list, or postcodes matched by a regular expression.

def get\_data(pc: Union[str, list], use\_gpu: bool):  
 edges = cudf.read\_parquet(Config.OS\_GRAPH / "edges.parquet")  
 nodes = cudf.read\_parquet(Config.OS\_GRAPH / "nodes.parquet")  
 postcodes = cudf.read\_parquet(Config.PROCESSED\_DATA / "postcodes.parquet")  
  
 if isinstance(pc, str):  
 postcodes = postcodes[postcodes["postcode"].str.match(pc)]  
 elif isinstance(pc, list):  
 postcodes = postcodes[postcodes["postcode"].isin(pc)]  
  
 gpp = cudf.read\_parquet(Config.PROCESSED\_DATA / "gpp.parquet")  
 gpp = gpp[gpp["postcode"].isin(postcodes["postcode"])]  
 gpp = nearest\_nodes(gpp.reset\_index(drop=True), nodes=nodes)  
  
 return {  
 "name": "gpp",  
 "edges": edges if use\_gpu else edges.to\_pandas(),  
 "nodes": nodes if use\_gpu else nodes.to\_pandas(),  
 "postcodes": postcodes if use\_gpu else postcodes.to\_pandas(),  
 "pois": gpp.to\_pandas(),  
 "buffer": 50\_000,  
 }

The Routing and CPURouting classes iterate over each POI, linked with their nearest road node, calculating the road distance from each postcode node within a 50km buffer, weighted by the estimated road drive-times. The following **‘Setup’** chunk contains two helper functions to take in the data, process this routing, and output the distances and time-taken.

def cpu\_liverpool(\*\*kwargs):  
 routing = CPURouting(\*\*kwargs)  
 t1 = time.time()  
 routing.fit()  
 t2 = time.time()  
  
 postcodes = kwargs.get("postcodes")  
 return routing.distances.set\_index("vertex").join(postcodes.set\_index("node\_id")), t2 - t1  
  
  
def gpu\_liverpool(\*\*kwargs):  
 routing = Routing(\*\*kwargs)  
 t1 = time.time()  
 routing.fit()  
 t2 = time.time()  
  
 postcodes = kwargs.get("postcodes")  
 return routing.distances.set\_index("vertex").join(postcodes.set\_index("node\_id")), t2 - t1

For our analysis we have selected a subset of postcodes for the High Peak Borough, containing 2,861 total postcodes, with an estimated population of around 90,000 people. The following code reads in a table of postcodes within this borough into a python list, which is then processed used to subset the overall postcode data.

hp\_pcs = (  
 pd.read\_csv(  
 "https://www.doogal.co.uk/"   
 "AdministrativeAreasCSV.ashx?district=E07000037"  
 )  
 .loc[lambda row: row["In Use?"] == "Yes", "Postcode"]  
 .tolist()  
)  
  
cpu, cpu\_time = cpu\_liverpool(\*\*get\_data(pc=hp\_pcs, use\_gpu=False))  
gpu, gpu\_time = gpu\_liverpool(\*\*get\_data(pc=hp\_pcs, use\_gpu=True))

The processing times are given below.

print(f"CPU Routing complete in {cpu\_time:.2f} seconds.")  
print(f"GPU Routing complete in {gpu\_time:.2f} seconds.")  
  
print(f"Speedup: {cpu\_time / gpu\_time:.2f}x!")

CPU Routing complete in 66.46 seconds.  
GPU Routing complete in 5.05 seconds.  
Speedup: 13.17x!

It is clear that for this analysis, the total processing time is much slower when CPU bound over using the cugraph GPU library. This difference also extends when applied to larger datasets, as the GPU processing scales exponentially (within memory limits).

[Figure 1](#fig-dists) gives an overview of the output from this analysis. There is a clear correlation between proximity to GP practice and drive-time accessibility, as expected. Due to the mixture of rural and urban settings within the chosen area, there is a relatively large disparity in accessibility between postcodes, while urban centred, dense postcodes have times primarily below 10 minutes, there are a select few remote postcodes with times exceeding 40 minutes. Notably, their relative proximity to GP practices does not account for this large difference, meaning in these remote areas, the road network is responsible for lower accessibility.

gpu = gpu.set\_index("postcode")  
cpu = cpu.set\_index("postcode")  
gpp = get\_data(pc=hp\_pcs, use\_gpu=False).pop("pois")  
  
joined = gpu[['distance']].to\_pandas().join(cpu, lsuffix='\_gpu', rsuffix='\_cpu')  
  
ax = joined.plot(  
 x="easting",  
 y="northing",  
 c="distance\_gpu",  
 kind="scatter"  
)  
gpp.plot(x="easting", y="northing", c="red", kind="scatter", ax=ax)  
ax.get\_xaxis().set\_visible(False)  
ax.get\_yaxis().set\_visible(False)  
  
plt.show()

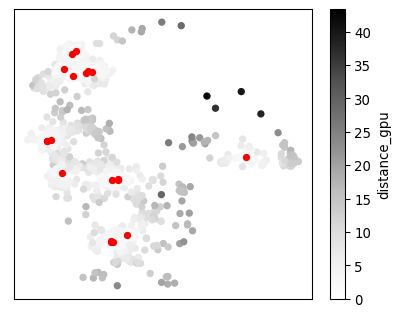


Figure 1: Distance from postcodes to nearest GP practices (red) within High Peak

For peace of mind it is worth considering whether CPU or GPU processing gets different results. The mean difference appears to be around 1.2 seconds (0.02 minutes), with a maximum disagreement of 1.55 minutes.

joined['diff'] = abs(joined['distance\_gpu'] - joined['distance\_cpu'])  
  
print(f'Mean difference: {joined["diff"].mean():.2f}')  
print(f'Max difference: {joined["diff"].max():.2f}')

Mean difference: 0.02  
Max difference: 1.55

1. Please note that only GP practices that fall within our postcode subset are considered, meaning for outer postcodes our estimates may be inaccurate [↑](#footnote-ref-22)