CME213, Spring 2022
Parallel Computing
Julian Cooper

Homework 2
Due Friday, April 22nd via GradeScope

Problem 1: Implement a parallel function that sums separately the odd and even values of a vector.

Idea: Need to implement `parallelSum` using `omp parallel for` with reductions for both even and odd accumulators.

```cpp
std::vector<uint> parallelSum(const std::vector<uint> &v)
{
    omp_set_num_threads(4);
    std::vector<uint> sums(2);
    uint sum0 = 0; uint sum1 = 0;

    #pragma omp parallel for reduction(+:sum0) reduction(+:sum1)
    for(uint i=0; i<v.size(); i++) {
        if (v[i] % 2 == 0) {
            sum0 += v[i];
        }
        else {
            sum1 += v[i];
        }
    }
    sums[0] = sum0; sums[1] = sum1;
    return sums;
}
```

Console logs from `main_q1.cpp`.

```
$ make main_q1
g++ -std=c++11 -g -Wall -O3 -fopenmp main_q1.cpp -o main_q1

$ ./main_q1
Parallel
Sum Even: 757361650
Sum Odd: 742539102
Time: 0.00433168
Serial
Sum Even: 757361650
Sum Odd: 742539102
Time: 0.106256
main_q1.cpp:60:main     TEST PASSED.
```

Problem 2: Implement Radix Sort in parallel ...

- **Question 1: computeBlockHistograms()** Idea: using openMP to parallelize computation across "blocks" when creaing local histograms. Code must pass Test1().

```
$ make main_q2
g++ -std=c++11 -g -Wall -O3 -fopenmp main_q2.cpp -o main_q2

$ ./main_q2
tests_q2.h:22:Test1     TEST PASSED.
```

- **Question 2: reduceLocalHistoToGlobal()** Idea: accumulate values based on the remainder of the index divided by bucketSize. Code must pass Test2().

```
$ make main_q2
g++ -std=c++11 -g -Wall -O3 -fopenmp main_q2.cpp -o main_q2

$ ./main_q2
tests_q2.h:38:Test2     TEST PASSED.
```

- **Question 3: scanGlobalHisto()** Idea: implement cumulative sum using std::partial_sum standard library function. Note, needed to adjust Output Iterator and Input Iterator to ensure we begin at zero and do not inadvertedly overflow.

```
$ make main_q2
g++ -std=c++11 -g -Wall -O3 -fopenmp main_q2.cpp -o main_q2

$ ./main_q2
tests_q2.h:50:Test3     TEST PASSED.
```

- **Question 4: computeBlockExScanFromGlobalHisto()** Idea: populate first using globalHistoScan and then increment using blockHisograms for subseuqent blocks. This has the effect of splitting the global histogram into blocks need to update our sorting algorithm (next step).

```
$ make main_q2
g++ -std=c++11 -g -Wall -O3 -fopenmp main_q2.cpp -o main_q2

$ ./main_q2
tests_q2.h:67:Test4     TEST PASSED.
```

- **Question 5: populateOutputFromBlockExScan()** Idea: use pre-computed `blockExScan` to help target where entries of our unsorted input vector should map to in `sorted`. We can parallelize this operation by block using `openMP`. Note, we still need to re-compute which "bucket" each of our unsorted entries are from at each step since this information is not stored and passed to the function.

  Also, this step only succeeds in sorting our input up to the `numBits`'th bit (in this case 8 bits of sorting per pass). Subsequent "passes" are needed to complete our radix sort algorithm since many input entires are greater than 256 (limit of 8 bits).

  ```
  $ make main_q2
  g++ -std=c++11 -g -Wall -O3 -fopenmp main_q2.cpp -o main_q2

  $ ./main_q2
  tests_q2.h:84:Test5     TEST PASSED.
  ```

Submission information logs.

```
$ /afs/ir.stanford.edu/class/cme213/script/submit.py hw1 private/cme213-jelc53/hw1
Submission for assignment 'hw1' as user 'jelc'
Attempt 1/10
Time stamp: 2022-04-01 20:53
List of files being copied:
    private/cme213-jelc53/hw1/main_q1.cpp  [3875 bytes]
    private/cme213-jelc53/hw1/main_q2.cpp  [1213 bytes]
    private/cme213-jelc53/hw1/main_q3.cpp  [1362 bytes]
    private/cme213-jelc53/hw1/main_q4.cpp  [5117 bytes]
    private/cme213-jelc53/hw1/matrix.hpp  [3036 bytes]

Your files were copied successfully.
Directory where files were copied: /afs/ir.stanford.edu/class/cme213/submissions/hw1/jel
List of files in this directory:
    main_q1.cpp  [3875 bytes]
    main_q2.cpp  [1213 bytes]
    main_q3.cpp  [1362 bytes]
    main_q4.cpp  [5117 bytes]
    matrix.hpp  [3036 bytes]
    metadata  [137 bytes]

This completes the submission process. Thank you!
```