Assignment 1: Report
Virginia Commonwealth University, Fall 2018
Sina Ghadermarzi

Method

Serial code: The KNN algorithm implemented works by finding the K nearest element in the dataset and choosing the class that has the majority on these K neighbors.

- The main loop iterates over different objects (instances) and produces the KNN prediction for each of them.
- The next inner loop computes the Euclidean distance between current instance and all other instances and stores them in an array (named "distances").
- Then we find the K smallest element in this array and for each of them increase the count of their corresponding class by one
- We store the class that has the highest count as the prediction for the current instance

The serial code is in "main_serial.cpp" and the parameter K is defined by the line "#define KNN_K 4"

Parallel Code: The program is parallelized using pthread by breaking the outer (main) loop which iterates over objects (instances) and dividing the work among the given numbers of threads. In the main part of the parallel code the threads are created in a for-loop, with each of them being assigned a subset of instances and all the created threads are joined in the next for-loop. This program is in a file named "main_parallel.cpp" and the number of threads is specified by the line "#define NUM_THREADS 4" in this file.

Results

These programs are tested on a virtual machine system and the maple server and accuracy for small dataset is 78% and for medium dataset is 49%. The runtimes (in milliseconds) are listed in the table below with their corresponding speedup in a parenthesis in front of them:

	# of threads	Serial	1	2	4	8	2048
Virtual Machine	Small	36	35 (1.02)	26 (1.38)	20 (1.8)	30 (1.2)	556 (0.06)
	Medium	12810	13342 (0.96)	6273 (2.04)	4212 (3.04)	4798 (2.67)	
Maple Server	Small	61	64 (0.95)	37 (1.64)	20 (3.05)	11 (5.5)	128 (0.47)
	Medium	9395	9589 (0.98)	4863 (1.93)	2529 (3.71)	1378 (6.96)	1995 (4.70)

Conclusion

The results show that on virtual machine, which can only use 4 cores the speedup is observed for up to 4 threads but after that the overhead leads to increase in runtime. On Maple server which has more number of cores this speedup is observed for up to 8 threads but with 2048 thread again the overhead of creating threads increases and leads to an increase in runtime.

Now we compute the parallel and serial portion of the program by seeing the increase in speedup with respect to the number of threads (processing units). We base our calculations on the scenario with maple server and medium dataset. Since the runtimes of this setting in more reliable (the numbers are not too small) and also we have more available cores.

The formula for speedup is as follows:

$$Speedup = \frac{1}{\left(s + \frac{p}{n}\right)} = \frac{1}{\left(s + \frac{p}{8}\right)} = 6.96$$

$$s + 0.125p = 0.144$$

$$s + p = 1$$

$$s = 0.02$$

$$p = 0.98$$

As we see a large part of the program is parallel. This is because we divide the whole work into parts which are independent (embarrassingly parallel) and the only part which is not parallel is creation of threads and creating reading the dataset.