Basic questions:

What is CUDA and what is the compiler that we are using?

CUDA is a parallel computing platform and programming model. We are using the nvcc compiler created by Nvidia.

Who is the host? Who is the device?

The host is the CPU and its memory, the device is the GPU and its memory.

What is the difference between normal C function and CUDA kernel function?

CUDA kernel functions are declared with __global__ at the beginning of the declaration. CUDA kernel functions are also called with <<<...>>> from the host.

What is __global__ and what does it indicate?

__global__ is a CUDA C keyword. It indicates a function that is called from the host, but is executed on the device.

What is SIMT? Were you able to observe it in HelloWorld.cu?

SIMT stands for Single Instruction Multiple Thread, and is an execution model for parallel programming. In this model all the threads on an SM are running the same instruction at a given cycle. Each thread has its own registers/local memory and therefor these instructions process different data. We can see this in HelloWorld.cu on the print statement, all the threads have their own data (threadId.x) which is the second value printed.

What happened when you uncommented the line "//if(threadIdx.x==0)" in HelloWorld.cu?

Only thread 0 in a block prints “Hello world! from the device! Thread:blockId.x,0“

What happened when you commented the line "cudaDeviceSynchronize();" in HelloWorld.cu? What does that tell you?

None of the print statements show up. This tells me that cudaDeviceSynchronize() tells the host to wait for the device to finish before continuing.

What is the syntax to invoke a CUDA kernel?

Foo<<< dim3 grid, dim3 block>>>(parameters,…)

grid = number of blocks in a grid (this can be 3d x, y, and z)

block = number of threads per block (this can be 3d x, y, and z)

OR

Foo<<< number_of_blocks, threads_per_block >>>(parameters,...)

Moderate questions:

What is a Streaming Multiprocessor (SM)? How many do we have in the 3 GPUs?

An SM is part of the GPU that runs CUDA kernels. Each SM contains many registers (that can be partitioned among threads), several caches, and shared memory for data interchange between threads within the SM.

Quadro K600 has 1 SM

GTX 980 has 16 SMs

Titan x has 24 SMs

What is a grid, a block and a thread?

A grid is a group of blocks (there can not be synchronization between the blocks). A block is a group of threads that can be synchronized. A thread is just an execution of kernel within a block of a given index.

Another way to think of it is that a grid is assigned to a GPU. A block is assigned to a Streaming Multiprocessor. Once a block is assigned to an SM it is divided into units called warps. A thread is within a warp.

Explain the memory model that CUDA exposes to the programmer.

Global memory, Shared memory within a block, local memory for each thread.

Look into matmultKernel00.cu in PA5.tar. What does the _shared_ keyword tell you?

That the memory is shared among threads within a block.

How is the CUDA memory model different from the standard memory model on CPU?

The CUDA memory model has global memory (on the CPU would be like main memory). Every block has shared memory which behaves similarly to shared memory on the CPU except it can not be shared between SMs. Every thread has its own registers (same as CPU), a thread also has its own local memory which last the lifetime of thread and stores data that cannot fit into the registers.

With respect to the previous 3 questions, what is the advantage that comes with CPU programming? (Hint: Shared memory in GPU is equivalent to ____ of CPU)

The advantage of CPU programming is that it does not require as much knowledge of the memory model. Shared memory is simply shared between all threads, where as on GPU, shared memory is only shared between threads on an SM. This can make problems more complex for example, when there are dependencies.

What are blockIdx.x, threadIdx.y, blockDim.z? What do they tell you?

blockIdx.x tells me the first dimension of block that a SM is working on. threadIdx.y tells me the second dimension of the thread. blockDim.z tells me size of the 3rd dimension of the blocks.

Assume a 1D grid and 1D block architecture: 4 blocks, each with 8 threads. How many threads do we have in total ? How do you calculate the global thread ID? Show the calculation for global threadID=26.

32 threads total

blockIdx.x * blockDim.x + threadIdx.x

3 * 8 + 2 = 26

Assume a 1D grid and 2D block architecture: 4 blocks, each with 2 threads in x direction and 4 threads in y direction. How many threads do we have in total ? How do you calculate the global thread ID? Show the calculation for global threadID=26.

32 threads total

(blockIdx.x * blockDim.x * blockDim.y) + (threadIdx.y * blockDim.x) + threadIdx.x

(3*2*4) + (1*2) + 0 = 26

Assume a 2D grid and 3D block architecture: 2 blocks in x direction and 2 blocks in y direction, each block with 2 threads in x direction and 2 threads in y direction and 2 threads in z direction. How many threads do we have in total ? How do you calculate the global thread ID? Show the calculation for global threadID=26.

32 threads total

block = (blockIdx.x + blockIdx.y * gridDim.x) * (blockDim.x * blockDim.y * blockDim.z)

thread = (threadIdx.z * (blockDim.x * blockDim.y)) + (threadIdx.y * blockDim.x) + threadIdx.x

global_id = block + thread

block = (1 + (1*2)) * (2*2*2) = 24

thread = (0 * (2*2)) + (1*2) + 0 = 2

global_id = 24 + 2 = 26

What were your observations on varying block and grid sizes in HelloWorld.cu? Show how you changed the print statements to reflect this.

Varying block and grid sizes changed the number of print statements that got executed, and there were duplicate messages.

For a 1D grid and 2D block I changed
```
dim3 blockSize(2,4,1);
dim3 gridSize(4,1,1);
```
To reflect this change in the print statement:
```
int global_id = threadIdx.x + blockDim.x * threadIdx.y + blockDim.x * blockDim.y * blockIdx.x;
printf("Hello world! from the device! Global_Thread_Id:%d\n",global_id);
```

For a 2D grid and 3D block I changed
```
dim3 blockSize(2,2,2);
dim3 gridSize(2,2,1);
```
To reflect this change in the print statement:
```
int block = (blockIdx.x + blockIdx.y * gridDim.x) * (blockDim.x * blockDim.y *    blockDim.z) ;
int thread = (threadIdx.z * (blockDim.x * blockDim.y) ) + (threadIdx.y *  blockDim.x) + threadIdx.x;
int global_id = block + thread;
printf("Hello world! from the device! Global_Thread_Id:%d\n",global_id);
```

Advanced questions:

Is the memory shared between CPU and GPU? Describe how data is transferred between CPU and GPU?

No the memory is not shared, however if the GPU runs out of ram it uses main memory like swap space. CPU allocates memory on the GPU then copies memory onto the GPU. To get memory from the GPU, the CPU copies the GPU's memory into the CPU's. The transfer uses pcie lanes to do the transfer.

Assume a 1D grid and 1D block architecture: 64 blocks, each with 64 threads. How many threads do we have in total? Do all the threads execute in parallel?

4096 threads total. No, the threads do not execute in parallel.

What is a Warp? What is the Warp size and the scheduler found in the 3 GPUs?

A warp is a vector of 32 threads. The warp size for all the GPUs is 32.

From above 2 questions, how many threads can run in parallel at a given time step for the 3 GPUs?

Quadro K600 = 2048

GTX 980 = 32768

Titan X = 49152

What is the theoretical peak performance that the 3 GPUs can achieve in terms of GFLOPsS? How is this derived?

Cuda cores * 2 * clock speed

Quadro K600 = 192 * 2 * 0.88GHz = 337.92 GFLOPS

GTX 980 = 2048 * 2 * 1.22GHz = 4997.12 GFLOPS

Titan X = 3072 * 2 * 1.08GHz = 6635.52 GFLOPS

How is the given vecaddKernel(in PA5) different from HelloWorld Kernel?

The vecaddKernel has parameters.