Exam : GP-GPU

Instructions

All exercises are independent.

1 General understanding

Answers should be short.

Question 1.
We want to use each thread to calculate eight (8) output elements of a vector addition. Each thread block process `8*blockDim.x` consecutive elements that form 8 sections. All threads in each block will first process a section first with each processing one element. They will then all move to the next section with each processing one element.

What would be the kernel code expression for forming the value of i, the data index of the first element to be processed by each thread ?

We recall that `threadIdx` is the index of the thread, `blockIdx` is the index of the block and `blockDim` is the dimension of the block. All three variables have three components : `x`, `y`, `z`.

Question 2.
For a vector addition, assume that the vector length is 8000, each thread calculates eight (8) output element, and the thread block size is 512 threads. The programmer configures the kernel launch to have a minimal number of thread blocks to cover all output elements.

How many threads will be in the grid ?

Question 3.
We are to process an 800×565 (784 pixels in the x or horizontal direction, 565 pixels in the y or vertical direction) picture with the `PictureKernel` below :

```
1   __global__ void PictureKernel(float* d_Pin, float* d_Pout, int n, int m) {
2     // Calculate the row # of the d_Pin and d_Pout element to process
3     int Row = blockIdx.y*blockDim.y + threadIdx.y;
4
5     // Calculate the column # of the d_Pin and d_Pout element to process
6     int Col = blockIdx.x*blockDim.x + threadIdx.x;
7     // each thread computes one element of d_Pout if in range
8     if ((Row < m) && (Col < n)) {
9       d_Pout[Row*n+Col] = 2*d_Pin[Row*n+Col];
10    }
11  }
```

Assume that each block is organized as a 2D `16X16` array of threads.

Assume that **`int`** variable `n` has value `784` and **`int`** variable `m` has value `565`. The kernel is launched with the statement `PictureKernel<<<DimGrid,DimBlock>>>(d_Pin, d_Pout, n, m);`

How do you specify the kernel configuration properly ? (you must provide the expression of `DimGrid` and `DimBlock`)

Question 4.
In the previous question, how many warps will have control divergence ?

Question 5.
The kernel resource usage is 30 registers/thread and 5KB of shared memory per block. The kernel has 1,920,000 threads in total and a square grid with dimension 50 on each side. What is the maximum number of simultaneous blocks that will run on a single streaming multiprocessors ?

Question 6.
Assume that a kernel has an atomic operation on a variable in the global memory. If you know that each load or store access to the global memory takes 1,000 clock cycles and the clock runs at the 1 GHz. What is the maximal throughput one can hope for the atomic operations on this variable ?

Question 7.
Assume a DRAM (CPU memory) system with a burst size (number of continuous bytes that can be transmitted simultaneously) of 512 bytes and a peak bandwidth of 240 GB/s. Assume a thread block size of 1024 and warp size of 32 and A as a double- precision array in the global memory. What is the maximal memory bandwidth we can hope to achieve ? Explain.

2 CUDA Basics

Question 8.
For the vector addition kernel and the corresponding kernel launch code, answer each of the sub- questions below.

```
__global__ void vecAddKernel(float* A, float* B, float* C, int n)
{
  int i = threadIdx.x + blockDim.x * blockIdx.x;
  if(i<n) C_d[i] = A_d[i] + B_d[i];
}
```

```
int vectAdd(float* A, float* B, float* C, int n)
{
  //assume that size has been set to the actual length of
  //arrays A, B, and C
  int size = n * sizeof(float);

  cudaMalloc((void **) &A_d, size);
  cudaMalloc((void **) &B_d, size);
  cudaMalloc((void **) &C_d, size);
  cudaMemcpy(A_d, A, size, cudaMemcpyHostToDevice);
  cudaMemcpy(B_d, B, size, cudaMemcpyHostToDevice);
  vecAddKernel<<<ceil(n/1024.0),1024>>>(A_d, B_d, C_d, n);
  cudaMemcpy(C, C_d, size, cudaMemcpyDeviceToHost);
}
```

1. Assume that the size of A, B, and C is 10,000 elements each. How many thread blocks will be generated ?
2. Assume that the size of A, B, and C is 10,000 elements each. How many warps are there in each block ?
3. Assume that the size of A, B, and C is 10,000 elements. How many threads will be created in the grid ?
4. Assume that the size of A, B, and C is 10,000 elements each. Is there any control divergence during the execution of the kernel ? If so, identify the block number and warp number that causes the control divergence. Explain why or why not.
5. Assume that the size of A, B, and C is 20,000 elements each. Is there any control divergence during the execution of the kernel ? If so, identify the line number of the statement that causes the control divergence. Explain why or why not.

Question 9.
The following CUDA code is intended to perform a sum of all elements of the `partialSum` array. Assume that the kernel is launched with 1024 threads in each block. First, provide the missing operation, and second estimate the fraction of the iterations of the for loop that will have branch divergence.

```
__shared__ float partialSum[2048];
unsigned int tid = threadIdx.x;
for (unsigned int stride = blockDim.x; stride > 0; stride = stride / 2)
{
  __syncthreads();
  if (tid < stride)
    partialSum[tid] += partialSum[tid+stride];
}
```

1. How many versions of partialSum[2048] will be created for each block ?
2. How many iterations will the for-loop take ?
3. Under what condition will there be control divergence in the if-statement ? Assume that the warp size is 32.
4. How many iterations of the for-loop will have control divergence in the if- statement ?
5. How many warps have control divergence in the for-loop where stride value is 16 ?
6. In each block, how many warps have control divergence in the for-iteration where stride value is 4 ?

3 Shared memory

Question 10.
This question tests your ability to handle boundary conditions in a tiled matrix multiplication algorithm. For simplicity, we will only handle square matrices.

Consider the following tiled matrix multiplication code :

```
#define TILE_WIDTH
__global__ void sgemm(float* M, float* N, float* P, int Width) {

  __shared__ float Mds[TILE_WIDTH][TILE_WIDTH];
  __shared__ float Nds[TILE_WIDTH][TILE_WIDTH];

  int bx = blockIdx.x;  int by = blockIdx.y;
  int tx = threadIdx.x; int ty = threadIdx.y;

  // Identify the row and column of the P element to work on
  int Row = by * TILE_WIDTH + ty;
  int Col = bx * TILE_WIDTH + tx;

  float Pvalue = 0;
  // Loop over the M and N tiles required to compute P element
  for (int m = 0; m < ____________________; ++m) {
    // Collaborative load of M and N tiles into shared memory
    if(____________________) {
      Mds[ty][tx] = M[Row*Width + m*TILE_WIDTH + tx];
      Nds[ty][tx] = N[(m*TILE_WIDTH + ty)*Width + Col];
    }
    __syncthreads();
    if(____________________) {
      for (int k = 0; k < ____________________; ++k) {
        Pvalue += Mds[ty][k] * Nds[k][tx];
      }
    }
    __syncthreads();
  }
  P[Row*Width + Col] = Pvalue;
}
```

Fill in the missing boundary checks on lines 17, 20, 26, and 27 to make this code run correctly. If you think it is impossible to make this code to run correctly as is, state why, and then make the necessary changes to fix it by inserting a maximum of 6 lines.