Written Reexam in HPPS—Practical Part

April 21—23, 2021

Preamble

This is the problem text practical part of the HPPS exam. This document consists of ii pages excluding this preamble; make sure you have them all. See the preamble of the theory part for general information. Your solution of this practical part is expected to consist of a *short* report in PDF format, *separate* from your solution of the theory part, as well as a `.zip` or `.tar.gz` archive containing your source code. That is, you are expected to upload *three* files in total for the entire exam. The report is not expected to be as polished as in the assignments. Focus on concrete details and numeric results.

1 Background

k-means clustering is an unsupervised learning technique that partitions n points in some d-dimensional space into k clusters, such that each point belongs to the cluster with the nearest *centroid* (computed as the mean of all points beloning to that cluster). This is a difficult (NP-hard) optimisation problem, but in practice there are simple heuristic algorithms that work well. You will be working with one such algorithm. Generally, k and d are fairly small, but n can be very large. As an example, $k = 5, d = 35, n = 494019$ is a plausible set of parameters.

1.1 k-means clustering via Lloyd's algorithm

Given an initial set of k centroids, we repeatedly perform two steps:

Assign points to clusters Each point is assigned to the cluster that has the closest centroid (computed with the Euclidean distance).

Recompute centroids The centroid for each of the k clsuters is recomputed as the mean of all the points beloning the cluster.

We stop when assignments of points to clusters no longer change, or after a specified iteration limit. For this exam, we arbitrarily set the iteration limit at 500.

The initial setting of centroids has implications for how long the algorithm takes to converge. For this exam, we just take the initial centroids to be the first k points. See the Wikipedia page[1] if you need more information, and for a visualisation of the approach.

2 Your task

You are given a working Python implementation of k-means clustering, `kmeans.py`. It is functionally correct, but very slow. Your task is to write a corresponding C program and to parallelise it.

The program reads the points from textual data files, where the first line contains integers for n and d, and the remaining n lines contain d numbers each, representing n points with d components each. See for example `n100_d2.txt` in the handout. Code for reading and writing these files is part of the handout.

We can run the Python program as follows:

[1] `https://en.wikipedia.org/wiki/K-means_clustering`

```
$ python3 kmeans.py 3 n100_d2.txt
Round   0.
Round   1.
Round   2.
Round   3.
Round   4.
Centroids:
31.731296 23.725892
58.151539 79.018219
84.318027 36.937313
```

It prints out a line for each round, and when it finishes, the computed centroids. To be practically useful, we should also print out the actual assignments of points to clusters, but that quickly becomes very verbose for high n, and the centroids are enough to gauge whether your C implementation is correct.

3 Task specifics

You are expected to complete the following tasks, documented in a short report.

3.1 Implement k-means in C

The code handout contains a skeleton implementation `kmeans.c`. It can read data files, but the clustering itself is incomplete. The implementation makes use of the file `util.h`, which you do *not* need to modify. **Hint:** the C equivalent of `np.inf` is called `INFINITY`.

Your tasks for 3.1 are:

- Finish the C-based implementation.
- Argue for why you believe your implementation is correct.
- Measure the performance of your implementation for various k, d, and n (it is fine to fix k and d and only vary n).
- Measure the performance of the Python implementation for the same inputs.
- Include the results in your hand-in.

3.2 Estimate scalability

Estimate the parallel fraction p of the k-means simulation (either the handed out Python code or your own C version).

Your tasks for 3.2 are:

- Provide your estimate of the parallel fraction of the k-means simulation.
- Provide a brief description of how you arrived at this fraction.

3.3 Parallelise your k-means implementation

Use what you have learned in the course to make your implementation run faster. Make sure to submit both your original implementation as well as any optimised ones. In particular, parallelise the simulation with OpenMP. Compare your scaling results with what might have been expected from Amdahl's and/or Gustafson's laws, based on your estimate of p.

Your tasks for 3.3 are:

- Parallelise your C implementation of k-means.
- Compute the speedup of your optimized program, compared to your original C-based implementation.

4 The code handout

`kmeans.py`: A functionally correct implementation of k-means clustering in Python.

`kmeans.c`: The start of k-means clustering in C. Same interface as `kmeans.py`, only the input/output parts are implemented.

`util.h`: Various utility definitions used by `kmeans.c`. You do not need to modify this file, but you are allowed to.

`genpoints.c`: A C program that generates random data files. You don't need to modify or understand this program.

`Makefile`: Contains one useful target:

`make`
: Compiles `genpoints` and `kmeans`. Add more programs to the `EXECUTABLES` variable in `Makefile` to compile these as well.