CMSC 691 Assignment 1

Basic idea

1. Thread partition

I divide the dataset into N_t part, N_t is the number of threads, each thread take charge different part of dataset, predict the class and store it into the corresponding position of prediction matrix.

Suppose the number of the instances in dataset is N_i , so the number of elements which are processed by each thread is $N_e = \frac{N_i}{N_t}$. If N_i is divisible by N_t , so N_e should be equal for each thread. However if N_i is not divisible by N_t , we suppose $quotient = N_i / N_t$, $reminder = N_i \% N_t$, for first $reminder$ threads, $N_e = quotient + 1$, for left threads, $N_e = quotient$.

For example, for the *small.arff* dataset, the number of instances is 336, if we want to divide it into ten threads, the elements of each threads are as Table 1:

Table 1: An example of thread partition for 10 threads and 336 instances

Thread #	Instances index	Number of elements
0	0-34	34
1	34-68	34
2	68-102	34
3	102-136	34
4	136-170	34
5	170-204	34
6	204-237	33
7	237-270	33
8	270-303	33
9	303-336	33

2. Parallel design

In my program, there are two data which need access by threads, one is the dataset, another is the prediction matrix. Dataset is the resource which are raced by all threads, and the prediction matrix is the resource which should be exclusively accessed by each thread, we should protect it when we want to change the value. We use mutex to protect the matrix.

Experiment statistics:

The following table lists the experiment collection, all the data are the average value of five experiments.

Table 2: Experiment statistics

		parallel(ms)			
small	sequential(ms)	2 threads	4 threads	8 threads	256 threads
	23.2	21	20.8	18.8	14.8
	speedup	1.10	1.12	1.23	1.57
medium	sequential(ms)	parallel(ms)			
		2 threads	4 threads	8 threads	255 threads
	8567.2	7303.8	4138.2	3646.8	3510
3510	Speedup	1.17	2.07	2.35	2.44

Figre 1 is the performance speedup for different number of threads. With the same dataset, the performance can benefit from more threads. When we assign the same number of threads for each dataset, medium size data set gain more significant performance speedup than small size, the dataset can be processed in a parallel manner.

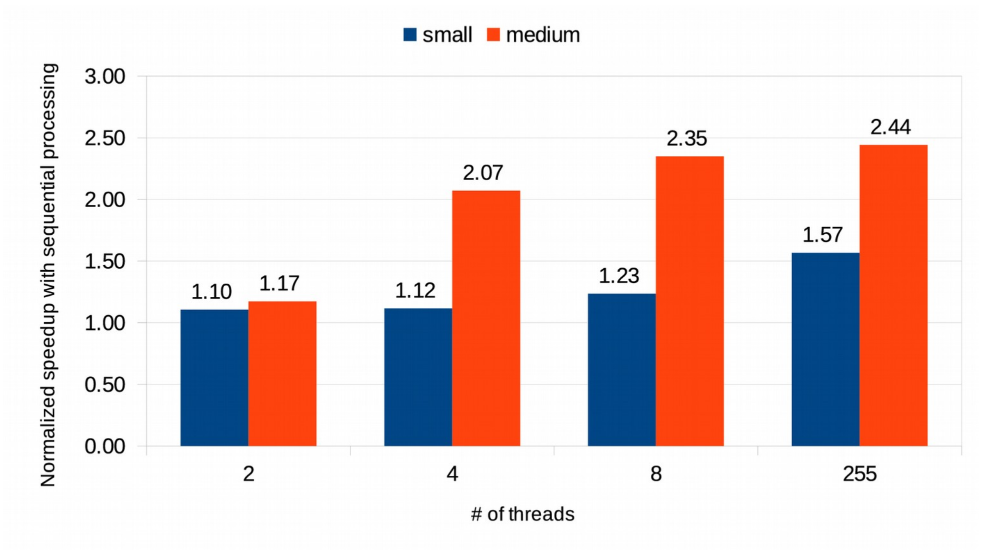

Figure 1: Performance speedup for different number of threads

If there are 255 threads, the performance do not behave as well as we expecte since the system has to afford the cost of the thread creation, switch, and destroy, however, each thread take over less work.

Analysis:

Q2: For 2 threads, the speedup is 1.10, according to Amdahl`s law, suppose the parallel part take p, so : $1.10=\frac{1}{(1-P+\frac{P}{2})}, P=0.2$

suppose we can launch unlimited threads, $N \to \infty$, so the most speedup is

$$\frac{1}{(1-0.2+\frac{P}{\infty})}=1.25$$