ECE 408 Exam 2
Fall 2016

December 6th 2017

- You are allowed to use one 8.5”x11” sheet of notes.
- Write your name on the top of each following page.
- No interactions with any people other than course staff are allowed.
- This exam is designed to take around 90 minutes to complete. Your exam is due promptly 3 hours after the test begins.
- No computing devices other than simple calculators are permitted.
- You can write down the reasoning behind your answers for possible partial credit.
- Good luck!

Name: _____

UID: _____

Name: _____

Question 1 1 (30 points, 30 minutes): Short Answer and Multiple Choice

1. Given a *jxk* matrix and a *kxl* matrix, we want to multiply these two matrices together and calculate how many multiplication and addition operations our kernel performs. Your 408 TA, Sharon, claims that there are ***k* multiplications** and ***k-1* additions** per thread. She even shows you an example matrix multiplication for ***k***=4 on paper:

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{bmatrix} \times \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 1*1+2*3+3*5+4*7 & 1*2+2*4+3*6+4*8 \\ 5*1+6*3+7*5+8*7 & 5*2+6*4+7*6+8*8 \end{bmatrix}$$

   Compare Sharon's claim to the simple matrix multiplication implementation for MP2. Do they match up? If not, explain.

   Each thread performs k multiplications, one per column of the first matrix (or per row of the second matrix). Each thread also performs k additions because in code, we are using +=. Sharon is forgetting about the first term that you add into the Pvalue.

   To illustrate, let's take example above: Each entry in the final matrix product has 4 multiplications and 3 additions. However, this is not how it's done in code. In code, each entry in the output is summed in a Pvalue, so the first entry in the final output is more like [+1*1+2*3 +3*5+4*7], which is 4 multiplications and 4 additions.

2. Assume a DRAM system with a burst size of 256 bytes and a peak bandwidth of 240 GB/s. Assume a thread block size of 256 and warp size of 32 and that A is a float array in the global memory. What is the maximal memory data access throughput we can hope to achieve in the following access to A?

```
int i = 4*blockIdx.x * blockDim.x + threadIdx.x;
float temp = A[i];
```

   (A) 240 GB/s
   (B) 120 GB/s
   (C) 60 GB/s
   (D) 30 GB/s

   Answer: (A)
   Explanation: Each warp is going to access 128 bytes of consecutive data. This is a fully coalesced access so the maximal achievable bandwidth is 240 GB/s.

3. Given a sparse matrix of integers with m rows, n columns, and k non-zeros. How many integers are needed to represent the matrix in JDS transposed format? Recall that JDS

has a transposed representation. Also, assume that we do not explicitly track the length of each row.

(A) m+n+k
(B) 2k+m+1
(C) 2k+m+n
(D) 2m+1

Answer: (C)
We need k integers for the non-zero elements, k integers for their column indices, n pointers to the beginning of each column after transposition, and m integers to track the original row index before permutation

4. What is the CUDA API call that makes sure that all previous kernel executions and memory copies have been completed?

(A) __syncthreads()
(B) cudaDeviceSynchronize()
(C) cudaStreamSynchronize()
(D) __barrier()

Answer: (B)

5. When your parallel reduction kernel generates a slightly different result than a sequential reduction function for a floating-point input array, what would be the most likely reason?

The operation order has changed, and in one of the orders, one or more additions involved a very small operand and a much larger operand.

6. What type of CUDA memory allocation should be used when allocating host memory for asynchronous data transfer? Why?

CudaHostAlloc(), which allocates pinned memory. This allows DMA to work safely with asynchronous data transfer.

7.

Name: _____

Question 2: Privatization

This question tests your understanding of parallel histogram computation and privatization. Assume that we would like to privatize a histogram that has 2048 bins. Each input data value (buffer array elements) will range from 0 to 2047. However, the shared memory can only accommodate 1024 bins for each block. As a compromise, we decide to privatize the first half of the bins into the shared memory. Whenever the data value falls into a bin in the second half, we will have to increment the global bin.

(A) Complete the following kernel to implement the partial privatization of the histogram.

```
__global__ void histo_kernel(unsigned char *buffer,  long size, unsigned int *histo)
{
  __shared__ unsigned int histo_private[1024];
  int i;
  for (i =  _____;_i <_____;__i+=_____) histo_privat[i] = 0;
   __syncthreads();
  int i = threadIdx.x + blockIdx.x * blockDim.x;
      // stride is total number of threads
  int stride = blockDim.x * gridDim.x;
  while (i < size) {
          if  ( _____) atomicAdd( &(private_histo[buffer[i]), 1);
        else atomicAdd( _____,1 );
          i += stride;
   }
 __syncthreads();
 for ( i= _____;  i < _____; i += _____  )
          atomicAdd( _____ );
}
```

Name: _____

```
__global__ void histo_kernel(unsigned char *buffer,  long size, unsigned int *histo)
{
  __shared__ unsigned int histo_private[1024];
  int i;
  for ( i =  threadIdx.x;   i < 1024;   i+=blockDim.x ) histo_private[i] = 0;
   __syncthreads();
  i = threadIdx.x + blockIdx.x * blockDim.x;
      // stride is total number of threads
  int stride = blockDim.x * gridDim.x;
  while (i < size) {
         if  ( buffer[i] < 1024 ) atomicAdd( &(private_histo[buffer[i]), 1);
         else atomicAdd( &histo[buffer[i]]   ,1 );
         i += stride;
   }
  __syncthreads();
 for ( i = threadIdx.x ; i < 1024; i += blockDim.x )
       atomicAdd( &(histo[threadIdx.x]),       private_histo[threadIdx.x] );
}
```

(B) Can you think of a better partial privatization strategy that will likely result in less contention in the while loop? Outline your strategy.

Each thread can start its privatized section at a position

Name: _____

Question 3: Parallelization

Consider the dense matrix-vector multiplication **Ax** = **b**. The i^{th} element of **b** is the dot product of **x** with the i^{th} row of **A**. Your friend writes a kernel for input matrices with tens of thousands of columns and dozens of rows, correctly invoked with one thread per column.
A is a pointer to a $numRows \times numCols$ row-major matrix,
b is a pointer to a vector of length *numRows*, and
x is a pointer to a vector of length *numCols*.

```
1: __global__ void mv_gpu(float *b, const float *A, const float *x,
2:                const int numRows, const int numCols) {
3:     const int colIdx = blockIdx.x * blockDim.x + threadIdx.x;
4:     if (colIdx < numCols) {
5:         for (int rowIdx = 0; rowIdx < numRows; ++rowIdx) {
6:             b[rowIdx] += A[rowIdx * numCols + colIdx] * x[colIdx];
7:         }
8:     }
9: }
```

(1/4) The result is incorrect! What is the problem?

Race condition on accumulation in b. All threads update all rows in the for-loop so they will have conflicting updates.

(1/4) Describe a correct approach (it does not have to be optimal, but it should be highly parallel), either by modifying the proposed kernel or through a new parallelization. Be sure to specify if any additional memory or synchronization is needed. Mention at least one potential performance shortfall from your approach for the input matrix shape bolded above. **Any CUDA code written will be ignored.**

Add a simple Atomic add - lots of contention.
Multiple columns per thread with privatization - synchronization

Name: _____

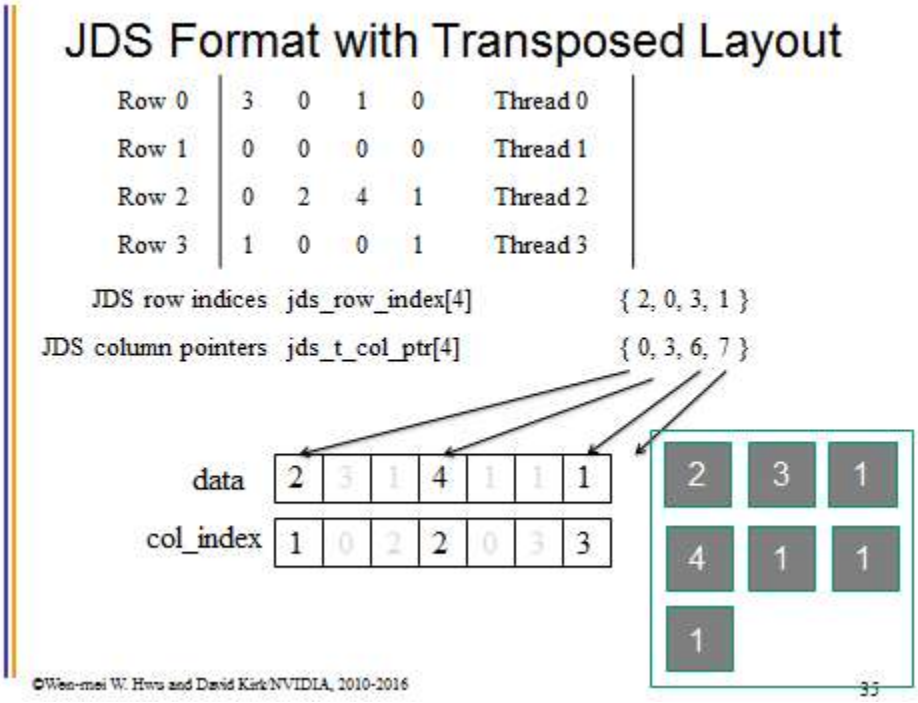
(1/2) Describe an approach that achieves a high degree of parallelism for an input matrix with with thousands of rows and dozens of columns. Be sure to specify any memory or synchronization needed. Describe at least one potential performance shortfall from your approach. Propose an optimization to improve the performance given the shortfall. **Any CUDA code written will be ignored.**

One thread per row - uncoalesced access.
We can use tiling with shared memory to do corner turing.

Question 4. Sparse Matrix Multiplication

This question tests your knowledge of Sparse Matrix representation and operation. For your convenience, we are enclosing a lecture slide that illustrates the JDS_transposed format and kernel design with a small example.

Name: _____



(A) In the following JDS_T kernel, fill in the missing indexing expressions for accessing data (input matrix), x (input vector) and y (output vector).

Name: _____

```
1.__global__ void SpMV_JDS_T(int num_rows, float *data,  int *col_index, int *jds_t_col_ptr,
                             int jds_row_index, float *x, float *y) {
2.  int row = blockIdx.x * blockDim.x + threadIdx.x;
3.  if (row < num_rows) {
4.          float dot = 0;
            unsigned in sec = 0;
5.          while (jds_t_col_ptr[sec+1]-jds_t_col_ptr[sec] > row){
6.           dot += data[____________________] * x[_____________________________________];
7.           sec++;
    }
8.  Y[_________________] = dot;
   }
 }
```

```
1.   __global__ void SpMV_JDS_T(int num_rows, float *data,
                    int *col_index, int *jds_t_col_ptr, int jds_row_index,
                    float *x, float *y) {
2.    int row = blockIdx.x * blockDim.x + threadIdx.x;
3.    if (row < num_rows) {
4.          float dot = 0;
            unsigned in sec = 0;
5.     while (jds_t_col_ptr[sec+1]-jds_t_col_ptr[sec] > row){
6.          dot += data[jds_t_col_ptr[sec]+row] * x[col_index[jds_t_col_ptr[sec]+row]];
7.          sec++;
      }
8.     y[jds_row_index[row]] = dot;
    }
   }
```

Name: _____

(B) Assume a matrix that has 32 original rows, 64 columns, and 10 non-zeros in every row. After we transform the matrix into JDS-transposed layout, and launch the SpMV_JDS_T kernel. Is there any control divergence? Why or why not?

There is no control divergence. All 32 threads will take 10 iterations.

(C) In (B), are the memory accesses to the matrix in the for-loop (line 6) coalesced? Why or why not?

The memory accesses to the matrix are coalesced. All elements in the same column of the original matrix are laid out consecutively.

Name: _____

Question 5. Convolution Neural Network

This question tests your understanding of the convolution layer of a CNN. We will start with a basic kernel implementation.

W is the convolution filter weight tensor, organized a tensor W[M, C, K, K], M is the number of output feature maps, C is the number of input feature maps, K is the height and width of each filter.

X is the input feature map, organized as a tensor X[C, H_out+K-1, W_out+K-1], where H_out is the height of ach output feature map, W_out is the width of each output feature map.

Y is the output feature map, organized as a tensor Y[M, H_out, W_out].

Assume that the blockDim is set to (TILE_WIDTH, TILE_WIDTH,1) and that gridDim is set to (M, H_grid*W_grid, 1).

(A) Fill in the missing parts of the basic kernel implementation. The kernel has each thread block to calculate an tile of output feature map elements. Each thread generates one output map element.

```
__global__ void ConvLayerForward_Basic_Kernel(int C, int W_grid, int K,
 float* X, float* W, float* Y)
{
        int m = blockIdx.x;
        int h =  blockIdx.y / W_grid  + threadIdx.y;
        int w = blockIdx.y % W_grid + threadIdx.x;
        float acc = 0.;
        for (int c = 0;  c < C; c++) {  // sum over all input channels
          for (int p = 0; p < K; p++)  // loop over KxK  filter
            for (int q = 0; q < K; q++)
                acc += X[_____________________] * W[_____________________________];
        }
        Y[_______________] = acc;
}
```

Name: _____

```
__global__ void ConvLayerForward_Basic_Kernel(int C, int W_grid, int K,
          float* X, float* W, float* Y)
{
   int m = blockIdx.x;
   int h =  blockIdx.y / W_grid  + threadIdx.y;
   int w = blockIdx.y % W_grid + threadIdx.x;
   float acc = 0.;
   for (int c = 0;  c < C; c++) {                    // sum over all input channels
     for (int p = 0; p < K; p++)                     // loop over KxK  filter
       for (int q = 0; q < K; q++)
         acc += X[c, h + p, w + q] * W[m, c, p, q];
   }
   Y[m, h, w] = acc;
}
```

(B) Define the meaning of variables h, and w in ConLayerForward_Basic_Kernel.

h: _____

w: _____

h is the row index of the output feature map generated by each thread
w is the column index of the output generated by each thread

(C) For a 72x64 input feature map, 8x8 tiles and 3x3 convolution filters, if we use the tiled 2D convolution, what this the average number of times that each input feature map element is reused once it is loaded into the shared memory?

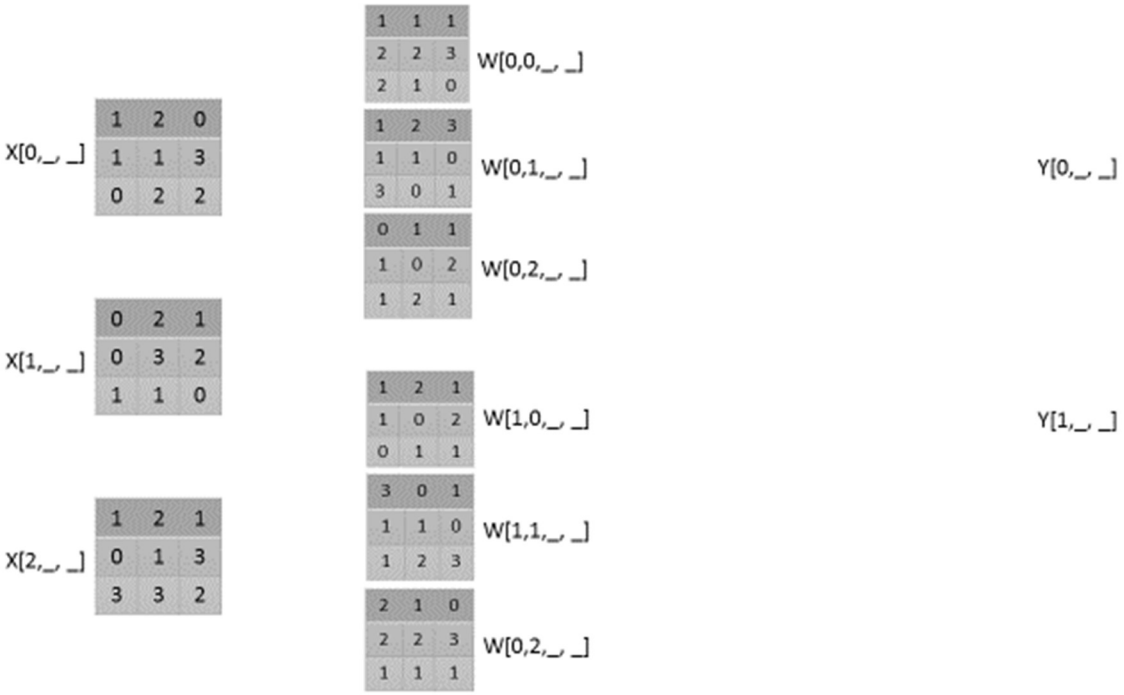
$8^2 * 3^2 / (8+3-1)^2 = 5.76$

(D) If we use each thread block to generate one tile of output feature map elements, how many thread blocks will be generated when we launch the kernel?

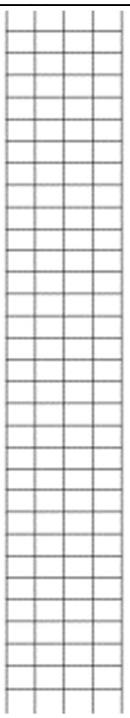
(72/8) * (64/8) = 72 thread blocks

Name: _____

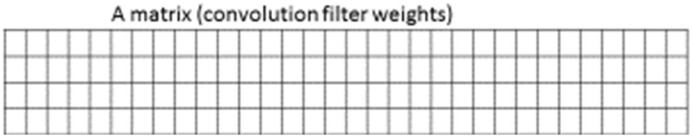
(E) In order to use matrix-multiplication formulation, you need to convert the input feature maps into the unrolled matrix. Assume that we have three 3x3 input feature maps and convolution filter is 3x3. Please fill out the unrolled matrix below based on the contents of the input feature maps. Use only the entries needed.

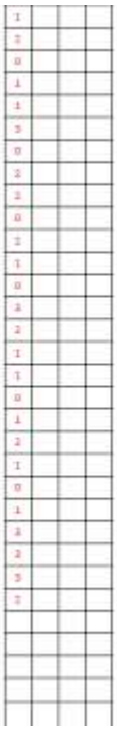


Name: _____



A matrix (convolution filter weights)





A matrix (convolution filter weights)

Name: _____

(D) When we form the A matrix, describe the actions we need to do in order to convert the W tensor into the A matrix. Explain your answer.

There is no need for conversion. W can be used as A, since A is exactly the row major layout of W.

(E) If we use tiled matrix multiplication, how many times do you expect the unrolled matrix elements to be reused if we used a 2x2 tile for the example in (C)?

2 times.