Trial Version Wondershare PDFelement

ECE408/CS483/CSE408 Exam #1, Fall 2017

Tuesday, October 17, 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 170 minutes to complete. To eliminate time pressure and 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 up to and including parallel scan.
- You can write down the reasoning behind your answers for possible partial credit.
- **Good luck!**

Trial Version Wondershare PDFelement

Question 1 (30 points, suggested time allocation 40 minutes): multiple-choice and short-answer questions. If you get more than 30 points by answering all questions (1-9), 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. (4 points) We want to use each thread to calculate four (4) output elements of a vector addition. Each block processes 4*blockDim.x consecutive elements that form 4 sections. All threads in each block will first process a section with each thread processing one element. They will then all move to the next section with each thread processing one element. For each section, consecutive threads should process consecutive elements. What would be the kernel code expression for forming the value of i, the data index of the first element to be processed by each thread?
(A) i=blockIdx.x*blockDim.x *4;
(B) i=blockIdx.x*threadIdx.x + threadIdx.x
(C) i=blockIdx.x*blockDim.x*4 + threadIdx.x
(D) i=blockIdx.x*blockDim.x + threadIdx.x*4

2. (4 points) For vector addition, assume that the vector length is 8,000, each thread calculates four (4) output elements, and the thread block size is 1,024 threads. The programmer configures the kernel launch to have a minimal number of blocks to cover all output elements. How many threads will be in the grid?
(A) 1,024
(B) 8,196
(C) 2,048
(D) 8,000

Trial Version Wondershare PDFelement

3. (4 points) We are to process an 565x784 (m=565 pixels in the y or vertical dimension, n=784 pixels in the x or horizontal dimension) picture with the **`PictureKernel`** below:

```
__global__ void PictureKernel(float* Pin, float* Pout, int m, int n) {

  // Calculate the row # of the d_Pin and d_Pout element to process
  int Row = blockIdx.y*blockDim.y + threadIdx.y;

  // Calculate the column # of the d_Pin and d_Pout element to process
  int Col = blockIdx.x*blockDim.x + threadIdx.x;

  // each thread computes one element of d_Pout if in range
  if ((Row < m) && (Col < n)) {
    Pout[Row*n+Col] = 2*Pin[Row*n+Col];
  }
}
```

Assume that each block is organized as a 2D 16x32 array of threads (16 in the y dimension and 32 in the x dimension). Which of the following statements sets up the kernel configuration properly? Assume that int variable n has value 784 and int variable m has value 565. The kernel is launched with the statement PictureKernel<<<gridDim, blockDim>>>(d_Pin, d_Pout, m, n);

(A) dim3 gridDim(ceil(1, ceil(n/32), ceil(m/16); dim3 blockDim(1, 32 16);
(B) dim3 gridDim(ceil(n/32.0), ceil(m/16.0), 1); dim3 blockDim(32, 16, 1);
(C) dim3 gridDim(ceil(m/32.0), ceil(n/16.0), 1); dim3 blockDim(32, 16,1);
(D) dim3 gridDim(ceil(m/16), ceil(n/32), 1); dim3 blockDim(16, 32, 1);

4. (4 points) In Question 3, how many warps will have control divergence?
(A) 0
(B) 25*32 + 35
(C) 565
(D) 576

Trial Version Wondershare PDFelement

5. (4 points) Write some CUDA kernel code in which the threads in the same block transpose the elements of a matrix **mat** in shared memory in-place. That is, all threads should see that mat[i][j] is swapped with element mat[j][i] after the transposition. You can assume that that each block has dimensions dim3(16,16,1). Make your code as efficient in time and space as possible.

```
__shared__ float mat[16][16];
```

6. (4 points) Your kernel runs on a device where each Streaming Multiprocessor has 32K registers and 64KB of shared memory. The kernel code uses 30 registers (local variables) and 5KB of shared memory. The kernel is launched with 1,920,000 threads in total organized into a square grid with dimension 50 on each side. What is the maximum number of simultaneous blocks that will run on a single SM?

7. (4 points) What are the possible values of *dst after this kernel execution?

```
__global__ void race_me(char *dst) {
         dst[0] = threadIdx.x;
}

// host code
  cudaMalloc(&dst, 1);
  cudaMemset(dst, 3);  // assign value 3 to dst
  race_me<<<1,2>>>(dst);
```

Trial Version Wondershare PDFelement

(A) 0
(B) 1
(C) 0, 1
(D) 0, 1, 3
(E) 3

8. (4 points) Which of the following are design philosophies in GPU architecture?
(1) GPUs use massive parallelism to hide stalls
(2) GPUs have many low-latency execution units
(3) GPUs spread the cost of managing an instruction stream across many ALUs

(A) 1 and 2
(B) 1 and 3
(C) 2 and 3
(D) all of the above

9. (4 points) Your friend has profiled a kernel and discovered that the performance is limited by the latency of the divide operator. It turns out variable d in his kernel code (below) is 1 half the time, so your friend proposed the following optimization:

```
if (1 == d) r = d; else r = 1 / d;
```

Unfortunately, the performance did not improve. In one sentence, explain why.

(bonus 3 points) List the errors and typos that you reported via Piazza postings, e-mails, or in person communication with Prof. Hwu.

Trial Version Wondershare PDFelement

Question 2 (15 points, suggested time allocation 20 minutes): CUDA Basics. For the vector addition kernel and the corresponding kernel launch code, answer each of the sub-questions below.

```
__global__
void vecAddKernel(float* A, float* B, float* C, int n) {
  int i = threadIdx.x + 2*blockDim.x * blockIdx.x;
  if(i<n) C_d[i] = A_d[i] + B_d[i];
  i += blockDim.x;
  if(i<n) C_d[i] = A_d[i] + B_d[i];
}


int vectAdd(float* A, float* B, float* C, int n) {
// assume that size has been set to the actual length of
// arrays A, B, and C
  int size = n * sizeof(float);

  cudaMalloc((void **) &A_d, size);
  cudaMalloc((void **) &B_d, size);
  cudaMalloc((void **) &C_d, size);
  cudaMemcpy(A_d, A, size, cudaMemcpyHostToDevice);
  cudaMemcpy(B_d, B, size, cudaMemcpyHostToDevice);
  vecAddKernel<<<ceil(n/(2*1024.0)), 1024>>>(A_d, B_d, C_d, n);
  cudaMemcpy(C, C_d, size, cudaMemcpyDeviceToHost);
}
```

2(a). (2 point) Assume that the size of A, B, and C is 20,000 elements each. How many thread blocks will be generated?

2(b). (2 point) Assume that the size of A, B, and C is 20,000 elements each. How many warps are there in each block?

2(c) (2 point) Assume that the size of A, B, and C is 20,000 elements. How many threads will be created in the grid?

Trial Version Wondershare PDFelement

2(d) (4 points) Assume that the size of A, B, and C is 20,000 elements each. Is there any control divergence during the execution of the kernel? If so, identify the block index and warp index that causes the control divergence. Explain why or why not.

2(e). (5 points) Assume that the size of A, B, and C is 40,000 elements each. Is there any control divergence during the execution of the kernel? If so, identify the line number of the statement that causes the control divergence. Explain why or why not.

Trial Version Wondershare PDFelement

Question 3. (15 points, suggested time allocation 25 minutes): Your friend wants to find the sum of all elements of the **bigArr** array with GPU to boost his program performance. After he heard that you are taking ECE408, he comes to you for help, desperately. His idea is as follows: 1) load elements into shared memory of blocks; 2) use threads of each block to sum up elements in its shared memory; 3) output the sum by each block to form a new array **kernelOutput**; 4) use serial code to calculate the sum of all elements in **kernelOutput**. Assume that the kernel is launched with 1024 threads in each block.

```
#define BLOCK_SIZE = 1024
```

Global variable: `float bigArr[0..n-1]` (the length is n)
`float kernelOutput[0..ceil(n / (2* BLOCK_SIZE)) - 1]`

Kernel code:

```
01.   __shared__ float partialSum[2048];
02.   unsigned int bid = blockIdx.x;
03.   unsigned int t = threadIdx.x;
04.   unsigned int start = ______________X________________;
05.   if (start + t < n)
06.     partialSum[t] = bigArr[start + t];
07.   else
08.     partialSum[t] = 0.0f;
09.   if ( _______Y_______ < n)
10.     partialSum[_______Z_______] = bigArr[_______Y_______];
11.   else
12.     partialSum[_______Z_______] = 0.0f;
13.   for (unsigned int stride = 1;  stride <= BLOCK_SIZE;  stride *= 2)
14.   {
15.       __syncthreads();
16.       if (t < BLOCK_SIZE / stride)
17.         partialSum[t*stride*2] += partialSum[t*stride*2+stride];
18.   }
19.   if (t == 0)
20.     kernelOutput[bid] = partialSum[0];
```

3(a). (3 pts) Please help your friend fill codes in the blanks:

X:

Y:

Z:

Trial Version Wondershare PDFelement

3(b). (2 pts) Does control divergence happen in the 3rd iteration of the for-loop? Assume that the warp size is 32. (Hint: draw a picture of how the threads are mapped to the data for the next few sub-questions.)

3(c). (3 pts) How many warps in a block are active in the 3rd iteration of the for-loop? Assume that the warp size is 32.

3(d). (3 pts) Do you think the method your friend uses to read the global memory is coalesced? Why?

3(e). (4 pts) Can we put **__syncthreads()** in the if-statement as follows? Why?

```
if (t < BLOCK_SIZE / stride) {
    __syncthreads();
    partialSum[t*stride*2] += partialSum[t*stride*2+stride];
}
```

Trial Version Wondershare PDFelement

Question 4. (20 points, suggested allocation of time 35 minutes).
Your friend Jin suggested that doing the shared matrix-matrix multiplication (M × N) with matrix M transposed (M_trans) might increase performance in terms of memory coalesced (M_trans is the transposed matrix of M). The host code would provide M in its transposed form when calling the kernel. Jin being busy could not finish the transposed shared matrix-matrix multiplication and left few blanks to fill in. His idea **may** or **may not** be correct. Note that M_trans and N are square matrices.

```
#define TILE_WIDTH 32

__global__ void sgemm(float* M_trans, float* N, float* P, int Width) {

__shared__ float Mds[TILE_WIDTH][TILE_WIDTH];
__shared__ float Nds[TILE_WIDTH][TILE_WIDTH];

int bx = blockIdx.x;  int by = blockIdx.y;
int tx = threadIdx.x; int ty = threadIdx.y;

// Identify the row and column of the P element to work on
int Row = by * TILE_WIDTH + ty;
int Col = bx * TILE_WIDTH + tx;

float Pvalue = 0;
// Loop over the M and N tiles required to compute P element
for (int m = 0; m < (Width - 1)/TILE_WIDTH + 1; ++m) {

    // Collaborative load of M and N tiles into shared memory
    if(Row < Width && m * TILE_WIDTH + tx < Width) {
        Mds[ty][tx]  = M_trans[_____________________];
    } else {
        Mds[ty][tx] = 0;
    }
    if(Col < Width && m * TILE_WIDTH + ty < Width) ) {
        Nds[ty][tx]  = N [_____________________];
    } else {
        Nds[ty][tx] = 0;
    }
    __syncthreads();

    if(Row < Width && Col < Width) {
        for (int k = 0; k < TILE_WIDTH; ++k) {
            Pvalue += Mds[__][__] * Nds[__][__];
```

Trial Version Wondershare PDFelement

```
            }
        }
        __syncthreads();
    }
    if(Row < Width && Col < Width)
        P[Row*Width + Col] = Pvalue;
}
```

4(a) (6 points) Fill in the missing index calculations to make this code run correctly. If you think it is impossible to make this code to run correctly, state why. (*Hint: Draw a picture of matrix multiplication for the original layout M, study the behavior of the tiles, draw another picture of matrix multiplication based on transposed M_trans, adjust the use of threads in loading the tile elements to maximize global memory coalescing if necessary.*)

4(b) (3 points) For the **tiled matrix-matrix multiplication** (MxN) based on the original row-major layout, which input matrix will have coalesced access? (*Hint: Draw a picture of the tile loading access patterns.*)

(A) M
(B) N
(C) Both
(D) Neither

4(c) (3 points) For the **transposed (M is transposed) tiled matrix-matrix multiplication** (MxN) based on row-major layout, which input matrix will have coalesced access? (Hint: *Look at your picture and make sure that you adjust the tile loading index calculation to maximize coalescing if necessary.*)

(A) M
(B) N
(C) Both
(D) Neither

Trial Version Wondershare PDFelement

4(d) (3 points) For the **basic matrix-matrix multiplication** (MxN) (without tiling) based on row-major layout, which input matrix will have coalesced access? (*Hint: Draw a picture of the access patterns.*)

a) M
b) N
c) Both
d) Neither

4(e) (3 points) For the **transposed (M is transposed) basic matrix-matrix multiplication** (MxN) (without tiling) based on row-major layout, which input matrix will have coalesced access? (*Hint: Draw a picture of the access patterns.*)

a) M
b) N
c) Both
d) Neither

4(f) (2 points) Was Jin correct on his assumption? Suppose the transposed matrix is given. Justify your answer.

Trial Version Wondershare PDFelement

Question 5. (20 points, suggested time allocation 40 minutes) Convolution Fun. After teaching your roommate how to do parallel convolution, they come up with the following kernel. Their kernel is similar to the Strategy 2 presented in class where enough threads are instantiated to load all the input elements of a tile in one round. There are three major differences. Each thread block calculates two adjacent tiles in the x-dimension and the tiles and mask are no longer cubes. Look at the code below and answer the following questions.

```
#define TILE_X 8                        // Output tile width in the X-dimension
#define TILE_Y 4                        // Output tile width in the Y-dimension
#define TILE_Z 2                        // Output tile width in the Z-dimension
#define NUM_X 2
#define X_MASK_WIDTH 3
#define Y_MASK_WIDTH 3
#define Z_MASK_WIDTH 5
#define MASK_SIZE X_MASK_WIDTH * Y_MASK_WIDTH * Z*MASK_WIDTH


__constant__ float mask[Z_MASK_WIDTH][Y_MASK_WIDTH][X_MASK_WIDTH];

__global__ void conv3d(float *input, float *output, const int z_size, const int y_size, const int
x_size) {
  __shared__ float inputTile
[TILE_Z+Z_MASK_WIDTH-1][TILE_Y+Y_MASK_WIDTH-1][TILE_X+X_MASK_WIDTH-1];
  int tx = threadIdx.x;  int ty = threadIdx.y;  int tz = threadIdx.z;
  int bx = blockIdx.x;  int by = blockIdx.y;  int bz = blockIdx.z;

  for(int x_tile = 0; x_tile < NUM_X; x_tile++) {

    int x_o = NUM_X * bx * TILE_X + tx + x_tile * TILE_X;
    int y_o = by * TILE_Y + ty;
    int z_o = bz * TILE_Z + tz;

    int x_i = x_o - X_MASK_WIDTH/2;
    int y_i = y_o - Y_MASK_WIDTH/2;
    int z_i = z_o - Z_MASK_WIDTH/2;

    if (x_i >= 0 && y_i >= 0 && z_i >= 0 && x_i < x_size && y_i < y_size && z_i < z_size)
      inputTile[tz][ty][tx] = input[(z_i * y_size + y_i) * x_size + x_i];
    else
      inputTile[tz][ty][tx] = 0.0;

    float acc = 0.0;
```

Trial Version Wondershare PDFelement

```
      if(tz < TILE_Z && ty < TILE_Y && tx < TILE_X) {

        for(int z_mask = 0; z_mask < Z_MASK_WIDTH; z_mask++) {

          for(int y_mask = 0; y_mask < Y_MASK_WIDTH; y_mask++) {

            for(int x_mask = 0; x_mask < X_MASK_WIDTH; x_mask++) {

          acc += maskl[z_mask][y_mask][x_mask] * inputTile[tz+z_mask][ty+y_mask][tx+x_mask];

            }

          }

        }

        if(z_o < z_size && y_o < y_size && x_o < x_size)
          output[(z_o * y_size + y_o) * x_size + x_o] = acc;

      }

    }

  }
```

5(a) (4 points) Write out the host code for declaring the grid and block dimensions below.

5(b) (4 points) Unfortunately your roommate wasn’t paying attention when you talked about __syncthreads() and when you should use it. Please insert only the necessary __syncthreads() to the kernel above to make sure it works.

5(c) (3 points) For an internal output tile, what is the average number of times that each input element will be accessed from the shared memory during the calculation an output tile? (*Note:*

Trial Version Wondershare PDFelement

We are referring to tiles in this case, not thread blocks. Each thread block calculates two adjacent output tiles.)

5(d) (3 points) How does your answer to (C) compare to an output tile that is a cube that contains the same volume and why? Note: We are referring to tiles in this case, not thread blocks. Each thread block calculates two adjacent output tiles. (*Hint: Are all the input tile elements reused the same number of times? Does it matter where they are in the input tile ?*)

5(e) (3 points) Consider another output tile with dimensions 2x4x8. Is there a difference in terms of execution efficiency of the kernel between this 2x4x8 output tile and the output tile in the kernel above with dimensions 8x4x2? If so explain why, if not, explain why not.

Trial Version Wondershare PDFelement

5(f) (3 points) Is there any inefficiency in loading the input tiles for the two adjacent output tiles processed by each block? If so, identify the cause of the inefficiency and briefly suggest a strategy to eliminate the inefficiency. (*Hint: Draw a picture of the input tiles for these two output tiles.*)