ECE408/CS483 Practice Exam #1, Fall 2012

- You are allowed to use any notes, books, papers, and other reference material as you desire. No electronic assistance other than a calculator is permitted. 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, you are also allowed a 60-minute automatic extension.

- This exam is based on lectures as well as lab MPs/projects.

- The questions are randomly selected from the topics we covered this semester.

- You can write down the reasoning behind your answers for possible partial credit.

- **Good luck!**

Question 1 (50 points): Short Answer

Answer each of the following questions in as few words as you can. Your answer will be graded based on completeness, correctness, and conciseness.

(7 points) Give two reasons for adding extra "padding" elements to arrays allocated in GPU global memory?

2. (7 points) Give two potential disadvantages associated with increasing the amount of work done in each CUDA thread, such as loop unrolling techniques, using fewer threads in total?

3. (8 points) The following CUDA code is intended to perform a sum of all elements of the **`partialSum`** array. First, provide the missing operation, and second estimate the fraction of the iterations of the **`for`** loop that will have branch divergence.

```
__shared__ float partialSum[];
unsigned int tid = threadIdx.x;

for (unsigned int stride = blockDim.x;
     stride > 0;
     stride = stride >> 1)
{
     _____________________ ;; missing operation
     if (tid < stride)
         partialSum[tid] += partialSum[tid+stride];
}
```

4. (7 points) For the following code, estimate the average run time in cycles of a thread. Assume for simplicity that the **__sin()** function operates on radians, and that all operations including the **__sin()** take 1 cycle.

```
if (__sin( (float) threadIdx.x) > 0.5)
{
     :
     A operations
     :
}
else
{
     :
     B operations
     :
}
```

5. (7 points) Provide two advantages of a programmer-managed memory, such as the CUDA shared memory, over a hardware managed cache?

6. (7 points) Assuming capacity were not an issue for registers or shared memory, give one case where it would be valuable to use shared memory instead of registers to hold values fetched from global memory?

7. (7 points) Assume the following code will execute all in the same warp of a CUDA thread block. Write a some CUDA code in which the threads within the warp reverse an array **list** of length 32 in shared memory. Make your code as efficient in time and space as possible.

```
__shared__ float list[32]
```

Question 2 (20 points): 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_d, float* B_d, float* C_d, int n)
{
1. int i = threadIdx.x + blockDim.x * blockIdx.x;
2. 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
3.  int size = n * sizeof(float);
4.
5.  cudaMalloc((void **) &A_d, size);
6.  cudaMalloc((void **) &B_d, size);
7.  cudaMalloc((void **) &C_d, size);
8.  cudaMemcpy(A_d, A, size, cudaMemcpyHostToDevice);
9.  cudaMemcpy(B_d, B, size, cudaMemcpyHostToDevice);

10. vecAddKernel<<<ceil(n/256), 256>>>(A_d, B_d, C_d, n);

11. cudaMemcpy(C, C_d, size, cudaMemcpyDeviceToHost);

}
```

1(a). (1 point) Assume that the size of A, B, and C is 1000 elements. How many thread blocks will be generated?

1(b). (1 point) Assume that the size of A, B, and C is 1000 elements. How many warps are there in each block?

1(c) (1 point) Assume that the size of A, B, and C is 1000 elements. How many threads will be created in the grid?

1(c) (3 points) Assume that the size of A, B, and C is 1000 elements. 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.

1(b). (3 points) Assume that the size of A, B, and C is 768 elements. 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.

1(c). (5 points) As we discussed in class, data structure padding can be used to eliminate control divergence. Assuming that we will keep the host data structure size the same but pad the device data structure. Declare and initialize a new variable **padded_size** in line 3 and make some minor changes to statements in lines 4, 5, and 6 to eliminate control divergence during the execution of the kernel. Assume that random input values to floating point addition operations will not cause any errors or exceptions.

```
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
10.     int size = n * sizeof(float);
11.     int padded_size
12.     cudaMalloc((void **) &A_d,            );
13.     cudaMalloc((void **) &B_d,            );
14.     cudaMalloc((void **) &C_d,            );
15.     cudaMemcpy(A_d, A, size, cudaMemcpyHostToDevice);
16.     cudaMemcpy(B_d, B, size, cudaMemcpyHostToDevice);

 10. vecAddKernel<<<ceil(n/256.0), 256>>>(A_d, B_d,
C_d,              );

  11. cudaMemcpy(C, C_d, size, cudaMemcpyDeviceToHost);
```

1(d). (3 points) With this change to the host code, do you think that the "if (i<n)" is still needed in Line 2 of the original kernel? Why or why not?

1(e). (3 points) For large vector sizes, say greater than 1,000,000 elements, do you expect that the padded code will have significant impact on performance? Why or why not?

Question 3 (20 points): MP Skills. The following streaming convolution kernel is executed on an input image N, using the convolution filter Mc. P is the output of the kernel. The kernel launch configuration and code are show below. BLOCK_SIZE is known at compile time, but can be set anywhere from 16 to 256.

```
#define KERNEL_SIZE 5
#define TILE_SIZE (BLOCK_SIZE-KERNEL_SIZE+1)

dim3 block(BLOCK_SIZE, 1, 1);
dim3 grid((P.width+TILE_SIZE-1)/TILE_SIZE, 1, 1);

ConvolutionKernel<<<grid,block>>>(N,P);

__global__ void ConvolutionKernel(Matrix N, Matrix P)
{
      int colOut = blockIdx.x * TILE_SIZE + threadIdx.x;
      int colIn = colOut-2;

      __shared__ float Ns[KERNEL_SIZE][BLOCK_SIZE];

      Ns[0][threadIdx.x] = 0.0f;
      Ns[1][threadIdx.x] = 0.0f;

      for(int i=2; i<KERNEL_SIZE-1; i++)

            if(colIn >= 0 && colIn < N.width)
                  Ns[i][threadIdx.x]= N.elements[(i-
            KERNEL_SIZE/2)*N.width+colIn];

            else
                  Ns[i][threadIdx.x] = 0.0f;


      for(int i=0; i<P.height; i++)
      {
            if(colIn >= 0 && colIn < N.width && (i+KERNEL_SIZE/2) < P.height)
               Ns[KERNEL_SIZE-1][threadIdx.x]=
                  N.elements[(i+KERNEL_SIZE/2)*N.width + colIn];
            else
                  Ns[KERNEL_SIZE-1][threadIdx.x] = 0.0f;

            float pValue = 0.0f;

            if(threadIdx.x < TILE_SIZE && threadIdx.y < TILE_SIZE)
            {
                  for(int j=0; j<KERNEL_SIZE; j++)
                        for(int k=0; k<KERNEL_SIZE; k++)
                              pValue += Mc[j][k] *
                        Ns[threadIdx.y+j][threadIdx.x+k];

                  if(colOut < P.width)
                        P.elements[i * P.width + colOut] = pValue;
            }

            for(int j=0; j<KERNEL_SIZE-1; j++)
                  Ns[j][threadIdx.x] = Ns[j+1][threadIdx.x];
      }
}
```

(a) Out of the possible range of values for BLOCK_SIZE, for what values of BLOCK_SIZE will this kernel function correctly when executing on a current device?

(b) If the code does not execute correctly for all BLOCK_SIZE values, suggest a fix to the code to make it work for all BLOCK_SIZE values.

(c) Out of the possible range of values for BLOCK_SIZE, for what values of BLOCK_SIZE will the kernel completely avoid uncoalesced loads from global memory?

(d) Does the last line in the kernel cause any shared memory bank conflicts? Why?

Question 4 (10 points): Multi-GPU programming. You have been hired by an Italian F1 race team which is designing the new chassis for their new prototype. In the first stage of the design process, the engineering team would like to use small workstations with two GPUs (sharing the PCIe bus) to allow engineers to perform fast simulations of small parts of the prototype. Each simulation is an iterative process where each point in the output 3D volume is computed using two neighboring in each dimension from the input 3D volume points (multiply and add for each neighbor, 12 floating-point operations total for each output point). Assuming that each volume has 4096 x 1024 x 1024 points, the simulation code delivers 480 GFLOPS, and the PCIe bandwidth is 6GBps:

(a) Assume that the data layout is that all elements in the x-dimension are consecutive, then the y-dimensio, then the z-dimension.What is the optimal domain decomposition strategy i.e., which dimension should be divided across GPUs)? Why?

(b) How would you implement the inter-GPU communication code in the CPU in CUDA 3.0 and CUDA 4.0?

(c) If GPU Compute is available, would it be the previous approach optimal if MPI communication (2 GBps) is required?