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.

```
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 algorithm 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.
```

- **Question 6: Serial vs parallel benchmarking** Use ICME GPU cluster to compare time estimates for different numbers of threads and blocks.

```
jelc@icme-gpu:~/cme213-para/hw2$ make
g++ -std=c++11 -g -Wall -O3 -fopenmp main_q1.cpp -o main_q1
g++ -std=c++11 -g -Wall -O3 -fopenmp main_q2.cpp -o main_q2
g++ -std=c++11 -g -Wall -O3 -fopenmp main_q2.cpp -D QUESTION6 -o main_q2_part6

jelc@icme-gpu:~/cme213-para/hw2$ sbatch script.sh
Submitted batch job 40975

jelc@icme-gpu:~/cme213-para/hw2$ cat job_40975.out
Date = Thu Apr 14 02:41:05 UTC 2022
Hostname = icmet01
Working directory  = /home/jelc/cme213-para/hw2

Number of nodes allocated = 1
Number of tasks allocated =
Number of cores/task allocated  =
tests_q2.h:22:Test1 TEST PASSED.
tests_q2.h:38:Test2 TEST PASSED.
tests_q2.h:50:Test3 TEST PASSED.
tests_q2.h:67:Test4 TEST PASSED.
tests_q2.h:84:Test5 TEST PASSED.
Serial Radix Sort: PASS
Parallel Radix Sort: PASS
stl: 0.311274
serial radix: 0.0460927
```

```
parallel radix: 0.0596201
Threads Blocks / Timing
           1       2       4       8      12      16      24      32      40
    1    0.049   0.050   0.046   0.048   0.059   0.055   0.059   0.068   0.062   0.
    2    0.048   0.047   0.049   0.049   0.051   0.053   0.058   0.060   0.061   0.
    4    0.045   0.047   0.049   0.048   0.051   0.054   0.058   0.059   0.059   0.
    8    0.045   0.047   0.049   0.050   0.051   0.054   0.056   0.060   0.061   0.
    12    0.048   0.047   0.049   0.049   0.051   0.056   0.059   0.062   0.061   0
    16    0.048   0.047   0.051   0.050   0.052   0.055   0.060   0.060   0.061   0
    24    0.051   0.048   0.049   0.050   0.052   0.055   0.063   0.065   0.063   0
    32    0.049   0.048   0.050   0.050   0.052   0.055   0.058   0.066   0.065   0
    40    0.048   0.050   0.050   0.050   0.053   0.057   0.065   0.068   0.065   0
    48    0.049   0.049   0.049   0.050   0.053   0.055   0.059   0.068   0.067   0
Benchmark runs: PASS
```

Optimal speed achieved at 4/8 threads and 1 block.

In general, time increases with number of blocks (more subdividing overhead), but goes both up and down for number of threads (best at 4/8). The thread result in particular was surprising to me since I had expected the parallel program running on a cluster machine to be able to take advantage of more threads.

Submission information logs.

```
jelc@cardinal1:~$ /afs/ir.stanford.edu/class/cme213/script/submit.py hw2 private/cme213-
Submission for assignment 'hw2' as user 'jelc'
Attempt 1/10
Time stamp: 2022-04-13 20:09
List of files being copied:
    private/cme213-jelc53/hw2/sum.h  [768 bytes]
    private/cme213-jelc53/hw2/parallel_radix_sort.h  [7625 bytes]

Your files were copied successfully.
Directory where files were copied: /afs/ir.stanford.edu/class/cme213/submissions/hw2/jel
List of files in this directory:
    sum.h  [768 bytes]
    parallel_radix_sort.h  [7625 bytes]
    metadata  [137 bytes]

This completes the submission process. Thank you!

jelc@cardinal1:~$ ls /afs/ir.stanford.edu/class/cme213/submissions/hw2/jelc/1
metadata  parallel_radix_sort.h  sum.h
```