ECE 408 Exam #2 Study Guide, Fall 2019

1. Exam format

- You are allowed one 8.0x11.5 cheat sheet with notes on both sides. The minimal font size for your text on the cheat sheet should be 8pts.
- No interactions with humans other than course staff are allowed.
- This exam is designed to take 150 minutes to complete. To allow for any unforeseen difficulties, we will give everyone 180 minutes.
- This exam is based on lectures, textbook chapters, as well as lab MPs/projects.
- The questions are randomly selected from the topics we covered.
- You can write down the reasoning behind your answers for possible partial credit.

2. Topics to Review from Lectures

Reduction

- Thread index to data index mapping and effect on control divergence

1. For the following basic reduction kernel code fragment, if the block size is 1024 and warp size is 32, how many warps in **a block** will have divergence during the iteration where stride is equal to 1?

```
unsigned int t = threadIdx.x;
Unsigned int start = 2*blockIdx.x*blockDim.x;
partialSum[t] = input[start + t];
partialSum[blockDim+t] = input[start+ blockDim.x+t];
for (unsigned int stride = 1; stride <= blockDim.x; stride *= 2)
{
    __syncthreads();
    if (t % stride == 0) {partialSum[2*t]+= partialSum[2*t+stride];}
}
```

(A) 0
(B) 1
(C) 16
(D) 32

2. In the Question 1, how many warps in a block will have divergence during the iteration

where stride is equal to 16?
(A) 0
(B) 1
(C) 16
(D) 32

3. In the Question 1, how many warps in a block will have divergence during the iteration where stride is equal to 64?
(A) 0
(B) 1
(C) 16
(D) 32

4. For the following improved reduction kernel, if the block size is 1024 and warp size is 32, how many warps will have divergence during the iteration where stride is equal to 16?

```
unsigned int t = threadIdx.x;
Unsigned int start = 2*blockIdx.x*blockDim.x;
partialSum[t] = input[start + t];
partialSum[blockDim+t] = input[start+ blockDim.x+t];
for (unsigned int stride = blockDim.x; stride > 0; stride /= 2)
{
__syncthreads();
if (t < stride) {partialSum[t] += partialSum[t+stride];}
}
```

(A) 0
(B) 1
(C) 16
(D) 32

5. In the previous question, how many warps in a block will have divergence during the iteration where stride is 64?
(A) 0
(B) 1
(C) 16
(D) 32

.

Prefix Sum Patterns

- Thread index to data index mapping and effect on control divergence
- Work efficiency of parallel prefix-sum algorithms
- Thread block prefix-sum algorithm using shared memory
- Global combination of partial prefix sums

1. For the Kogge-Stone scan kernel based on reduction trees, assume that we have 1024 elements, which of the following gives the closest approximation for the number of useful floating-point add operations performed?
   (A) (1024-1)*2
   (B) (512-1)*2
   (C) 1024*1024
   (D) 1024*10

2. For the Brent-Kung scan kernel based on reduction trees and inverse reduction trees, assume that we have 1024 elements, which of the following gives the closest approximation on the total number of useful floating-point add operations performed in both the reduction tree phase and the inverse reduction tree phase?
   (A) (1024-1)*2
   (B) (512-1)*2
   (C) 1024*1024
   (D) 1024*10

3. For the Brent-Kung scan kernel based on reduction trees and inverse reduction trees, assume that we have 2048 elements in each section and warp size is 32, how many warps in each

block will have control divergence during the reduction tree phase iteration where stride is 16?

(A) 0
(B) 1
(C) 16
(D) 32

4. For the Kogge-Stone scan kernel based on reduction trees, assume that we have 1024 elements in each section and warp size is 32, how many warps in each block will have control divergence during the iteration where stride is 16?

(A) 0
(B) 1
(C) 16
(D) 32

5. In the previous question, how many warps in each block will have control divergence during the iteration where stride is 64?

(A) 0
(B) 1
(C) 16
(D) 32

Histogram and Atomic Operations

- Thread index to data index mapping and its effect on memory coelescing
- The reason why parallel histogram algorithms need atomic operations
- The use of privatization to increase efficiency

1. Assume that each atomic operation in a DRAM system has a total read-modify-write latency of 100ns. What is the maximal throughput we can get for atomic operations on the same global memory variable?
   (A) 100G atomic operations per second
   (B) 1G atomic operations per second
   (C) 0.01G atomic operations per second
   (D) 0.0001G atomic operations per second

2. For a processor that supports atomic operations in L2 cache, assume that each atomic operation takes 4ns to complete in L2 cache and 100ns to complete in DRAM. Assume that 90% of the atomic operations hit in L2 cache. What is the approximate throughput for atomic operations on the same global memory variable?
   (A) 0.25G atomic operations per second
   (B) 2.5G atomic operations per second
   (C) 0.0735G atomic operations per second
   (D) 100G atomic operations per second

3. In question 1, if a kernel performs 5 floating-point operations per atomic operation, what is the maximal floating-point throughput of the kernel execution as limited by the throughput of the atomic operations?
   (A) 500 GFLOPS
   (B) 5 GFLOPS
   (C) 0.05 GFLOPS
   (D) 0.0005 GFLOPS

4. In Question 2, if a kernel performs 5 floating-point operations per atomic operation, what is the maximal floating-point throughput of the kernel execution as limited by the throughput of the atomic operations?
   (A) 1.25 GFLOPS
   (B) 12.5 GFLOPS
   (C) 0.368 GFLOPS
   (D) 500 GFLOPS

5. In Question 3, after Gather-to-Scatter transformation, there is no longer atomic operation in the kernel. Assume that the kernel performs 2 floating-point operations for every global memory access. Also, assume that kernel performs single-precision floating-point arithmetic and the global memory bandwidth is 160GB/second. What is the approximate floating-point throughput of the kernel execution as limited by the memory bandwidth?
   (A) 1000 GFLOPS
   (B) 100 GFLOPS
   (C) 80 GFLOPS
   (D) 2 GFLOPS

Sparse Matrix-Vector Multiplication

- Cost and benefit of each format – CSR, ELL, COO, JDS, JDS-Transpose
- No data reuse for matrix elements

1. 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 COO?
   (A) m+n+k
   (B) 3m
   (C) 3n
   (D) 3k

2. 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 CSR?

(A) m+n+k
(B) 2k +m+1
(C) 2k+n
(D) 3k

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 ELL?
   (A) m+n+k
   (B) 2k+m+1
   (C) 2k+n
   (D) None of the above

4. Given a sparse matrix of integers with m rows, n non-zero elements in the row with the largest number of non-zeros, and k non-zeros. How many integers are needed to represent the matrix in JDS-T? Recall that JDS-T has a transposed representation. Also, assume that we keep track of the number of non-zero's in each row, as we specified in the MP assignment.
   (A) m+n+k
   (B) 2k+m+1
   (C) 2k+2m+n
   (D) 2m+1

5. Assume that a GPU has a global memory bandwidth of 160 GB/s. For a single-precision JDS-T kernel, what is the approximate floating-point throughput of the kernel execution as limited by memory bandwidth?
   (A) 4000 GFLOPS
   (B) 400 GFLOPS
   (C) 40 GFLOPS
   (D) 4 GFLOPS

PC System Architecture

- Calculation of PCIe Gen 2 and Gen 3 bandwidth given an X configuration
- Understand the nature, use, and benefit of pinned (page-locked) memory
- Understand the DMA used in CPU-GPU data transfers

1. If Carl's PC has a PCIe Gen3 x16 interconnect for his GPU, what is the closest approximation of the maximal cudaMemcpyAsync() copy throughput from host to GPU that he can expect?
   (A) 100 GB/sec
   (B) 500 GB/sec
   (C) 16 GB/sec
   (D) 1 GB/sec

2. After ran a few test of cudaMemcpy(), Carl realized that the achieved copy throughput was about half of the PCIe bandwidth, what do you think was most likely the main cause of this degradation?
   (A) cudaMemcpy() has a lot of software overhead
   (B) cudaMemcpy() requires an extra copy from the user space to pinged buffer and the extra copy nearly doubles the total copying time.
   (C) The manufacturer lied. The PCIe in the system is actually Gen2.
   (D) The execution time of cudaMemcpy() was measured incorrectly.

3. Peter has a 200MB array that he would like to process with GPU. He measured that the execution time of the code on CPU was 0.02 seconds. He also implemented a kernel and measured that the kernel execution on the GPU was 0.0005, a 40x speedup! However, he needs to transfer the data into the GPU memory and transfer 200MB data output data back to the host memory. His system has a PCIe Gen3 x16 interconnect. What would the real speedup be?
   (A) 40x speedup
   (B) 20x speedup
   (C) 8x slow down
   (D) 1.275x slow down

CUDA Streams and Task Parallelism

- Use of CUDA streams – creation, and insertion into queues
- Correspondence between stream queues and engine queues
- Loop unrolling and call ordering to overlap computation with data transfer

1. For the following code:

```
1) cudaMemcpyAsync(d_A0, h_A+i, SegSize*sizeof(float),.., stream0);
2) cudaMemcpyAsync(d_B0, h_B+I, SegSize*sizeof(float),.., stream0);
3) cudaMemcpyAsync(h_C+i, d_C0, SegSize*sizeof(float),.., stream0);
4) cudaMemcpyAsync(d_A1, h_A+i+SegSize, SegSize*sizeof(float),.., stream1);
5) cudaMemcpyAsync(d_B1, h_B+i+SegSize, SegSize*sizeof(float),.., stream1);
6) cudaMemcpyAsync(h_C+i+SegSize, d_C1, SegSize*sizeof(float),.., stream1);
```

Which of the statements could be executed in parallel on the GPU
(A) 1), 2) and 3)
(B) 1), 2) and 4)
(C) 1) and 4)
(D) 3), and 4)

OpenACC

- The semantics of Parallel regions vs. Parallel Loops. The statements in a parallel region will be redundantly executed by all the gang leaders
- Loops in the parallel regions that are not marked as OpenACC Loops are executed redundantly by all gang leaders
- Iterations of loops in the parallel regions that are marked as OpenACC Loops are distributed to gangs and workers to be executed in parallel.

1. In the following OpenACC code, which of the following is false?

(A) Statement 1 will be executed redundantly executed by the the 32 gangs.
(B) The n iterations of loop i will be divided and distributed to the 32 gangs for execution.

(C) Statement 2 will be executed a total of n times.

(D) Statement 3 will be executed a total of m times.

```
#pragma acc parallel num_gangs(32)
{
    Statement 1;
    #pragma acc loop gang
    for (int i=0; i<n; i++) {
        Statement 2;
    }
    for (int j=0; j<m; j++) {
        Statement 3;
    }
}
```

MPI+CUDA

- The basic API functions of MPI, especially MPI_Send and MPI_Receive
- What are the meanings of MPI Ranks
- How do the MPI processes specialize themselves after they enter main() function.

3. Topics to Review from Lab

Common sources of bugs

- Function prototype problems
- Barrier synchronization problems
- Indexing problems

Performance Issues

- Access patterns that result non-coalesced global memory accesses / shared memory bank conflicts

Convolution

- Different convolution implementations, their pros and cons, and how they reflect on kernel launch configurations

Reduction and Prefix Sum

- Reduction trees, memory access patterns, thread utilization, and branch divergence
- Kogge-Stone vs. Brent-Kung thread organization and element indexing
- Parallel execution overhead and tradeoff between parallel execution and sequential execution

Histogram and Privatization

- Levels of privatization and their applicability
- Allocation and indexing of Shared Memory
- Barrier synchronization and final contribution to the global histogram

Question: 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( ____________________________________ );
}
```

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

Sparse Matrix

- Index calculation for the various formats.
- Control divergence and memory coalescing.

Sparse Matrix Multiplication in JDS_T

This question tests your knowledge of Sparse Matrix representation and operation.

(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).

```
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;
      }
   }
```

(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?

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

Final Project

- Basic convolution layer kernel
- Tiled convolution layer kernel
- Kernel for unrolling X (input feature maps) used in the matrix-matrix multiplication implementation of convolution layer
- Properties of unrolled matrix and the parallelism, data reuse in the corresponding grid executing the matrix multiplication

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 each 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].

(A) Fill in the missing parts of the basic kernel implementation. Use multidimensional indexing notation for simplicity. The kernel has each thread block to calculate a tile of output feature map elements. Each thread generates one output map element.

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

```
__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;
}
```

Answer:

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

h: _____

w: _____

(C) Fill in the missing parts of the tiled kernel implementation. Use multidimensional indexing notation for simplicity. The kernel has each thread block to calculate a tile of output feature map elements. Each thread generates one output map element.

```
__global__ void
ConvLayerForward_Kernel(int C, int W_grid, int K, float* X, float* W, float* Y)
{
   int m, h0, w0, h_base, w_base, h, w;
   int X_tile_width = TILE_WIDTH + K-1;
   extern __shared__ float shmem[];
   float* X_shared = &shmem[0];
   float* W_shared = &shmem[X_tile_width * X_tile_width];
```

```
  m = blockIdx.x;
  h_base = (blockIdx.z / W_grid) * TILE_SIZE;   // vertical base out data index for the block
  w_base = (blockIdx.z % W_grid) * TILE_SIZE; // horizontal base out data index for the block

  tx = threadIdx.x;
  ty = threadIdx.y;
  h = h_base + tx;
  w = w_base + ty;

  float acc = 0.;
  int c, j, k, p, q;
  for (c = 0; c < C; c++) {                    // sum over all input channels
                                               // load weights for W [m, c,..],
// tx and ty used as shorthand for threadIdx.x and threadIdx.y
      if (( ty < K) && ( tx < K))
       W_shared[___,___]= W [__,__,__,__];
       _____// load tile from X[n, c,…] into shared memory

      for (int i = h; i < h_base + X_tile_width; i += TILE_WIDTH) {
       for (int j = w; j < w_base + X_tile_width; j += TILE_WIDTH)
        X_shared[__________,___________] = X[__, __, __, __]
      }

     for (p = 0; p < K; p++) {
      for (q = 0; q < K; q++)
        acc = acc + X_shared[____, _____] * W_shared[__,__];
     }

  }
  Y[__,__,__,__] = acc;
}
```

(D) 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?

(E) 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?

(F) Fill in the missing parts of the unroll kernel implementation. Use multidimensional indexing notation for simplicity. The kernel has each thread block to calculate a tile of output feature map elements. Each thread generates one output map element.

```
void unroll_host_code(int C, int H, int W, int K, float* X, float* X_unroll)
{
   int H_out = H – K + 1;
   int W_out = W – K + 1;
   int num_threads = C * H_out * W_out;
   int num_blocks = ceil((C * H_out * W_out) / CUDA MAX_NUM_THREADS);
   unroll_Kernel<<<num_blocks, CUDA MAX_NUM_THREADS>>>();
}

__global__ void unroll_Kernel(int C, int H, int W, int K, float* X, float* X_unroll)
{
   int c, s, h_out, w-out, h_unroll, w_base, p, q;
```

```
int t = blockId.x * CUDA MAX_NUM_THREADS + threadId.x;
int H_out = H – K + 1;
int W_out = W – K + 1;
int W_unroll = H_out * W_out;

if (t < C * W_unroll) {
  c = t / W_unroll;
  s = t % W_unroll;
  h_out = s / W_out;
  w_out = s % W_out;
  h_unroll = _____;
  w_base = c * K * K;
  for(p = 0; p < K; p++)
    for(q = 0; q < K; q++) {
      w_unroll = _____;
      X_unroll(_____, _____) = X(____, _____, _____);
    }
  }
}
```

(G) How many times on average will be each X element be replicated by the unrolling kernel?

Note that C and M does not play a role. Why?
You should explore the effect of different H_out and W_out on the answer. What happens when H_out and W_out are much larger than K? What is the intuition?

(H) Think about the level of parallelism (number of thread blocks) that will be present when launching matrix multiplication kernel for a convolution layer, (1) for the C1 layer in the LeNet, (2) towards the end of the network, such as C5.

Parallelization

- Understanding dependence constraints
- Understanding communitivity and associativity

Question: Consider the following fragment of C code:

```
for(unsigned int x = 0; x < 512; ++x) {
  for(unsigned int y = 0; y < 512; ++y) {
    for(unsigned int z = 0; z < 512; ++z) {
      out[x][y] = out[x][y] <OP> in[x][y][z];
    }
  }
}
```

Explain how you would optimally parallelize this code and why if:
(a) <OP> was not associative nor communitive
(b) <OP> was associative and communitive

You need to state to what you will assign your blocks and your threads to and why, then write out the kernel function code.

Assume out[x][y] is initialized correctly.

Part (a):

Part (b):