DAT470/DIT065 Assignment 7

Mirco Ghadri
mircog@chalmers.se

June 2, 2024

This assignment will focus on implementing nearest neighbor search algorithm using different CPU based as well as GPU based methods. The GPU based methods will focus on using the CUPY library as well as the Rapids library.

Problem 1

a) See `assignment7_problem1.py` and `nnquery_default.py`.

b) Results:
The result showed that even running GloVe 300d medium on default implementation took a lot longer than expected(22691 seconds $\approx$ 6.3 hours). It also took a lot longer than expected to run GloVe 300d big on the NumPy implementation (31137 seconds $\approx$ 8.6 hours).

For `nnquery_default.py` we did not use batch size.

Table 1: Summary of Query Statistics for nnquery_default.py

Dataset	Query Size	Total Query Time (seconds)	Throughput (queries/second)
Pubs	Tiny	4.277	2.34
Pubs	Small	42.234	2.37
Pubs	Medium	437.482	2.28
Pubs	Big	4449.243	2.25
GloVe 50d	Tiny	35.822	0.28
GloVe 50d	Small	371.184	0.27
GloVe 50d	Medium	3794.955	0.26
GloVe 300d	Tiny	226.308	0.04
GloVe 300d	Small	2471.628	0.04
GloVe 300d	Medium	22691.8	0.044

For `assginemnt7_problem1.py` we were forced to process data in smaller batch sizes on certain query/dataset combinations. We only used smaller batch size on those query/dataset combinations were the program would run out of memory without splitting and processing data in smaller chunks. For the rest of the queries, we just used a batch size as large as the query set which would do the same thing as processing all the data at once. From the table of results, we can see that the largest batch size we were able to use was 100.

Table 2: Summary of Query Statistics for assignment7_problem1.py

Dataset	Query Size	Total Query Time (seconds)	Throughput (queries/second)	Batch Size
Pubs	Tiny	0.0911	109.78	10
Pubs	Small	0.7894	126.70	100
Pubs	Medium	7.9685	125.52	100
Pubs	Big	76.2957	131.05	100
GloVe 50d	Tiny	1.5852	6.31	10
GloVe 50d	Small	15.6405	6.39	100
GloVe 50d	Medium	155.9290	6.41	100
GloVe 50d	Big	1894.8280	5.28	100
GloVe 300d	Tiny	32.5040	0.31	1
GloVe 300d	Small	413.2502	0.24	10
GloVe 300d	Medium	3554.7272	0.28	10
GloVe 300d	Big	31136.96	0.321	10

From the table of results, we can see that the throughput was the highest for Pubs queries. This makes sense since those queries have the smallest dimension (only a dimension of 2). We then see a significant drop in throughput for the queries on the GloVe 50d dataset. This makes sense since those queries have a much higher dimension of 50. So processing an individual query and comparing it to an individual item in the data is more computationally heavy. We can see the slowest throughput for GloVe 300d since that data has the highest dimensionality.

Problem 2

a) See `assignment7_problem2.py`

b) Results: A result which does not make sense here is that we needed to use a batch size of 10 for GloVe 50d medium, but we could run the algorithm fine with a batch size of 100 for GloVe 50d small. If we tried to run GloVe 50d medium with batch size of 100, it would run out of memory.This made us wonder if we had implemented the batch algorithm incorrectly.

Table 3: Summary of Query Statistics for `assignment7_problem2.py`

Dataset	Query Size	Total Query Time (seconds)	Throughput (queries/second)	Batch Size
Pubs	Tiny	1.056	9.47	10
Pubs	Small	1.067	93.83	100
Pubs	Medium	1.165	857.90	100
Pubs	Big	1.681	5951.75	100
GloVe 50d	Tiny	1.163	8.60	10
GloVe 50d	Small	1.270	78.74	100
GloVe 50d	Medium	2.243	446.12	10
GloVe 50d	Big	12.082	827.72	10
GloVe 300d	Tiny	4.219	2.37	1
GloVe 300d	Small	6.845	14.59	1
GloVe 300d	Medium	33.200	30.12	1
GloVe 300d	Big	297.069	33.73	1

Problem 3

a) See `assignment7_problem3.py`

b) Results:
Here, we did not use batch size to split the query data into smaller chunks since it was not necessary. We can see that matrix operations indeed are less heavy on computer memory than corresponding NumPy array operations. The reason is that we were forced to split and process the data in smaller chunks when processing the same query-/dataset combination using NumPy in problem 1.

Table 4: Summary of Query Statistics for `assignment7_problem3.py`

Dataset	Query Size	Total Query Time (seconds)	Throughput (queries/second)	Erroneous Queries	Batch Size
Pubs	Tiny	0.031	322.58	1	10
Pubs	Small	0.067	1492.54	18	100
Pubs	Medium	0.567	1763.36	145	1000
Pubs	Big	5.849	1708.01	1657	10000
GloVe 50d	Tiny	0.128	78.13	0	10
GloVe 50d	Small	0.826	120.98	0	100
GloVe 50d	Medium	7.134	140.10	0	1000
GloVe 50d	Big	70.474	141.93	0	10000
GloVe 300d	Tiny	2.118	4.72	0	10
GloVe 300d	Small	7.111	14.08	0	100
GloVe 300d	Medium	49.044	20.39	0	1000
GloVe 300d	Big	829.606	12.06	0	10000

Problem 4

a) See `assignment7_problem4.py`

b) Results We had to use batch size for our calculation using matrix operations on the GPU. Otherwise the resulting matrix would be too large to be stored in GPU memory. The GPU had less memory than the CPU. However, we could use larger batch sizes for the same datasets compared to the same code running on Numpy (see problem 2), so this proved that Matrix operations are more memory efficient.

Table 5: Summary of Query Statistics for `assignment7_problem4.py`

Dataset	Query Size	Total Query Time (seconds)	Throughput (queries/second)	Erroneous Queries	Batch Size
Pubs	Tiny	1.144	8.73	1	10
Pubs	Small	1.141	87.64	18	100
Pubs	Medium	1.142	875.70	145	1000
Pubs	Big	1.166	8568.04	1657	10000
GloVe 50d	Tiny	1.257	7.96	0	10
GloVe 50d	Small	1.273	78.57	0	100
GloVe 50d	Medium	1.251	798.56	0	1000
GloVe 50d	Big	1.553	6437.99	0	1000
GloVe 300d	Tiny	3.926	2.55	0	10
GloVe 300d	Small	3.999	25.00	0	100
GloVe 300d	Medium	4.698	212.87	0	100
GloVe 300d	Big	11.872	842.29	0	100

Problem 5

a) See `assignment7_problem5.py`

b) Results:
Here, we did not implement batch processing at all, and simply fed all the data to the knn function of Rapids. It worked fine and did not run out of memory like in problem 4 or problem 2.

Table 6: Summary of Query Statistics for `assignment7_problem5.py`

Dataset	Query Size	Query Time (seconds)	Throughput (queries/second)	Number of Erroneous Queries
Pubs	Tiny	0.9919	10.11	0
Pubs	Small	0.9521	105.05	20
Pubs	Medium	0.9558	1048.08	160
Pubs	Big	0.9548	10495.48	1731
GloVe 50d	Tiny	1.0465	9.56	0
GloVe 50d	Small	1.0492	95.36	0
GloVe 50d	Medium	1.0587	943.71	0
GloVe 50d	Big	1.1510	8692.88	0
GloVe 300d	Tiny	3.7920	2.64	0
GloVe 300d	Small	3.8089	26.27	0
GloVe 300d	Medium	4.0895	244.37	0
GloVe 300d	Big	6.3702	1568.72	0

Appendix

For the shellcode that was used to submit all slurm jobs, see `all_jobs.sh`.