Assignment 3 CS4745/6025 Fall 2017

Due: 9:00 am Friday October 27 electronically in Desire2Learn

1. **Tweet Complexity**. Consider the second decomposition of the histogram evaluation in Figure 3.1b. Coarsen the decomposition into fewer tasks and implement it using an OpenMP parallel loop in two different ways:

   a. All threads write to the same histogram array, which is protected using a critical section
   b. Each thread writes to a separate histogram array, then these arrays are added together, again using a critical section.

   You will modify the `tweetHistogram.c` program, and use the provided file of tweets (`training_set_tweets.txt`, obtained from a September 2009 - January 2010 twitter scrape) to report speedup for each version, for 2, 4, 8 threads. Each runtime value should be determined from the average of 5 runs.

2. **SPMD prefix sum**. Algorithm 5.1 presents an inclusive prefix sum of `n` values by `p` threads, where `n` is divisible by `p`. Note the error in the line just before the call to `scan(b)`, and also the need to place a barrier before this call (see the slides shown in class). Your task is to implement this algorithm for an *exclusive* sum for arbitrary values of `n`, in the context of a program that performs decompression using run-length decoding (see Fig. 3.9).

   The `rld.c` program generates a random sequence of repeats (assuming that the first sequence of pixels is white (0)), and writes an uncompressed sequence to an array using a parallel OpenMP loop and a prefix sum. Your task is to implement the following function:

   ```
   void prefixSumCoarse(int *a, int *as, int *b, int *bc, int n, int id, int nt);
   ```

   which performs the parallel exclusive prefix sum of array `a` (without modifying `a`), of length `n`, which is written to the `as` array, using an algorithm based on Algorithm 5.1 (as described above). It requires two working arrays (`b` and `bc`) of size `nt`. An important feature of this function is that it must terminate with a barrier.

   Experimentally determine the speedup for 2, 4, 8 threads, for 3 different problem sizes (i.e., 3 speedup curves). Each runtime value should be determined from the average of 5 runs.

What to submit: a pdf report with code listings and experimental results and a zip file with both source files.