LSU EE 7722 **Homework 3** Solution **Due: 28 March 2016**

Problem 1: Read about the microarchitecture of NVIDIA GPUs from the following resources. Focus on the material needed to understand warp scheduling and instruction issue, and in particular to answer the next problem in this assignment.

NVIDIA has published whitepapers describing the microarchitecture of their GPUs. They describe a variety of features, including those for graphics. When reading them focus on the operation of the MPs (or SM's or SMX's). The whitepapers below describe three generations of GPUs, Fermi (2.X), Kepler (3.X), and Maxwell (5.X). NVIDIA uses a two-letter, three-digit name to describe the microarchiteture. The first letter is always g, the second letter indicates the generation (f, k, m), and the number indicates something like a version.

For Fermi read `http://www.ece.lsu.edu/gp/refs/gf100-whitepaper.pdf`, for Kepler read `http://www.ece.lsu.edu/gp/refs/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf`, and for Maxwell read. `http://www.ece.lsu.edu/gp/refs/GeForce-GTX-980-Whitepaper-FINAL.pdf`.

Additional information can be found in Section 4 of the CUDA C Programming Guide: `https://docs.nvidia.com/cuda/cuda-c-programming-guide`.

Problem 2: In the matrix/vector multiply code used in class we noticed that when we assigned an entire matrix/vector multiply to a thread the FMADD instructions could access matrix elements directly. But when a thread only operated on one output vector component the compiler emitted `LDC` instructions.

Based on the descriptions above, would such `LDC` instructions be a barrier to saturating FP capability (assuming sufficiently large input)?

Answer your question with diagrams showing how instructions are issued and indicate where in the references the illustrated behavior is supported.

Short Answer: In a GK110 (CC 3.5) GPU a scheduler can dispatch two instructions per cycle. See the Quad Warp Scheduler section at the bottom of page 9 of the Kepler GK110 white paper. `LDC` instructions would probably not be a barrier to FP saturation for double-precision instructions because with 16 functional units per scheduler it would take two cycles per warp to dispatch, and so one dispatch unit could be dispatching a DP instruction while the other could be dispatching an `LDC` instruction. The `LDC` instructions won't prevent DP saturation unless they take more than two cycles to dispatch. Things are more complex for single precision, see the detailed answer.

Detailed Answer: First consider a region of code consisting only of single-precision FFMA instructions executing on an MP in a CC 3.5 (GK110) device. Since there are 192 CUDA cores the peak rate is $192/32 = 6$ "warps" per cycle. Since there are only four warp schedulers and each can schedule from one warp per cycle the throughput can be achieved by having each scheduler dispatch 1.5 instructions per warp. Or, more accurately, dispatch two instructions, one of these for all threads in the warp, and the other for half the instructions in a warp. This is illustrated in the diagram below. The code fragment consists of independent `FFMA` instructions. Call the four schedulers *scheduler 0* to *scheduler 3*. Warp 0 is handled by scheduler 0, warp 1 is handled by scheduler 1, warp w is handled by scheduler $w \bmod 4$.

Two designs are shown below. In design 1 each scheduler has $192/4 = 48$ CUDA cores, which are functional units that can perform FFMA operations. The diagram shows that in cycle 0 all 32 threads of warp 0 dispatch I0, but only 16 threads it seems have I1 dispatched, since there are only 48 CUDA cores. Therefore I1 takes two cycles to dispatch. In cycle 1 I1 is finished up and I2 dispatches completely. The diagram shows that warp 1 executes identically, to save space warps 2 and 3 aren't shown but they would execute identically also. Warp 4 would likely be handled by scheduler 0, perhaps starting when the next instruction in warp 0 has a dependence with an instruction that has not yet finished.

In Design 2, each scheduler has 32 *private* CUDA cores and there are 32 CUDA cores shared by schedulers 0 and 1, and another 32 shared by schedulers 2 and 3. In cycle 1 scheduler 0 uses its 32 CUDA cores and the 32 shared cores, and so two instructions in warp 0 can completely dispatch while for warp 1 only one instruction dispatches. In cycle 1, scheduler 1 gets the shared cores, so warp 0 only executes on instruction and warp 1 executes two.

```
I0:  FFMA r1, r2, r3, r4
I1:  FFMA r5, r6, r7, r8
I2:  FFMA r9, r10, r11, r12
...

# Design 1 - 48 CUDA cores per scheduler.
       t/Cycle ->
warp   0    1    2    3    4 ...
0      [I0] [I2] [I3] [I5]
0      [I1     ] [I4     ]
1      [I0] [I2] [I3] [I5]
1      [I1     ] [I4     ]

# Design 2 - 32 private CUDA cores per scheduler + 32 shared CC per 2 sched.
       t/Cycle ->
warp   0    1    2    3    4 ...
0      [I0] [I2] [I3] [I5]
0      [I1]      [I4]
1           [I1]      [I4]
1      [I0] [I2] [I3] [I5]
```

The executions shown above saturate the CUDA cores. Now lets suppose that we have a mix of LDC and FFMA instructions. Assuming design 1, we could not saturate DP because we need both dispatch units to be busy in order to use all 48 CUDA cores. However, in Design 2 notice that either warp 0 or warp 1 has an idle dispatch unit each cycle. That dispatch unit could be handling a non-FP instruction, such as LDC. That's shown below, where `i3` and `i7` are LDC instructions and the upper-case I instructions are FP.

Note that the description in this problem does not contradict anything in the white paper, but that doesn't mean it's possible. There might be another factor that prevents saturation. Also note that for single precision the ratio must be three FP instructions per LDC instruction, which would not work with our vector/matrix multiplication code.

```
# Design 2 - 32 private CUDA cores per scheduler + 32 shared CC per 2 sched.
       t/Cycle ->
warp   0    1    2    3    4 ...
0      [I0] [I2] [I4] [I6]
0      [I1] [i3] [I5] [i7]
1           [I1] [i3] [I5]
1      [I0] [I2] [I4] [I6]
```