408: Applied Parallel Programming

Fall 2018 – Midterm Exam 1

October 9th, 2018

1. This is a closed book exam except for 1 sheet of hand-written notes
2. You may not use any personal electronic devices except for calculator
3. Absolutely no interaction between students is allowed
4. Illegible answers will likely be graded as incorrect

Good Luck!

Name:____

NetID:____

Exam Room:___________________________________

Question 1 (25 points): _______________

Question 2 (25 points): _______________

Question 3 (25 points): _______________

Question 4 (25 points): _______________

Total Score: _______________

Problem 1 (25 points): Multiple Choice

Choose the proper response, and if multiple responses are correct, choose all. No partial credit will be provided if the answer is partially correct, or wrong.

Part 1(a) (2 points) We want to use each thread to calculate eight elements of a vector addition. Each thread block process 8*blockDim.x consecutive elements that form eight sections. All threads in each block will first process a section first, each processing one element. They will then all move to the next section, each processing one element. Assume that variable i should be the index for the first element to be processed by a thread. What would be the expression for mapping the thread/block indices to data index of the first element?

- ☐ i = blockIdx.x * blockDim.x + threadIdx.x+8
- ☐ i = (blockIdx.x * blockDim.x + threadIdx.x) * 8
- ☐ i = blockIdx.x * blockDim.x * 8
- ☐ i = blockIdx.x * blockDim.x * 8 + threadIdx.x
- ☐ None of the above

Part 1(b) (2 points) What are the scopes of shared memory and barrier synchronization?

- ☐ per block; per warp
- ☐ per warp; per warp
- ☐ per block; per block
- ☐ per SM; per block
- ☐ None of the above

Part 1(c) (2 points) For a vector addition, assume that the vector length is 9000, each thread calculates 9 output elements, and the thread block size is 256 threads. The programmer configures that kernel launch to have a minimal number of thread blocks to cover all output elements. How many thread will be created in the grid?

- ☐ 1000
- ☐ 900
- ☐ 1024
- ☐ 2000
- ☐ 2048
- ☐ 8292

Part 1(d) (2 points) If a CUDA device's SM (streaming multiprocessor) can take up to 1536 threads and up to 8 thread blocks. Which of the following block configuration would result in the most number of threads in the SM?

- ☐ 64 threads per block
- ☐ 128 threads per block
- ☐ 256 threads per block
- ☐ 1024 threads per block

Part 1(e) (2 points) We would like to launch a matrix multiplication kernel to multiply an 80 X 96 matrix M and a 96 X 40 matrix N, using 16 X 16 thread blocks. How many blocks will be launched if each thread is responsible for four elements?

- ☐ 4
- ☐ 6
- ☐ 8
- ☐ 10

Part 1(f) (2 points) For a tiled-matrix multiplication kernel, if we use a 16 X 16 tile, what is the reduction of memory bandwidth usage for input matrices M and N?

- ☐ 1/8 of original usage
- ☐ 1/16 of original usage
- ☐ 1/32 of original usage
- ☐ 1/64 of original usage

Part 1(g) (3 points) For a 1D tiled convolution kernel using shared memory (using strategy 2), assume that we use a block size of 1024 and a mask_width of 7. What is the average number of times each data element of an internal tile (no ghost cells) is reused from the shared memory? If you don't have a calculator, provide an expression that indicates how to derive the answer

- ☐ 6.86
- ☐ 6.96
- ☐ 8.93
- ☐ 8.89

Part 1(h) (3 points) For a tiled 3D convolution, assume that we load an entire input tile, including the halo elements into the shared memory when calculating an output tile.

Further assume that the tiles are internal and thus do not involve any ghost elements. The mask is a 3x3x3 cube and each output tile is a 4x4x4 cube. Which of the choices is the closest to the average number of times each input element will be accessed from the shared memory during the calculation of an output tile?

- ☐ 32
- ☐ 16
- ☐ 8
- ☐ 4

Part 1 (i) (3 points) In matrix multiplication, suppose we use 16 X 32 rectangular tiles to process output matrices of 1000 X 1200. Within EACH thread block, what are the possible number of warps that will have control divergence due to handling boundary conditions? For example, the response 32, 16, 0 means that each thread block will have one of 32, 16 or 0 divergent warps. Chose the answer that is most precise.

- ☐ 32, 16, 0
- ☐ 32, 8, 0
- ☐ 32, 0
- ☐ 16, 8, 0
- ☐ None of the above

Part 1(j) (4 points) For a tiled 2D convolution, assume that we load an entire input tile, including the halo elements into the shared memory when calculating an output tile. Assume we use Strategy 2. The mask is a 3x3 square and each output tile is a 4x4 square. Which of the choices is the closest to the average number of different blocks any particular input element will be accessed by?

- ☐ 40/16
- ☐ 36/8
- ☐ 9/4
- ☐ 3/2

Solution:

1.a	D
1.b	C
1.c	C
1.d	C
1.e	B
1.f	B
1.g	B
1.h	C
1.i	D
1.j	C

Problem 2 (22 points): Column Permutation

In real-world applications, programmers may need to reorder columns in a matrix based on a specified permutation vector. For example, if the original matrix **`idata`** is [*0 1 2 3 6 4 7 5 8*] and the permutation vector (**`perms`** in the code; the length equal to number of columns in the input/output matrix) is [*2 0 1*], the result matrix **`odata`** should be [*2 0 1 4 3 6 8 7 5*] after the column permutation. The "2" in **`perms[0]`** vector means the column **`odata[0]`** comes from the column 2 in **`idata`** and so forth. Please answer the following questions.

Part 2(a) (5 points): To do this on GPU, please fill in the missing index calculations in the CUDA code below. The matrices are stored in one-dimensional arrays in the row-major layout. In the following code, we only handle square input/output matrix (**`rows = cols`**) with **`rows/cols`** as multiples of 32.

```
#define BLOCK_WIDTH 32
// Kernel Code
__global__
void column_reorder(float *odata, float *idata, int *perms,
                        int cols, int rows)
{
   int x = blockIdx.x * BLOCK_WIDTH + threadIdx.x;
   int y = blockIdx.y * BLOCK_WIDTH + threadIdx.y;

    odata[_______________] = idata[_______________];
}

// Host Code
int main ()
{
    ...
    // Invoke Kernel Here
    dim3 dimGrid(cols/BLOCK_WIDTH, rows/BLOCK_WIDTH, 1);
    dim3 dimBlock(BLOCK_WIDTH, BLOCK_WIDTH, 1);
    column_reorder<<<dimGrid, dimBlock>>>(d_odata, d_idata, d_perms,
                                          cols, rows);
    ...
}
```

Odata blank: y * cols + x (2 points)
Idata blank: y * cols + perms[x] (3 points)

P

Part 2(b) (5 points): For the access pattern of **`idata`** and **`odata`**, which of them will have coalesced access, or both, or neither? Please explain.

Access to odata **(2.5 points)** is coalesced while access to idata **(2.5 points)** is not. + reasonable explanation

Part 2(c) (5 points): Consider the case where we decide to put the **`perm`** vector in shared memory and call it **`perm_shared`**. Assuming the numbers in the vector are 32-bits integers, what is the minimum amount of shared memory in bytes we need to allocate for each block with the block and grid dimensions shown in the code line 18 and line 19. Show your computation to help explain your answer.

BLOCK_WIDTH*4 bytes **(5 points)**
Anything greater than that **(1 point)**

Part 2(d) (2 points): For question 2(c), how many total blocks will be allocated? Show your computation to help explain your answer.

`(cols/BLOCK_WIDTH) * ( row/BLOCK_WIDTH)   (2 points)`

Part 2(e) (5 points): We now need to make sure this code can be applied on rectangular matrices and arbitrary **`BLOCK_WIDTH`**. What changes do you need to make on the original code? You can modify, add, or delete lines in the original code, both kernel side and host side. Please note that you may not need all the empty lines below and overly complex answers will result in lost points.

Example:
[Add between line 1 and line 2]: __syncthreads();

[Add between line 7 & 8]: _______if (x < cols && y < rows) _________. **(2.5 points)**

[__Change line 15 to___]:
dimGrid(ceil(1.0*cols/BLOCK_WDITH),ceil(1.0*rows/BLOCK_WDITH), 1); **(2.5 points)**

[_______________________]: ____.

[_______________________]: ____.

[_______________________]: ____.

[_______________________]: ____.

Problem 3 (28 points): Convolution

For this question, we will developed a tiled 2D convolution where each thread is responsible for computing 2 consecutive output elements in the x dimension. This means each thread block will need to compute 2 output tiles in the resulting output. Each thread block should load enough data from the input into the shared memory for computing its output without loading any row or column of the input tile more than once. For this problem we will use Strategy 2, where all threads participate in loading shared memory, and some of the threads participate in generating output. For this question, the mask is 3x3 and each thread block consists of 5x5 threads.

Part 3(a) (15 points): Fill in the blanks in the code below to complete the kernel.

```
#define MASK_WIDTH 3
#define MASK_RADIUS 1
#define BLOCK_WIDTH 5
#define INPUT_TILE_WIDTH_X  8 (10)
#define OUTPUT_TILE_WIDTH_X 6 (8)

#define INPUT_TILE_WIDTH_Y BLOCK_WIDTH
#define OUTPUT_TILE_WIDTH_Y MASK_WIDTH

     __constant__ float mask[MASK_WIDTH][MASK_WIDTH];

     __global__
     void conv2d(float *input, float *output, int y_size, int x_size) {

     __shared__ float inputTile [INPUT_TILE_WIDTH_Y][INPUT_TILE_WIDTH_X];
       int tx = threadIdx.x; int ty = threadIdx.y;
       int bx = blockIdx.x; int by = blockIdx.y;

       //Calculate index of first output element
       int first_x_o =  bx * OUTPUT_TILE_WIDTH_X + (2*tx);
       int first_y_o =  by * OUTPUT_TILE_WIDTH_Y + ty ;

       //Calculate index of input element to put in shared memory

       int x_i = first_x_o - tx - MASK_RADIUS; (or MASK_RADIUS)
       int y_i = first_y_o - MASK_RADIUS;
       //Iterate to load whole input tile
       for (int i = 0; i < 2; i++) {
         //Calculate where to put element from input into shared memory

         int tile_x_idx = tx + + i * BLOCK_WIDTH; (or i + tx)
         int tile_y_idx = ty;
         if ( (x_i >= 0)  &&  (x_i < x_size)  &&
              (y_i >= 0)  &&  (y_i < y_size) &&
              (tile_x_idx < INPUT_TILE_WIDTH_X) &&
36.                  (tile_y_idx < INPUT_TILE_WIDTH_Y))   {
37.
38.              inputTile[tile_y_idx][tile_x_idx] =
39.                        input[ (y_i * x_size) + (x_i) ];
40.            }
41.            else if ((tile_x_idx < INPUT_TILE_WIDTH_X) &&
42.                     (tile_y_idx < INPUT_TILE_WIDTH_Y)) {
43.
44.               inputTile[tile_y_idx][tile_x_idx] = 0.0f;
45.            }
46.            //Determine which column to load the next iteration
47.            x_i += BLOCK_WIDTH; (or 1)
48.          }
49.
50.
51.          //Iterate to compute 2 output elements per thread
52.          for (int i = 0; i < 2; i++) {
53.            float val = 0.0f;
54.            int new_tx = 2*tx + i;
55.            if ( ty <  OUTPUT_TILE_WIDTH_Y &&
56.                 new_tx <  OUTPUT_TILE_WIDTH_X) {
57.              for (int j = 0; j < MASK_WIDTH; j++)
58.                 for (int k = 0; k < MASK_WIDTH; k++)
59.                     val += mask[j][k]  *
60.                            inputTile[j + ty][k + new_tx];
61.              if (first_y_o < y_size    &&    first_x_o < x_size)
62.                 output[first_y_o * x_size + first_x_o] = val;
63.            }
64.            //Determine next element in x dimension to compute
65.
66.            first_x_o += 1;
67.          }
68.        }
```

Every line is worth 2 point each, except lines 65 is 1 points.
The bracket describes the alternate solution. Every student is given points based on the best of the two complete solution. i.e if the student uses all answer in brackets, he gets full points. A mixed solution will get you best of the two answers.
Example:

MASK_RADIUS, i*BLOCK_WIDTH, BLOCK_WIDTH will give you 4 points and not 6.

If you have mentioned 10,8 for lines 4 and 5→ request regrade for lines 4,5
The two solutions for lines 4 and 5 are independent of the solutions for the other lnes

Part 3(b) (5 points): The code above is incorrect in that it contains no synchronization (i.e., `__syncthreads()`). Please provide pair(s) of line numbers between which `__syncthreads()` is required for correct execution. For the sake of efficiency, we want to **execute** as few `__syncthreads()` as possible.

Between 45 and 47 (only 2 points) Its correct but its not minimal number in execution
Anywhere Between 47 and 50 (full 5 points)

Part 3(c) (5 points): For an internal output tile (no ghost elements), what is the average number of times each input element will be accessed from shared memory during the calculation of an output tile for the correctly working version of this code?

```
(OUTPUT_TILE_WIDTH_X *  OUTPUT_TILE_WIDTH_Y) *
(MASK_WIDTH *  MASK_WIDTH) /
( INPUT_TILE_WIDTH_X *  INPUT_TILE_WIDTH_Y)
```

(6*3) * (3 * 3) / (8 * 5) = 4.05 if they used `INPUT_TILE_WIDTH_X` as 8 and `OUTPUT_TILE_WIDTH_X` as 6

(8*3) * (3 * 3) / (10 * 5) = 4.32 if they used `INPUT_TILE_WIDTH_X` as 10 and `OUTPUT_TILE_WIDTH_X` as 8

Part 3(d) (3 points): Which of the lines in the code in 3(a) might suffer from lack of memory coalescing?

Lines 37-38 (reading from global memory) (1.5 points)
Line 61 (writing to global memory) (1.5 points)

Problem 4 (25 points): Machine Learning

Part 4(a) (10 points): Pied Piper is hiring interns with CUDA + ML experience for Silicon Valley Season 6. Richard Hendricks has a bunch of questions to check how knowledgeable you are.

Choose the proper response, and if multiple responses are correct, choose all. No partial credit will be provided if the answer is partially correct, or wrong. (Each carries 2 point)

1. A multi-layer perceptron can perfectly learn a linear function given enough training steps.

   ☐ True
   ☐ False

2. A single perceptron can compute the XOR function.

   ☐ True
   ☐ False

3. A 3 layer perceptron with 10 neurons in each layer has a total of X connections, requires Y weight parameters, and Z biases.

   ☐ X = 200, Y = 200, Z = 200
   ☐ X = 100, Y = 200, Z = 20
   ☐ X = 200, Y = 200, Z = 10
   ☐ X = 200, Y = 200, Z = 20

4. With back propagation, we are evaluating the gradient of the ___________ relative to the ___________.

   ☐ loss function, weights
   ☐ activation function, cost function
   ☐ cost function, input
   ☐ cost function, biases

5. With stochastic gradient decent, a mini-batch requires processing all inputs in the training set.

   ☐ True
   ☐ False
   ☐ Neither

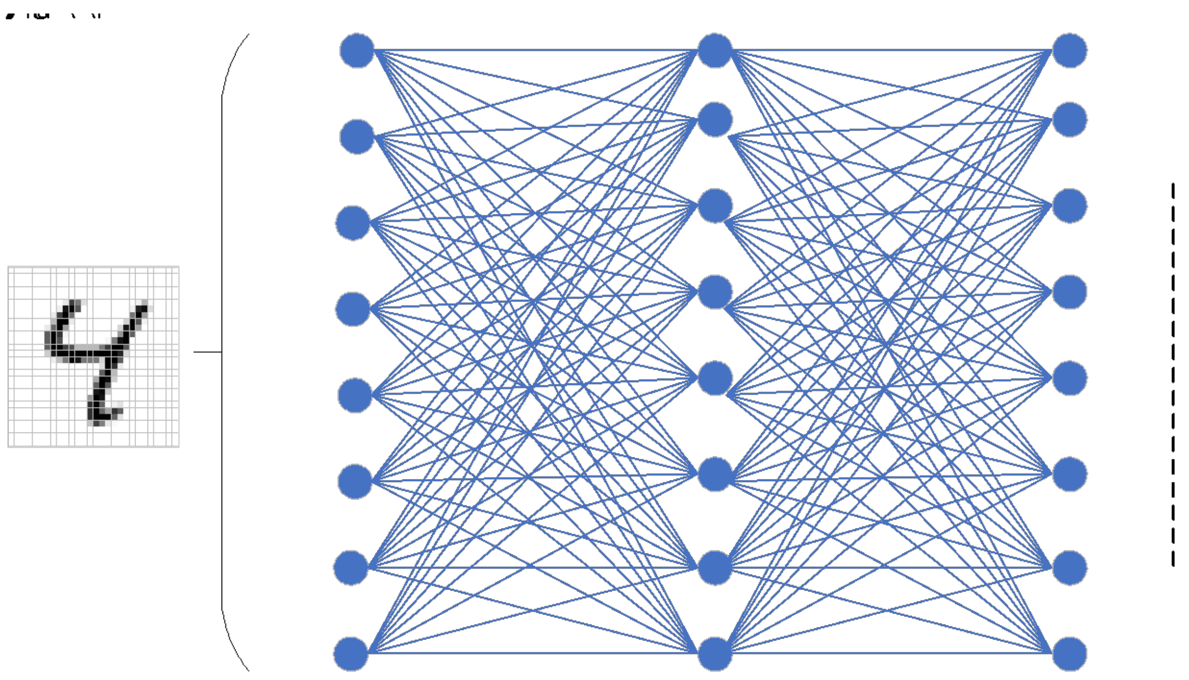
Q1.

1. True
2. False
3. X = 200, Y = 200, Z = 10 or X = 200, Y = 200, Z = 20 (request regrade if you wrote Z = 20 - it based on assumption and the question is not clear)
4. A and D – Give 2 point only if both are mentioned. No point for 1 correct.
5. False

Part 4(b) (10 points): Congratulation, You are hired! Now, you are an intern working for Pied Piper! You work very closely with Richard Hendricks to pivot a digit recognition application for Season 6! Richard asks you to implement optimized multi-layer perceptron with 3 layers as shown in the figure. (NOTE: Image is not scaled to dimensions). It takes input x gray scale image of size 32x32 and has 10 classes for output y, each representing a digit. The inputs and outputs are represented as linearized vectors, x and y. The hidden layer h has 100 neurons in it. The overall equation of the model can be given by

$$h = \sigma(1x + b_1)$$
$$y = \sigma(2h + b_2)$$

Where b_1 and b_2 are vectors holding the bias values, and W_1 and W_2 are weight matrices, and the function $\boldsymbol{\sigma}$ is the sigmoid function.



P

Fill in the dimensions in the following table, based on the architecture of the deep network: (Each carries 2 points)

Q1	Dimension of b_1	
Q2	Dimension of b_2	
Q3	Dimension of W_1	
Q4	Dimension of W_2	
Q5	Total number of parameters to be learned	

Q2.
Initial math:
First layer output - 100 = [100,1024] * [1024] + 100
Second layer output - 10 = [10,100] * [100] + 10

Total parameters:
100*1024 = 102400 + 100 = 102500
10*100 = 1,000 + 10 = 1,010
Total = 1010 + 102500 = 103510

Q1	Dimension of b_1	100
Q2	Dimension of b_2	10
Q3	Dimension of W_1	100,1024 or 1024,100
Q4	Dimension of W_2	10,100 or 100,10
Q5	Total number of parameters to be learned	103510

Part 4(c) (5 points): You realize the both forward-pass equations (for h and y) are the same computation, but with different input dimensions. You want to use a single GPU kernel general enough to perform both. You can disregard the sigmoid function $\boldsymbol{\sigma}$ for this question. Please complete below code base to complete the implementation. More credit will be given to code that is better optimized. Assume **x** is the input vector, **w** is the weight matrix in row-major order, **b** is the bias vector, and **y** is the output vector.

```
 1. __global__
 2. void general_layer(float *y, const float *x, const float *w, const
    float *b, const int ySize, const int xSize) {
 3.
 4.    int tx = blockDim.x * blockIdx.x + threadIdx.x;
 5.    int bx = gridDim.x * blockDim.x;
 6.
 7.    for( int j = tx; j < ySize; j+= bx){
 8.       float  sum =0;
 9.       for( int i =0; i< xSize; i++){
10.
11.             sum += x[______________] * w[______________];
12.       }
13.
14.       y[______________] = sum + b[______________];
15.    }
16.}
```

Q3.
Answer:

Line 10:	i	i * ySize + j
Line 12:	j	j

Reason: We use row-major approach which has coalesced access.

why ySize ---> it is W*x calculation and not x*W.

4 points for fill up the blank. (each blank carries 1 points)
1 point if they have used row major approach to fill it up.