PA4 Parallel Programming with GPU CS140 Fall 2019

Please read ALL released code carefully, and pay an attention to the function specification comments, including the sequential code for the Jacobi and Gauss-Seidel methods. Do not change the provided function signatures or .h files. Do not add any global variable. Do not change Makefile and the script to run the code. Add your code in the area under Phrase "Your solution". The number of lines of code to add for this assignment is small, but CUDA code cannot run in CSIL or GradeScope and you have to submit a GPU job to run and debug on Comet. The waiting queue for GPU on Comet can be very long, and completing this assignment can take much longer time than PA3. Start your coding as soon as possible!

1. Following Question 3 of PA3, we want to parallelize the Jacobi method with GPU. Matrix A is of size *n* x *n*. Column vectors *x* , y, and *d* are of size nx1.

k=0; error =1;
while *(k<t and error >threshold)*
y = d+ Ax;
error = ||y-x||;
x=y;
k=k+1;
EndWhile

The error formula is: $||y-x||=\max(|y_i-x_i|)$ which is the maximum of all elemental differences of two vectors. You only need to consider that matrix A is a regular dense square matrix of dimension n=4096 or less, and the test data used is the same as Question 3(a) of PA3.

Write the code to run parallel threads in each iteration of k using a GPU device with block mapping. Allocate b blocks of m threads with bxm threads in total, and each thread handles a block of rows to perform $y_i=d_i+A_i$ *x where A_i is the i-th row of matrix A. Report time performance in seconds for solving the given test matrix with n=4096 and t=1024 using the following four thread configurations: 4 blocks of 128 threads, 8 blocks of 128 threads, 16 blocks of 128 threads, and 32 blocks of 128 threads. The total number of threads is doubled from one and another thread. Do you observe the performance gain proportionally when doubling the total number of threads? Provide an explanation for the trend.

2. Revise the code of Question 1 to utilize the shared memory within each thread block. The key steps within each thread block in computing $y_i=d_i+A_i$ *x for some rows of matrix A can be outlined as follows :

Allocate the shared memory space for 4096 elements of vector x
Fetch vector x from global memory to this local shared memory in parallel
Call a barrier to synchronize the threads within the same thread block
Perform $y_i=d_i+A_i$ *x in parallel assigned to this block of threads

Because there is not enough shared memory space, we assume n does not exceed 4096 and the above scheme still directly accesses the global memory for matrix A and vector d. Note that the impact through fast access of vector d is limited as time cost is dominated by Ai *x.

Report and compare the time performance in seconds for n=4096 and t=1024 with 8 blocks of 128 threads, and 32 blocks of 128 threads with and without using shared memory for vector x. Explain the performance difference. Notice when more threads are allocated, less workload per thread can utilize the shared memory copy.

3. Revise the code of Question 1 by using parallel asynchronous Gauss-Seidel update. The sequential code that uses Gauss-Seidel method is revised as:

k=0; error =1; y=x;
While *(k<t and error >threshold)*
Update each element: $y_i=d_i+A_iy$ *for index i from 1 to n;*
error = $||y-x||$*;*
x=y;
k=k+1;
EndWhile

After mapping computation to threads, we let threads run asynchronously to compute vector y for an extra number of iterations without explicit inter-thread synchronization and error check.

k=0; error =1; y=x;
While *(k<t and error >threshold)*
ParallelFor *i= 1 to n with no inter-thread synchronization*
Repeat *r times*
$y_i=d_i+A_iy$
EndRepeat
$e_i=y_i-x_i$
EndFor
error = $||e||$*;*
x=y;
k=k+ r;
EndWhile

Report the time performance in seconds for n=4096, t=1024, and r=5 with 8 blocks of 128 threads, and 32 blocks of 128 threads, and also report the number of actual iterations used, and if the solver converges faster. Compare performance with code of Question 1 in the above setting and explain the difference.

Functions to be tested: it_mult_vec, mult_vec, , mult_vec_shared_x , mult_vec_async

Files to submit: it_mult_vec.cu**,** README.txt**,** it_mult_vec_test.out

File It_mult_vec_test.out is the output trace of your code in running the tests of the *unmodified* it_mult_vec_test.cu on Comet. README.txt report sand compares performance for the required tests with an explanation.