Midterm 2

Started: Dec 7 at 7pm

Quiz Instructions

Please Read These Instructions Carefully

This is an open book exam, but you must **not** solicit help from anyone other than the course staff, and you must not provide help to anyone. **By submitting this exam for grading, you are attesting that this work is your own, and that you have not provided or received help from anyone other than the course staff.**

The exam is 1 hour 30 minutes long, consisting of 15 multiple choice questions for a total of 100 points. The questions will be presented to you in randomized fashion, and it's very possible that each student will have a unique exam. Canvas will save and submit your responses automatically when time expires.

We are expecting that this exam will be time-constrained for most students. Please use your time effectively. As a guide, the number of points allocated to each question roughly corresponds to the amount of time in minutes we expect it to take.

Due to the randomization, we will be unable to make corrections to the exam once its started.

For multiple choice questions, pick the single response that best represents the correct answer. If the question appears confusing, difficult, or incorrect, skip it and move on, and come back to it later. If we discover that the question has errors, we will discard it during the grading process.

If you need assistance during the exam itself, you can connect to the **zoom channel** **(https://illinois.zoom.us/j/86068073561?pwd=TWs1WUswa2FkaXV6M2JyTjJtRlNNUT09&from=addon)** where staff will be available during the exam time. Be prepared to share your screen in order for us to answer questions about specific problems. The zoom link can be also found on the course wiki page.

Most important, do not submit your exam until you answer all questions. You only have one attempt, and once the exam is submitted, you will not be able to return to it.

Good luck!

Question 1 — 5 pts

We need to calculate the histogram of an array with 10^9 elements. The histogram has four bins. Assume that each atomic operation in the global memory has a constant total latency of 100ns and each atomic operation in the shared memory has a constant total latency of 1ns. Further, assume that when we launch the kernel, blockDim = 10^3 and gridDim = 10^5, thus each thread is responsible for 10 elements. If we only consider the latencies caused by atomic operations, what is the theoretical minimum runtime if privatization is implemented and the elements of the array have an access distribution of (50%, 30%, 10%, 10%)? Suppose that there is only one global histogram and one shared memory histogram in each block.

- (●) $(10^7 + 5000)$ ns
- ( ) $(4 * 10^7 + 10^4)$ ns
- ( ) $(10^7 + 10^4)$ ns

- ○ $(4 * 10^7 + 1000)$ ns
- ○ $(4 * 10^7 + 5000)$ ns
- ○ None of these answers are correct
- ○ $(10^7 + 1000)$ ns

Question 2

5 pts

Consider the following sparse Matrix:

$$\begin{bmatrix} 1 & 0 & 4 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 7 & 0 & 9 & 3 \\ 0 & 0 & 0 & 0 & 0 \\ 6 & 5 & 0 & 0 & 8 \end{bmatrix}$$

For each of the following data layouts in memory, select the option that best matches all the sparse matrix formats that can store the data in memory as depicted.

Layout 1:

1	4	2	7	9	3	6	5	8

Layout 2:

1	2	7	6	4	0	9	5	0	0	3	8

Layout 3:

7	9	3	6	5	8	1	4	2

Layout 4:

9	7	1	2	4	3	5	8	6

Layout 5:

7	6	1	2	9	5	4	3	8

- ○ **Layout 1: JDS**
  Layout 2: CSR
  Layout 3: COO
  Layout 4: JDS-Transposed
  Layout 5: ELL

- ○ **Layout 1: CSR, COO**
  Layout 2: ELL
  Layout 3: JDS, COO
  Layout 4: COO
  Layout 5: JDS-Transposed, COO

- ○ **Layout 1: CSR**
  Layout 2: ELL
  Layout 3: JDS
  Layout 4: COO
  Layout 5: JDS-Transposed

- ◉ None of these answers are correct.

- ○ **Layout 1: JDS-Transposed**
  Layout 2: ELL
  Layout 3: COO
  Layout 4: CSR
  Layout 5: JDS

Question 3

5 pts

W is the convolution filter weight tensor, organized as tensor W[M, C, K, K], where M is the number of output feature maps, C is the number of input feature maps and K is the height and width of each filter. Tensors are stored as multi-dimensional arrays in the memory. X is the input feature map, organized as a tensor X[C, H, W], where H is the height of each input feature map and W is the width of each input feature map. Y is the output feature map, organized as a tensor Y[M, H_out, W_out], where H_out is the height of each output feature map and W_out is the width of each output feature map.

What are H_out and W_out in terms of M,C,K,H,W?

- ◉ H_out = H-K+1
  W_out = W-K+1

- ○ None of these answers are correct

- ○ H_out = H-K-1
  W_out = W-K-1

- ○ H_out = H+K-1
  W_out = W+K-1

- ○ H_out = H-K

W_out = W-K

Question 4

5 pts

Assume you want to use the GPU to compute the distribution of a series of non-negative integers into a large number of bins shown as [0, 3], [4, 7], [8, 11], …, such that each interval has a size of 4. Since the shared memory is limited, you decide to use the shared memory to keep the first 256 bins and the global memory for the rest of the bins. Whenever a number doesn't belong to the first 256 bins, you will increment the global bin for it. Assume that there is enough global memory to accommodate the rest of the bins and all global histogram elements are initialized to 0. Also, assume that the block size in 256. Fill in the blanks in the code provided below.

```
dim3 gridDim(8);
dim3 blockDim(256);

__global__ void histo_kernel(unsigned int *numbers, long size, unsigned int *histo)
{
    __shared__ unsigned int histo_private[256];
    // Reset histogram
    if (threadIdx.x < 256)
        histo_private[____A____] = 0;

    __syncthreads();

    int i = threadIdx.x + blockIdx.x * blockDim.x;
    // stride is total number of threads
    int stride = ____B____;

    while (i < size) {
        if ( numbers[i] < ____C____ ) atomicAdd(&histo_private[numbers[i]/4], 1);
        else atomicAdd(&histo[numbers[i]/4], 1);
        i += stride;
    }
    __syncthreads();

    // contribute to global histogram
    if (threadIdx.x < 256)
        atomicAdd(____D____, ____E____);
}
```

- ○ A: threadIdx.x
  B: blockDim.x
  C: 1024
  D: &histo[numbers[i]]
  E: &histo_private[numbers[i]]

- ◉ A: threadIdx.x
  B. blockDim.x * gridDim.x
  C: 256
  D: &histo[numbers[i]/4]

E: &histo_private[numbers[i]/4]

○ A: blockIdx.x
B. blockDim.x
C: 1024
D: &histo[numbers[i]]
E: &histo_private[numbers[i]]

○ A: threadIdx.x
B: blockDim.x * gridDim.x
C: 1024
D: &histo[numbers[i]/4]
E: &histo_private[numbers[i]/4]

○ None of these solutions are correct

Question 5

7 pts

You are tasked with performing some operations on a very long vector. Upon profiling your kernel code with nsys, you realize that a significant portion of the execution time is taken up by serial memory transfers to and from the device. You decide to use CUDA streams to overlap the memory transfers with computation to decrease the total execution time. There is a constant overhead for starting a kernel computation on the GPU, which is independent of the size of the data that the kernel is processing. The following diagram illustrates the timeline for a single stream execution (not to scale).

The times for each of the section shown above is as follows:

- Total Input Transfer time: 10 sec
- Constant Kernel Launch overhead: 1 sec
- Total Compute time: 10 sec
- Total Output Transfer Time: 10 sec

Assume that you have access to 3 hardware streams that enable continuous pipelining. How many segments should the input vector be divided into so as to minimize the total execution time where continuous pipelining is possible? Choose the best option.

○ 8 segments

○ 4 segments

○ 2 segments

- (●) 16 segments
- ( ) 32 segments

Question 6

7 pts

Assume you want to implement a forward convolution layer in a CNN. In order to optimize the memory bandwidth, you decide to unroll your input features by replicating them in a new matrix of size S = X * Y. Each input feature element is replicated R times. Your convolution filters are 3 * 3 and there are 3 input feature maps of size 30 * 30. What are the values of S and R?

- (●) **S: 9 * 28 * 28, R: 7.84**
- ( ) **S: 3 * 28 * 28, R: 9**
- ( ) None of these answers are correct
- ( ) **S: 27 * 28 * 28, R: 7.84**
- ( ) **S: 3 * 28 * 28, R: 9**

Question 7

7 pts

Given matrix A, which of the following are correct?

$$A = \begin{vmatrix} 1 & 0 & 2 & 3 \\ 3 & 0 & 4 & 0 \\ 0 & 0 & 4 & 0 \\ 5 & 0 & 0 & 6 \end{vmatrix}$$

i) CSR representation

Data = [1,2,3,3,4,4,5,6]
Col_idx = [0,2,3,0,2,2,0,3]

Row_ptr = [0,3,5,6,8]

ii) ELL representation

Data = [1,2,3,3,4,4,5,6]
Col_idx = [0,2,3,0,2,2,0,3]
Row_idx = [0,0,0,1,1,2,3,3]

iii) JDS representation

Data = [1,2,3,3,4,5,6,4]
Col_idx = [0,2,3,0,2,0,3,2]
Row_ptr = [0,3,5,7,8]
Row_idx = [0,1,3,2]

iv) JDSt representation

Data = [1,3,5,4,2,4,6,3]
Col_idx = [0,0,0,2,2,2,3,3]
Col_ptr = [0,4,7,8,8]
Row_idx = [0,1,3,2]

- ○ i and iii only
- ◉ i, iii and iv only
- ○ i and ii only
- ○ None of these answers are correct
- ○ i, ii and iii only
- ○ i, ii, iii, and iv

Question 8 — 8 pts

Suppose you want to try a different approach to write the reduction kernel for calculating block sums in your MP 5.1. Given a long input array of floating point elements (you may assume the input array contains at most 2048 x 65535 elements so that it can be handled by only one kernel launch), your reduction kernel calculates the sum of all elements in each block and returns an array of block-wise partial sums, then let the CPU host code do the summation on the output array of your kernel to calculate the sum of all elements in the original long input array. In MP5.1 you have used the improved approach with reduced control divergence that you have learned in our lectures. But now, instead of letting the first few threads to remain active in the last several iterations, you want to let the last few threads to remain active in the last several iterations. What is the correct answer for the blanks, (1), (2), and (3) in the kernel code below?

```
__global__ void total(float *input, float *output, int len)
{
    __shared__ float partialSum[2*BLOCK_SIZE];

    // Load data into shared memory.
    if((blockIdx.x * blockDim.x * 2 + threadIdx.x) < len) {
      partialSum[threadIdx.x] = input[blockIdx.x * blockDim.x * 2 + threadIdx.x];
    }
    else {
      partialSum[threadIdx.x] = 0;
    }
    if((blockIdx.x * blockDim.x * 2 + blockDim.x + threadIdx.x) < len) {
      partialSum[blockDim.x + threadIdx.x] = input[blockIdx.x * blockDim.x * 2 + blockDim.x + threadIdx.x];
    }
    else{
      partialSum[blockDim.x + threadIdx.x] = 0;
    }

    // Calculate sum of current block.
    for(int stride=blockDim.x; stride>=1; stride/=2){
      __syncthreads();
      if(____1____) {
        partialSum[blockDim.x + threadIdx.x] += partialSum[blockDim.x + threadIdx.x - stride];
      }
    }

    __syncthreads();

    // Only one thread in each block writes the result of the current block sum to output.
    if(____2____) {
       output[blockIdx.x] = partialSum[____3____];
    }
}
```

- (1): blockDim.x + threadIdx.x <= stride
  (2): threadIdx.x == blockDim.x + 1
  (3): blockDim.x + threadIdx.x - 1

- (1): blockDim.x <= stride - threadIdx.x
  (2): threadIdx.x - blockDim.x == 0
  (3): blockDim.x

- (1): blockDim.x - threadIdx.x > stride
  (2): threadIdx.x == blockDim.x

```
(3): blockDim.x - threadIdx.x
```

○
```
(1): blockDim.x - threadIdx.x <= stride
(2): threadIdx.x == blockDim.x - 1
(3): blockDim.x + threadIdx.x
```

○ None of these answers are correct

○
```
(1): threadIdx.x <= stride + blockDim.x
(2): threadIdx.x - 1 == blockDim.x
(3): threadIdx.x
```

Question 9

7 pts

Consider a kernel performing the element wise vector operation: **D = A* B + C** where A, B, C, D are one dimensional vectors. These vectors are very long, so you decide to decrease the total execution time by overlapping the memory transfers with kernel computations using CUDA streams. The skeleton code to do so is shown below. Which of the following sequences of operations in the 'for' loop will produce the **fastest** overall execution time with the **correct** output?

```
cudaStream_t stream0, stream1;
cudaStreamCreate( &stream0);
cudaStreamCreate( &stream1);

float *d_A0, *d_B0, *d_C0, *d_D0; // device memory for stream 0
float *d_A1, *d_B1, *d_C1, *d_D1; // device memory for stream 1

// Cuda Malloc for d_A0, d_B0, d_C0, d_D0, d_A1, d_B1, d_C1, d_D1

for (int i=0; i<n; i+=SegSize*2)  {
   // Which option produces the fastest and correct execution?
}
```

Statement Mappings:

```
A0_HtoD(stream) : cudaMemcpyAsync(d_A0, h_A+i, SegSize*sizeof(float),.., stream);
B0_HtoD(stream) : cudaMemcpyAsync(d_B0, h_B+i, SegSize*sizeof(float),.., stream);
C0_HtoD(stream) : cudaMemcpyAsync(d_C0, h_C+i, SegSize*sizeof(float),.., stream);
D0_DtoH(stream) : cudaMemcpyAsync(h_D+i, d_D0, SegSize*sizeof(float),.., stream);
```

```
A1_HtoD(stream) : cudaMemcpyAsync(d_A1, h_A+i+SegSize, SegSize*sizeof(float),.., stream);
B1_HtoD(stream) : cudaMemcpyAsync(d_B1, h_B+i+SegSize, SegSize*sizeof(float),.., stream);
C1_HtoD(stream) : cudaMemcpyAsync(d_C1, h_C+i+SegSize, SegSize*sizeof(float),.., stream);
D1_DtoH(stream) : cudaMemcpyAsync(h_D+i+SegSize, d_D1, SegSize*sizeof(float),.., stream);
```

```
VecKernel_0(stream): vecMulAdd<<<SegSize/256, 256, 0, stream>>>(d_A0, d_B0, …);
VecKernel_1(stream): vecMulAdd<<<SegSize/256, 256, 0, stream>>>(d_A1, d_B1, …);
```

- (●) **A0_HtoD(stream0), B0_HtoD(stream0), C0_HtoD(stream0), A1_HtoD(stream1), B1_HtoD(stream1), C1_HtoD(stream1), VecKernel_0(stream0), VecKernel_1(stream1), D0_DtoH(stream0), D1_DtoH(stream1)**
- ( ) **A0_HtoD(stream0), B0_HtoD(stream0), C0_HtoD(stream0), VecKernel_0(stream0), D0_DtoH(stream0), A1_HtoD(stream1), B1_HtoD(stream1), C1_HtoD(stream1), VecKernel_1(stream1), D1_DtoH(stream1)**
- ( ) None of these answers are correct.
- ( ) **A0_HtoD(stream0), B0_HtoD(stream0), C0_HtoD(stream0), A1_HtoD(stream1), B1_HtoD(stream1), C1_HtoD(stream1), VecKernel_0(stream0), D0_DtoH(stream0), VecKernel_1(stream1), D1_DtoH(stream1)**
- ( ) **A0_HtoD(stream0), B0_HtoD(stream0), C0_HtoD(stream0), D0_DtoH(stream0), A1_HtoD(stream1), B1_HtoD(stream1), C1_HtoD(stream1), VecKernel_0(stream0), VecKernel_1(stream1), D1_DtoH(stream1)**

Question 10

7 pts

Suppose we use Brent-Kung in a hierarchical approach to perform parallel scan on a 1D input array of 2^{42} elements. We use 2^{10} =1024 threads per block in all our Brent-Kung kernels and our GPU supports at most 2^{11} = 2048 blocks per grid. Remember that in the Brent-Kung kernel, each block processes 2048 elements. Select the best answer pair for the following two questions:

Question 1: What is the best approximation of the number of floating-point add operations performed per thread block in the reduction and post scan steps in the Brent-Kung kernel (summed together)?

Question 2: What is the best approximation of the minimum number of times we need to launch the Brent-Kung kernel?

- ( ) Q1: 2048 x 2

  Q2: $2^{20}+2^{9}+2$
- ( ) Q1: 2048 x 2

  Q2: 3
- ( ) Q1: 2048

  Q2: 3

- Q1: 2048 x 2
  Q2: $2^{31}+2^{20}+2^{9}+2$
- Q1: 2048 x 1024
  Q2: $2^{31}+2^{20}+2^{9}+2$
- Q1: 2048 x 1024
  Q2: $2^{20}+2^{9}+2$
- Q1: 2048
  Q2: $2^{31}+2^{20}+2^{9}+2$
- Q1: 2048 x 1024
  Q2: 3
- **Q1: 2048**
  Q2: $2^{20}+2^{9}+2$

Question 11

7 pts

In this question, we will examine the idea of privatization and see how it helps save global memory bandwidth. Imagine we want to find out the frequency of letters or digits appeared in an alphanumeric (consisting of both letters and digits) string with a length of 10000. Assume that we are using blockDim(128, 1, 1) and each block could process 2 * BLOCK_SIZE of elements. Calculate the ratio of total number of global atomics operations before and after we apply this privatization technique, given the scenario of case sensitive and case insensitive.

- **10000 / 40 and 10000 / 20**

- ○ 10000 / 2480 and 10000 / 1440
- ○ None of these answers are correct.
- ○ 10000 / 4960 and 10000 / 2880
- ○ 10000 / 80 and 10000 / 40

Question 12

10 pts

Given a heterogeneous CPU-GPU system, you are asked to deploy a benchmark program to calibrate the performance of the GPU. In this benchmark, all the parallel workloads are multiply-add operations. The manual of the single-core CPU tells you that each multiply-add operation requires 10 CPU cycles to complete, and both GPU and CPU are running under the same clock at 2 GHz. Part of the manual for the GPU is missing, and you are told that each multiply-add operation requires 250 GPU cycles to complete. Each warp has 32 threads and each Streaming Multiprocessor (SMs) could take up to 1536 threads and across no more than 8 blocks or 32 warps.

The benchmark program has the following characteristic:

- Parallel workload / Sequential workload = 3
- Launch with blockDim (64, 2, 1) with GridDim that allows maximum occupancy.
- All data on the GPU have been preloaded.

Now, you observe a speed up of ~3.94225 compared to running on the same single-core CPU only. How many Streaming Multiprocessor (SMs) are there on the GPU? (You may need to round to the nearest integer)

- ◉ 5
- ○ 4
- ○ 3
- ○ 2

Question 13

10 pts

In GPU convolution, assume we have input X with shape (B, C, H, W) = (1, 3, 28, 28), where B is the batch size, C is the number of input channels, H is the height of each input feature map, and W is the width of each input feature map; Y with shape (B, M, H_out, W_out) = (1, 5, 24, 24), where M is the number of output channels, H_out is the height of output feature map, W_out is the width of output feature map; and filter with shape (M, C, K, K) = (6, 3, 5, 5). In the forward propagation, besides using regular convolution, we can also reduce the convolution into the general matrix multiplication (GEMM) with unrolling, as we covered in lectures. In the implementation of unrolling and GEMM, assume we use one CUDA kernel for unrolling and another kernel for GEMM. Input X, output Y, and the filter are all arrays of 32-bit floating-point numbers. Which is the closest to the minimum value of **(total Bytes allocated in global memory in convolution using unrolling and GEMM) / (total Bytes allocated in global memory in regular convolution)**?

- ○ 1.0
- ◉ 8.0
- ○ 4.0
- ○ 16.0
- ○ 2.0

Question 14

5 pts

AMD GPUs will always be inferior to Nvidia GPUs because they don't have shared memory.

- ◉ False
- ○ True

Question 15

5 pts

In DPC++, a single source code file cannot contain both CPU host code and GPU device code.

- ◉ True

○ False

No new data to save. Last checked at 7:57pm Submit Quiz