Assignment 2a: "Histogram w/ random numbers"

In this part of the assignment (2a), you are given a C++ program that records the occurrences of 30000000 randomly generated numbers in a histogram. This histogram contains 10 bins, and the generated values range from 0 to 9. As the program generates random values within this range, it counts the occurrences of each value by incrementing the corresponding bin of the histogram (see **Example**).

Example: if the random number generator produces the value 5, then the current value of the bin number 5 of the histogram should be incremented.

Requirements summary:

- Use **OpenMP** to implement 2 parallel versions of the histogram program (**histogram-v1.cpp**).
- The first parallel version (**histogram-v1-naive.cpp**) should use the data-scoping and synchronization mechanisms provided by OpenMP to ensure the correctness of your parallel solution.
- The second version (**histogram-v1-best.cpp**) should be your best-effort implementation, and it must minimize synchronization contention as much as possible to achieve the desired speedup (**see below**).
- The sum of the elements inside each bin of the histogram needs to add up to the number of random numbers generated. If this is not verified, your parallel implementation is not correct.
- Your best-effort implementation should be correct and achieve a speedup of at least **15x** (for **32 threads on ALMA**) compared to the sequential version of the code (**histogram-v1.cpp**).
- The desired speedup can be achieved just by including OpenMP directives. Therefore, avoid unnecessary algorithmic optimizations.
- Keep your code flexible by avoiding OpenMP run-time routines and hard-coded clauses. Resort to environment variables whenever possible. E.g., use the OMP_NUM_THREADS environment variable in place of the `num_threads` clause or the `omp_set_num_threads` routine.

Detailed Description

1. Naive Histogram **[Implement in histogram-v1-naive.cpp]**

At this stage, your goal is to parallelize the populate method of the histogram structure. The populate method already generates the random values and increments the bins of the histogram accordingly (in a sequential manner). To parallelize this method, distribute the workload among a team of OpenMP threads, and use the synchronization and data-scoping attributes provided by OpenMP to avoid data-races and ensure the correction of your solution.

2. Best Effort Histogram **[Implement in histogram-v1-best.cpp]**

Implement the same functionality as above, but avoid synchronization contention as much as possible to achieve the desired speedup. This version should achieve at least a speedup of 15x on a node of the ALMA cluster (for **32 threads**), while remaining correct.

Hint: the `default(none)` clause of the parallel construct might be helpful here.

Challenge (optional): Try to implement your best-effort version by including a single (yet long) OpenMP compiler directive. This challenge is entirely optional.

3. Compiling and Running your code

To compile at home use:

```
g++ -o <executable_name> --std=c++20 -fopenmp -O2 <cpp_file_name>.cpp
```

To compile on ALMA:

```
/opt/global/gcc-11.2.0/bin/g++ -o <executable_name> --std=c++20 -fopenmp -O2
<cpp_file_name>.cpp
```

To run on ALMA use (example for 32 threads):

- `OMP_NUM_THREADS=32 srun --nodes=1 ./<executable_name>`

The expected output is the following:

```
Bins: 10, sample size: 30000000
0:3001319
1:2998561
2:3002571
3:2998314
4:3000574
5:3000892
6:2997455
7:3001024
8:3001479
9:2997811
total: 30000000

time elapsed: 0.819826 seconds
```

Attention: The contents of each bin **may differ** since the samples are **randomly generated**!

4. Submission Guidelines

The program has to be submitted before the deadline on the online platform after it has passed all checks. You also need to run your code on ALMA to measure the speedup achieved.

The required speedup on ALMA is at least ~15x for 32 threads, and you should measure the speedup of your best effort solution for **2**, **4**, **8**, **16**, and **32 threads** (not on the front-end, but by using the **srun** command as shown above). The sequential execution time on ALMA is around **0.83** seconds. Once you have the results, enter them on the online platform. Make sure that both the code and speedup graph are present on the online platform. The online platform should not be used for speedup measurements.

Deadline: Deadline: June 22, 2025 until 23:59 on the online platform.