Homework 5.2 Answer

1. Changes made as per the directions - set #define DEFAULT_NUM_ELEMENTS to 16777216. Set #define MAX_RAND to 3
2. Results when run without arguments, including the host CPU and GPU processing times and the speedup is as given below
    1. Processing **16000000** elements

       Host CPU Processing time: 119.331001 (ms)

       G80 CUDA Processing time: 4.625000 (ms)

       Speedup: 25.801298X

       Test PASSED
    2. Processing **16777216** elements

       Host CPU Processing time: 125.331001 (ms)

       G80 CUDA Processing time: 4.828000 (ms)

       Speedup: 25.959196X

       Test PASSED
3. Arrays not a power of 2 are handled by zero padding the last elements till they fit exactly into the block.

   Shared memory bank conflicts are handled in the following way :
   Each thread accesses the shared memory in following pattern:
   scan_array[t+CONFLICT_FREE_OFFSET(t)]=d_idata[t+start];
   scan_array[t+blockDim.x+CONFLICT_FREE_OFFSET(t+blockDim.x)]=d_idata[t+blockDim.x+start];

   Here, each thread will load the 2 elements from global memory into the shared memory, which are blockDim.x (256) away. We have 16 shared memory banks.
   Therefore, in a warp, 2 threads which are 16 away, will access the same bank. This will reduce memory conflict as compared with the case where the threads access consecutive banks.
   Also, the function CONFLICT_FREE_OFFSET is used to provide an offset to each element of shared memory to avoid bank conflicts.

   To optimize performance :
    1. High degree shared bank conflicts was avoided as per Mark Harris's report "Parallel Prefix Sum (Scan) with CUDA" by setting an offset to each element and leaving every 16th element blank.
    2. Unnecessary iterations are avoided based on number of blocks at the end of each iteration.

Homework 5.2 Answer

4. For GPU :

   For reduction step, we have total of (N-1) operations for each block consisting of (N/2) threads. And each reduction operation involves 1 floating point operation. => Total Floating point operations per block for reduction = (N-1)x1 = N-1

   For post reduction, we again have (N-1) operations for each block consisting of (N/2) threads. And each operation involves 1 floating point operation. => Total Floating point operations per block for post-reduction(exclusive swap method) = (N-1)x1 = N-1

   => Total floating point operations/block = 2(N-1)

   Total number of blocks = ceil(16M/512) + ceil(31250/512) + ceil(31250/512) + ceil(62/512) = 31250+62+1 = 31313 blocks

   => Total floating point operations = 2(N-1)x 31313 = 32001886 operations

   => No. of floating point operations per second = 32001886/(4.5ms) = 7.1x(10^9) = 7.1 GFLOPS

   For CPU :

   ```
   for( unsigned int i = 1; i < len; ++i)
   {
       total_sum += idata[i-1];
       reference[i] = idata[i-1] + reference[i-1];
   }
   ```

   In the above code, there are 2 floating point operations. Even though 1st operation(total sum) is just a check operation, it takes finite time and it is accounted while calculating time taken by CPU.

   Total floating point operations/iteration contributing to the final sum = 2

   No. of iterations = 16x(10^6)-1

   => No. of floating point operations per second = 1.599999x(10^6)x2/((119.435ms) = 0.2678x(10^9) = 267.8 MFLOPS
   GPU performs approx. (2x(10^4)) times more FLOPS than CPU.

   Bottlenecks which limit the performance:
   1. Size of shared memory - limited size of shared memory per SM
   2. No. of registers - limited no. of registers