



ECE 408 Exam 2, Fall 2017

December 12th, 2017

- 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 consider giving everyone up to 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.

Good luck!

Name: _____

Netid: _____

UIN: _____

Question 1: _____

Question 2: _____

Question 3: _____

Question 4: _____

Question 5: _____

Question 6: _____



Name: _____

Question 1 (30 points, 40 minutes): multiple-choice and short-answer questions. If you get more than 30 points by answering all questions (1-10), your score will saturate at 30 points. The bonus question is extra.

For multiple-choice questions, give a concise explanation for your answer for possible partial credit. Answer each of the short-answer questions in as few words as you can. Your answer will be graded based on completeness, correctness, and conciseness.

1. (3 points) For the Brent-Kung scan kernel based on reduction trees and inverse reduction trees, assume that we have 2048 elements, which of the following values is the closest to 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) $(2048-1)*2 - 11$
(C) $1024*2048$
(D) $2048*11$

2. (4 points) 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 64?
(A) 0
(B) 1
(C) 16
(D) 32

3. (4 points) For a processor that supports atomic operations in L2 cache, assume that each atomic operation takes 5ns to complete in L2 cache and 500ns to complete in



Name: _____

DRAM. Assume that 99.9% of the atomic operations hit in L2 cache. What is the approximate throughput for atomic operations on the same global memory variable?

- (A) 1/500 G atomic operations per second
- (B) 1/50 G atomic operations per second
- (C) 1/5 G atomic operations per second
- (D) 1/13.6 G atomic operations per second

4. (3 points) In Question 3, if a kernel performs 10 floating-point operations per atomic operation, what is the approximate maximal floating-point throughput of the kernel execution as limited by the throughput of the atomic operations?
- (A) 1 GFLOPS
 - (B) 2 GFLOPS
 - (C) 10 GFLOPS
 - (D) 50 GFLOPS
5. (3 points) Given a sparse matrix of integers with R original rows, L non-zero elements in the original row with the largest number of non-zeros, and a total of N non-zeros. How many integers are needed to represent the matrix in JDS-T? Assume that we keep track of the number of non-zeros in each original row, as we specified in the MP assignment.
- (A) $R+L+N$
 - (B) $2R+L+N+1$
 - (C) $2R+L+2N$
 - (D) $2R+2L+N$
6. (4 points) For a sparse matrix-vector multiplication (SpMV) with R rows, a total of N non-zero elements, and a maximal of L non-zeros in each row, how many times is each matrix element used?



Name: _____

- (A) 1
- (B) 2
- (C) R
- (D) L

7. (4 points) For a sparse matrix-vector multiplication (SpMV) with R rows, C columns, a total of N non-zero elements, how many times will each vector element be used?

- (A) 1
- (B) N/R on average
- (C) N/C on average
- (D) R

8. (4 points) Keven has a 320MB array that he would like to process with GPU. He measured that the execution time of the code on CPU was 0.2 seconds. He also implemented a kernel and measured that the kernel execution on the GPU with the data in the GPU memory was 0.004 seconds, a 50x speedup! However, he needs to transfer the data into the GPU memory and transfer 320MB 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) 5x speedup
- (D) 1.5x slow down

9. (2 points) For the following host code sequence:

```
1) cudaMemcpyAsync(d_A0, h_A+i, SegSize*sizeof(float),..., stream0);
2) cudaMemcpyAsync(h_C+i, d_C0, SegSize*sizeof(float),..., stream0);
3) cudaMemcpyAsync(d_A1, h_A+i+SegSize, SegSize*sizeof(float),..., stream1);
4) cudaMemcpyAsync(h_C+i+SegSize, d_C1, SegSize*sizeof(float),..., stream1);
5) cudaMemcpyAsync(d_A2, h_A+i+2*SegSize, SegSize*sizeof(float),..., stream2);
6) cudaMemcpyAsync(h_C+i+2*SegSize, d_C2, SegSize*sizeof(float),..., stream2);
```



Name: _____

Which of the statements could be executed in parallel on the GPU

- (A) 1) and 2)
- (B) 2) and 3)
- (C) 1) and 3)
- (D) 2) and 4)

10. (2 points) When your parallel reduction kernel generates a slightly different result than a sequential reduction function for a `float` input array, what would be the most likely reason?

- (A) The hardware failed
- (B) There is a missing `__syncthreads()` call
- (C) Floating-point operations are not necessarily commutative nor associative
- (D) CPU and GPUs have different precision for `float` numbers

(Bonus 2 points) List the errors and typos that you reported via Piazza postings, e-mails, or in person communication with Prof. Hwu or Carl Pearson. (0.5 points for each item.)

Question 2 (15 points, suggested time allocation 20 minutes): This question tests your understanding of parallel histogram computation and privatization.

You are the owner of a supermarket and want to know the distribution of the price tags of your merchandise. So you decided to build a histogram with intervals of \$5(e.g $\$0 \leq \text{histo}[0] < \5 , $\$5 \leq \text{histo}[1] < \10 and so on) on a GPU. However, the shared memory can only accommodate 256 bins for each block. As a compromise, you decide to privatize the first 256 of the bins into the shared memory. Whenever the data value doesn't fall in the first 256 bins, you



Name: _____

will have to increment the global bins. Assume that the prices are integer values and the **global histogram array is sized to accommodate the prices of all the items**. Also, assume that all global histogram elements have been initialized to zero before the kernel is launched.

(A) (3 Points) Complete the following kernel to implement the partial privatization of the histogram.

```

1.  /* histo_kernel is launched with the following parameters
2.
3.  dim3 gridDim(8);
4.  dim3 blockDim(256);
5.  */
6.
7.  __global__ void histo_kernel(unsigned int *prices, long size, unsigned int *histo){
8.  __shared__ unsigned int histo_private[256];
9.
10.  // Reset histogram
11.  histo_private[ ] = 0;
12.  __syncthreads();
13.
14.  int i = threadIdx.x + blockIdx.x * blockDim.x;
15.  // stride is total number of threads
16.  int stride = blockDim.x * gridDim.x;
17.
18.  while (i < size) {
19.      if ( prices[i] < _____ ) atomicAdd(_____, 1);
20.      else atomicAdd(_____, 1);
21.      i += stride;
22.  }
23.  __syncthreads();
24.
25.  // contribute to global histogram
26.  atomicAdd( _____, _____);
27. }
```

(B) (4 Points) how many **non-atomic global memory reads/writes** and **shared-memory/global-memory atomic operations** are being performed by all the threads executing your kernel if there are **4096** items in total which are all priced less than \$1280?

Non-atomic Global Memory reads: _____

Non-atomic Global Memory writes: _____

Shared-memory atomic operations: _____

Global-memory atomic operations: _____

Explanation:



Name: _____

(C) (4 Points) how many **non-atomic global memory reads/writes** and **shared-memory/global-memory atomic operations** are being performed by all the threads executing your kernel if there are **4000** items in total which are all priced less than \$1280?

Non-atomic Global Memory reads: _____

Non-atomic Global Memory writes: _____

Shared-memory atomic operations: _____

Global-memory atomic operations: _____

Explanation:

(D) (4 Points) how many **non-atomic global memory reads/writes** and **shared-memory/global-memory atomic operations** are being performed by your kernel if there are 4000 items are priced less than \$1280 and 96 items are priced above \$1280?

Non-atomic Global Memory reads: _____

Non-atomic Global Memory writes: _____

Shared-memory atomic operations: _____

Global-memory atomic operations: _____

Explanation:



Name: _____

Question 3: Parallelization (15 points, suggested time allocation 25 minutes):

Consider the **dense** matrix-vector multiplication $\mathbf{Ax} = \mathbf{b}$. The i^{th} element of \mathbf{b} is the dot product of \mathbf{x} with the i^{th} row of \mathbf{A} .

\mathbf{A} is a pointer to a $\text{numRows} \times \text{numCols}$ row-major matrix,

\mathbf{b} is a pointer to a vector of length numRows with all entries initialized to 0, and

\mathbf{x} is a pointer to a vector of length numCols .

The following is a CPU sequential code that needs to be parallelized:

```
void mv_cpu(float *b, const float *A, const float *x, const int numRows,
            const int numCols) {
    for (int colIdx = 0; colIdx < numCols; ++colIdx) {
        for (int rowIdx = 0; rowIdx < numRows; ++rowIdx) {
            b[rowIdx] += A[rowIdx * numCols + colIdx] * x[colIdx];
        }
    }
}
```

Your colleague proposes the following three parallelizations to take advantage of GPU parallelism:

```
1. /* mv_gpu_1 launched with the following parameters
2.   dim3 gridDim(10);
3.   dim3 blockDim(256);
4. */
5. __global__ void mv_gpu_1(float *b, const float *A, const float *x,
6.                          const int numRows, const int numCols) {
7.     const int rowStart = blockIdx.x * blockDim.x + threadIdx.x;
8.
9.     for (int r = rowStart; r < numRows; r += blockDim.x * gridDim.x) {
10.         float dot = 0;
11.         for (int c = 0; c < numCols; ++c) {
12.             dot += A[r * numCols + c] * x[c];
13.         }
14.         b[r] = dot;
15.     }
16. }
```

```
1. /* mv_gpu_2 launched with the following parameters
2.   dim3 gridDim(16);
3.   dim3 blockDim(256);
4. */
5. __global__ void mv_gpu_2(float *b, const float *A, const float *x,
6.                          const int numRows, const int numCols) {
7.     int colStart = blockIdx.x * blockDim.x + threadIdx.x;
8.
9.     for (int c = colStart; c < numCols; c += blockDim.x * gridDim.x) {
10.         const float v = x[c];
11.         for (int r = 0; r < numRows; ++r) {
```




Name: _____

```

12.     atomicAdd(&b[r], A[r * numCols + c] * v);
13. }
14. }
15. }

1. /* mv_gpu_3 launched with the following parameters
2. dim3 gridDim(4, 4);
3. dim3 blockDim(16,16);
4. */
5. template <size_t BS> // Assume that BS is 16
7. __global__ void mv_gpu_3(float *b, const float *A, const float *x,
8.                         const int numRows, const int numCols) {
9.
10.    __shared__ float b_s[BS];
11.    const int tx = threadIdx.x;
12.    const int ty = threadIdx.y;
13.    const int colStart = blockIdx.x * blockDim.x + tx;
14.    const int rowStart = blockIdx.y * blockDim.y + ty;
15.
16.    for (int r = rowStart; r < numRows; r += gridDim.y * blockDim.y) {
17.        if (ty == 0)
18.            b_s[tx] = 0;
19.        __syncthreads();
20.
21.        for (int c = colStart; c < numCols; c += gridDim.x * blockDim.x) {
22.            atomicAdd(&b_s[ty], A[r * numCols + c] * x[c]);
23.            __syncthreads();
24.        }
25.
26.        if (tx == 0)
27.            atomicAdd(&b[r], b_s[ty]);
28.        __syncthreads();
29.    }
30. }

```

For each column of the following table, choose entries from the first row of that table that apply to the kernel in question.

- The first column should contain only A-D, and the second column should contain only E-J.

Name: _____

- Each box may contain 0, 1, or more than one letter.

(Hint) It might be more efficient to focus on one of (A)-(J) and determine if it is applicable to mv_gpu_1 through mv_gpu_3 before moving to the next one.

	could be fast because	could be slow because
Possible choices for this column	(A) Coalesced memory accesses to A (B) numRows global memory writes (C) Many active threads if numCols is large (D) Many active threads if numRows is large	(E) contention in shared memory atomics (F) Barrier synchronization (G) numRows & numCols global memory writes (H) redundant loads from X (I) uncoalesced memory accesses to A (J) contention in global memory atomics
mv_gpu_1		
mv_gpu_2		
mv_gpu_3		

Question 4. Sparse Matrix Multiplication (10 points, suggested time allocation 25 minutes):

This question tests your knowledge of Sparse Matrix representation and operation. For your convenience, we are enclosing an example that illustrates how a dense matrix is transferred to the sparse matrix in ELL format. Recall that we take CSR, pad elements to make all rows of equal length, and transpose the padded matrix.



Name: _____

Dense Matrix:

Row 0	3	0	1	0
Row 1	0	0	0	0
Row 2	0	2	4	1
Row 3	1	0	0	1

Sparse Matrix in ELL:

data:

3	*	2	1	1	*	4	1	*	*	1	*
---	---	---	---	---	---	---	---	---	---	---	---

col_index:

0	*	1	0	2	*	2	3	*	*	3	*
---	---	---	---	---	---	---	---	---	---	---	---

num_elem = 3

num_rows = 4

(A) (4 points) In the following ELL kernel, fill in the missing indexing expressions for accessing data (input matrix), x (input vector) and y (output vector).

```

1. __global__ void SpMV_ELL (int num_rows, float *data, int *col_index, int
num_elem, float *x, float *y) {
2.   int row = blockIdx.x * blockDim.x + threadIdx.x;
3.   if (row < num_rows) {
4.     float dot = 0;
5.     for (int i = 0; i < num_elem; i++) {
6.       dot += data[_____] * x[_____];
7.     }
8.     y[row] = dot;
9.   }
10. }
```

(B) (4 points) After we transform the matrix into ELL layout as the example shows, and launch the kernel in (A), circle one answer for each question below and **justify your choice**.

(1) Is there any control divergence (assuming num_rows is a multiple of 32)?

Answer: **Yes** or **No**

Explanation:



Name: _____

(2) Is memory access coalesced?

Answer: **Yes** or **No**

Explanation:

(C) (2 points) State what the hybrid ELL-COO format is and give a situation when hybrid ELL-COO method performs better than the ELL format.



Name: _____

Question 5. Convolution Neural Network (15 points, suggested time allocation 15 minutes):

A basic convolution layer consists of filter W , input X and output Y . We want to accelerate the forward propagation of convolution layers in the training process.

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. Tensors are stored as multi-dimensional arrays in the memory.

X is the input feature map, organized as a tensor $X[B, C, H, W]$, where B is the number of images in one mini-batch (recall that mini-batch is used to efficiently updates gradients while keeping relatively fast convergence), 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 $Y[B, M, H_{out}, W_{out}]$, where $H_{out} = H - K + 1$ is the height of each output feature map and $W_{out} = W - k + 1$ is the width of each output feature map.

- (A) You first start with a naïve implementation for the first convolutional layer given that an output feature map of $H_{out} \times W_{out}$ (28×28) can fit in one block (so that each thread is processing one element in the output). You need to fill in the missing parts so that the convolution layer is complete.

```
// assume that the kernel will be launched with the following configuration
01: dim3 gridDim(B, M, C);
02: dim3 blockDim(W_out, H_out, 1);
03:
04: __global__ void convLayerForward_Naive(int C, int K, int W_out, int H_out,
                                         float* X, float* W, float* Y){
05:     int b = _____;
06:     int m = _____;
07:     int c = _____;
08:     int h = _____;
09:     int w = _____;
10:
11:     float acc = 0;
12:     for (int p = 0; p < K; p++) // KxK filter
13:         for (int q = 0; q < K; q++)
14:             acc += X[b, c, _____, _____] * W[m, c, p, q];
15:
16:     Y[b, m, h, w] = acc;
17: }
```



Name: _____

- (B) You then find out that the x, y, z dimension in the thread blocks can be mapped arbitrarily. Suppose now you change blockDim to blockDim(H_out, W_out, 1), which line(s) in the kernel of part (1) need to be changed in order to be an correct implementation?

New blockDim:

```
01: dim3 blockDim(B, M, C);  
02: dim3 blockDim(H_out, W_out, 1);
```

Line(s) need to be changed (You may not need to fill in all the lines):

Line _____, change to: _____.
Line _____, change to: _____.
Line _____, change to: _____.
Line _____, change to: _____.
Line _____, change to: _____.

- (C) Comparing the kernel in part (2) to the kernel in part (1), would you expect the performance of kernel in part (2) to increase, decrease or stay the same? Explain why.



Name: _____

Question 6. Scan (15 points, suggested time allocation, 25 minutes): You are a new hire at a Parallelism for Cheap Inc. The company specialises in creating parallel algorithms for machines with cheaper hardware. Your boss heard you took Applied Parallel Programming with the Wen-Mei Hwu and wants you show your expertise. Your boss wants you to make some changes to the Brent-Kung algorithm to further improve the efficiency of the hardware use.

You reason that you can further improve the execution efficiency by using fewer threads to do more work. For example, instead of using 1024 threads in each block to process 2048 elements in each section, you would like to use 64 threads in each block. To do this you break up the work at each level of the reduction and post-scan steps into parts.

At each level, all threads in the block will process one part of their section of input elements before moving onto the next part. For example, at the first level of the reduction tree, all 64 threads will process the first 128 elements (64 pairwise additions) as part 0. They will then move to process the next 128 elements as part 1. So the threads will iterate through 16 parts for the first level of the reduction tree.

(A) (5 Points) With 2048 elements in each section, about how many times more computations will each thread be doing compared to a regular Brent-Kung thread on average? Answer should be in the form 1 times, 5 times, 100 times etc. Explain for full credit.

(B) (5 Points) Below is the skeleton code for implementing this strategy. The given code is very similar to your MP. Fill in the missing conditionals and indexing to make the code run as described in part A. Note that you may leave a blank empty if you feel nothing should go there. No explanation is necessary.

```

01: #define BLOCK_SIZE 64
02: #define SECTION_SIZE 2048
03: __global__ void scan(float *input, float *output, int len) {
04:     __shared__ float shared[SECTION_SIZE];
05:     int bx = blockIdx.x;
06:     int tx = threadIdx.x;
07:     int i = bx * SECTION_SIZE + tx;
08:
09:     for(int part = 0; part < SECTION_SIZE / BLOCK_SIZE; part++) {
10:         if(i + part * BLOCK_SIZE < len)
11:             shared[tx + part * BLOCK_SIZE] = input[i + part * BLOCK_SIZE];
12:         else
13:             shared[tx + part * BLOCK_SIZE] = 0;
14:     }
15:
16:     for(unsigned int stride = 1; stride < SECTION_SIZE; stride *= 2) {
17:         __syncthreads();
18:         for(int part = 0; part < _____; part++){
19:             int index = (tx+1+_____) * (stride *2) - 1 _____;

```



Name: _____

```
20:     if(index < SECTION_SIZE)
21:         shared[index] = shared[index] + shared[index - stride];
22:     }
23: }
24:
25: for(unsigned int stride = (SECTION_SIZE)/4; stride > 0; stride /= 2) {
26:     __syncthreads();
27:     for(int part = 0; part < _____; part++){
28:         int index = (tx+1+_____) * (stride *2) - 1 _____;
29:         if(index + stride < SECTION_SIZE)
30:             shared[index + stride] = shared[index + stride] + shared[index];
31:     }
32: }
33:
34: __syncthreads();
35: for(int part = 0; part < SECTION_SIZE / BLOCK_SIZE; part++){
36:     if(i + part * BLOCK_SIZE < len)
37:         output[i + part * BLOCK_SIZE] = shared[tx + part * BLOCK_SIZE];
38: }
39: }
```




Name: _____

- (C) (5 Points) Does this strategy optimally use the hardware efficiently in terms of active threads and control divergence? Explain why or why not. If it does not, how could you improve the strategy? Your answer must begin with yes or no followed by an explanation.