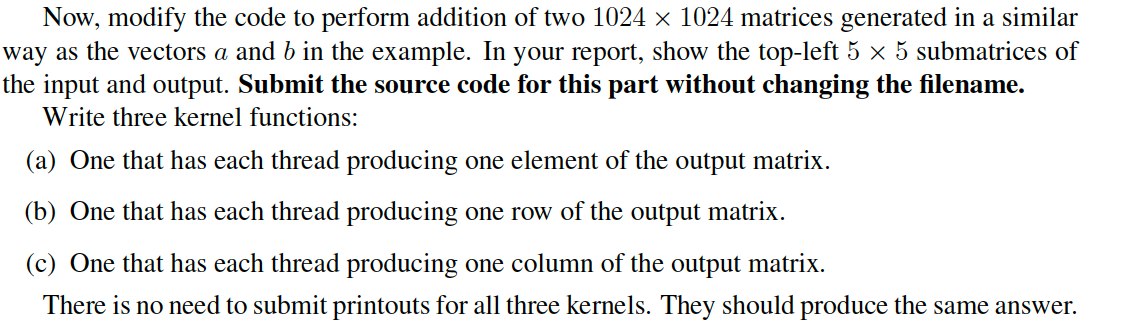
1.



Please see the code in attachment .

Report:

the top-left 5\*5 input a:

0.000000 0.708073 0.826822 0.019915 0.572750

0.025133 0.555284 0.928855 0.087783 0.414230

0.098005 0.396936 0.987775 0.197092 0.264333

0.211290 0.248949 0.997658 0.336853 0.138128

0.353599 0.126201 0.957511 0.493015 0.048302

the top-left 5\*5 input b:

1.000000 0.291927 0.173178 0.980085 0.427250

0.974867 0.444716 0.071145 0.912217 0.585770

0.901995 0.603064 0.012225 0.802908 0.735667

0.788710 0.751051 0.002342 0.663147 0.861872

0.646401 0.873799 0.042489 0.506985 0.951698

the top-left 5\*5 output c:

1.000000 1.000000 1.000000 1.000000 1.000000

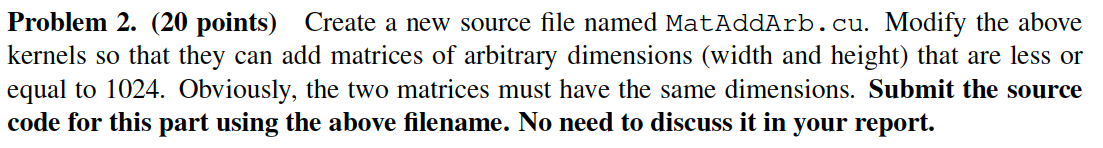
1.000000 1.000000 1.000000 1.000000 1.000000

1.000000 1.000000 1.000000 1.000000 1.000000

1.000000 1.000000 1.000000 1.000000 1.000000

1.000000 1.000000 1.000000 1.000000 1.000000

2.



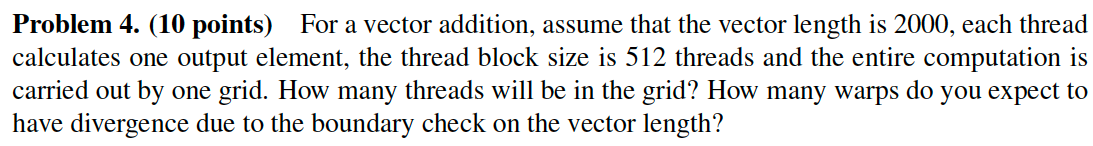
Please see the code in attachment .

3. 图片包含 天空

描述已自动生成

Please see the code in attachment .

4.



Answer:

1. Ceil(2000/512) =4 , so there will be 4 \* 512 = 2048 threads in the grid.
2. Only one warp , because for the extra 48 threads, 32 will be totally out of bound. The remaining 16 will be part of a warp where half the threads will be in-bound.

5.

图片包含 天空

描述已自动生成

Answer:

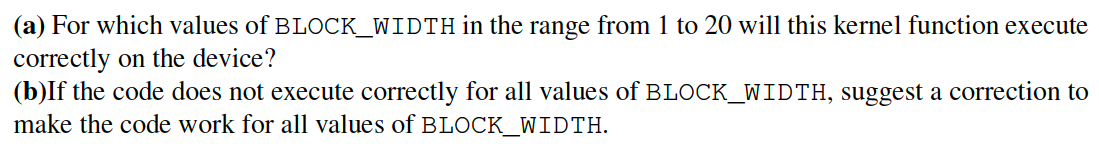
NO.

Assume the kernel code like this:

\_\_global\_\_ void matrix\_output\_element(double \*a, double \*b, double \*c, int n)  
{  
 // Get our global thread ID  
 int id = blockIdx.x \* blockDim.x + threadIdx.x;  
 c[id] = a[id] + b[id];  
}

because each thread access the corresponding elements of the two input matrices a[id] and b[id] for only 1 time, and no threads share input data, so using shared memory will not reduce the global memory bandwidth consumption.

6.



Answer:

a).This kernel function will execute correctly on device for the value of BLOCK\_WIDTH less than or equal to 5.

Because for this situation, a bock will contain one warp of threads, and will execute the entire program in lock-step for all the threads. If the BLOCK\_WIDTH is over 5, it will create thread blocks with more than 1 warp, so the values written by a warp to shared memory array will precede the reads by another warp from same memory locations. Thus, there will have a overwrite problem which causes data lost.

b). We can solving this problem by call a \_\_syncthreads() function between the read and write of the shared memory array, which can guarantee the order of shared memory reads and writes across threads.

\_\_global\_\_ void BlockTranspose(double\* A, int A\_width, int A\_height)  
{  
 \_\_shared\_\_ double blockA[BLOCK\_WIDTH][BLOCK\_WIDTH];  
 int baseIdx = blockIdx.x \* BLOCK\_WIDTH + threadIdx.x;  
 baseIdx += (blockIdx.y \* BLOCK\_WIDTH + threadIdx.y) \* A\_width;  
 blockA[threadIdx.y][threadIdx.x] = A[baseIdx];  
 \_\_syncthreads();  
 A[baseIdx] = blockA[threadIdx.x][threadIdx.y];  
}