Discussion

Write short summary of what you learned (limit to 5 sentences), and answer the following questions. As with previous labs, you may work and submit a report individually or collectively with your discussion group.

I learned that the execution time, GFlopS, and GBytesS can all be improved through memory coalescing. Memory coalescing is the grouping of memory accesses for all threads in a warp to a single region of memory. Finally, I learned how grids, blocks, and threads are organized within one another. Finally, I understood how we can use this organization to coalesce memory accesses. Threads within blocks utilize shared memory which is very fast.

Basic questions:

1. What is CUDA and what is the compiler that we are using?
    1. CUDA is a framework and API for parallel programming using NVIDIA graphics cards. We are using the NVIDIA CUDA compiler, nvcc.
2. Who is the host? Who is the device?
    1. The host is the CPU and the device is the graphics processor.
3. What is the difference between normal C function and CUDA kernel function?
    1. A C function executes on the host while a CUDA kernel function executes on the device.
4. What is __global__ and what does it indicate?
    1. The __global__ keyword is for defining a function which will run on the device.
5. What is SIMT ? Were you able to observe it in the vector add program?
    1. SIMT is single thread multiple threads. Yes, we observe this when we coalesce the data access. We see that each thread executes the single instruction at the same time, which grabs the corresponding memory, and the warp's threads are all able to execute faster since the threads all have their data within one instruction.
6. What is the syntax to invoke a CUDA kernel?
    1. To invoke a CUDA kernel, we use chevrons (<<< and >>>) to denote a device function and pass the grid number and block number before passing regular function arguments using (and).

Moderate questions:

1. What is a Streaming Multiprocessor (SM)? How many do we have in the GPUs used in this lab?

    - A streaming multiprocessor is the GPU component which runs our CUDA kernels. Our department machines have the Nvidia GTX 1060 with 10SMs.

2. What is a grid, a block and a thread?

- A grid is a group of blocks, a block is a group of threads, and a thread is a single processing unit

3. Explain the memory model that CUDA exposes to the programmer.

   - CUDA exposes a model where the host and device share separate memory spaces. CUDA provides an interface for transferring and managing data between these memory spaces.

4. What does the __shared__ keyword tell you? This is used in matmultKernel00.cu in PA4.tar.

   - The __shared__ keyword is used to declare a variable which shares memory between all threads within a block. It is up to 100x faster than uncached global memory.

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

   - The CUDA memory model introduces more restrictions, finer grain of control, and more data types. For example, the CPU memory model allows for data to be changed (unless it's constant), however, the CUDA memory model prevents the host from modify the device data and vice versa. Most importantly, caching is done manually by the programmer for CUDA device memory.

6. 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)

   - Shared memory in GPU is equivalent to cache of CPU. The advantage of CPU programming is caching is taken care of by hardware.

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

   - Blocks, threads, and grids are multidimensional, hence why we have member variables x, y, and z. Specifically, blockDim.z gives the number of threads in a block for the z direction. Next, blockIdx.x gets the index of the current block in the x direction, Likewise, threadIdx.y gets the current thread's index in the y direction.

8. 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.

   - 4 blocks each with 8 threads = 32 total threads.

- Global thread ID = threadIdx.x * blockIdx.x * M where M is the number of threads per block.
- 26 = 2 + 3 * 8
    1. M = 8. 26/8 = 3 with remainder 2, so block 3 thread 2.

9. 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. 2 blockDim.x * 4 blockDim.y. = 8 threads/block, 8 threads/block * 4 blocks = 32 threads.
    - Global thread ID = blockIdx.x * blockSize + threadIdx.y * blockDim.x + threadIdx.x
        1. blockSize = blockDim.x * blockDim.y
        2. blockIdx.x = ID / blockSize
            1. 26 / 8 = 3 remainder 2
        3. threadIdx.y = remainder // blockDim.x
            1. 2 / 2 = 1 remainder 0
        4. threadIdx.x = remainder
            1. threadIdx.x = 0
        5. 3 * 8 + 1 * 2 + 0 = <u>26</u>

10. 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 Total threads
        1. (gridDim.x * gridDim.y) * (blockDim.x * blockDim.y * blockDim.z) = (2 * 2) * (2 * 2 * 2) = 32
    - Global thread ID = blockIdx.y * (gridDim.x * blockSize) + blockIdx.x * blockSize + threadIdx.z * (blockDim.x * blockDim.y) + threadIdx.y * blockDim.x + threadIdx.x
        1. blockSize = blockDim.x * blockDim.y * blockDim.z
            1. 2 * 2 * 2 = 8
        2. rowSize = gridDim.x * blockSize
            1. 2 * 8 = 16
        3. blockIdx.y = ID / rowSize
            1. 26 / 16 = 1 (remainder 10)
        4. blockIdx.x = remainder / blockSize
            1. 10 / 8 = 1 (remainder 2)
        5. threadIdx.z = remainder / blockDim.x * blockDim.y
            1. 2 / (2 * 2) = 0 remainder 2
        6. threadIdx.y = remainder / blockDim.x
            1. 2 / 2 = 1 remainder 0

7. threadIdx.x = remainder
    1. 0
8. blockIdx = {x: 1, y: 1}, threadIdx = {x: 0, y: 1, z: 0}
    1. ID = 1 * (2 * 8) + 1 * 8 + 0 + 1 * 2 + 0 = 26

Advanced questions: (You can refer PA4 tarball and/or Internet)

1. Is the memory shared between CPU and GPU? Describe how data is transferred between CPU and GPU.
    1. Data can be shared between CPU and GPU, however the memory has to be allocated in both memories, and it has to be explicitly copied between them. With CUDA 6, Unified Memory allows programmers to access data using a single pointer.
2. 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?
    1. We have 64^2 threads total.
    2. Yes, we have more than the 2 SMs required to execute all 4096 threads.
3. What is a Warp? What is the Warp size and the scheduler found in the GPUs used in this lab?
    1. A warp is a group of threads executed in Single Instruction Multiple Thread fashion.
    2. Warp size is 32.
    3. For the GP106 architecture, warps are scheduled dynamically. Each warp scheduler (one per processing block) is capable of executing two warp instructions per clock cycle.
4. From above 2 questions, how many threads can run in parallel at a given time step for the GPUs used in this lab?
    1. From this paper, we see that max threads per multiprocessor is 2048. Our GPUs have 10 SMs, so we can execute 20,480 threads at any given time.
5. What is the theoretical peak performance that the these GPUs can achieve in terms of GFLOPsS? How is this derived?
    1. 34.18 GFLOPs
    2. This is derived by SMs * frequency (GHz) * 2
        1. 10 * 1.709 * 2 = 34.18