LSU EE 7700-2 **Homework 5** **Due: 13 March 2013**

For details on how the GPU works see the CUDA C Programming Guide, linked to `http://www.ece.lsu.edu/gp/gp/ref.html`. For hardware details in particular see Chapter 1 (brief history and overview), Chapter 4 (organization overview), and Sections 5.2.3 (warp/instruction scheduling), 5.3.2 (memory), 5.4 (instruction scheduling and functional units), and Appendix F (some more details on resources and instruction handling).

Problem 1: The Volkov paper describes algorithms for dense linear algebra on GPUs, it is linked to `http://www.ece.lsu.edu/gp/gp/ref.html` under the heading "Matrix Multiplication." The paper is important as much for the algorithms themselves, as for the methodical approach used in characterizing the GPUs and developing the algorithms. The paper was written for the NVIDIA CC 1.x generation of GPUs, which are now obsolete. Nevertheless, the paper is valuable in showing how to measure the capabilities of a device and tune code for it.

Read the following sections of the Volkov paper: Abstract, Introduction, Section 2, 2.1, 2.3, and Section 3.

(*a*) Add a column to Table 1 for the following devices: A GTX 580 and a Kepler K20.

As mentioned in class, NVIDIA GPUs have high instruction latency, but are designed so that the latency can be hidden by using lots of threads.

(*b*) Find the part of the paper in which Volkov describes the experiments used to find instruction latency. What latencies does he report?

(*c*) This section also gives the number of warps (warps aren't called warps elsewhere in the paper) needed to hide this latency. Show how that number is computed.

(*d*) Find the place in the NVIDIA documentation which describes the number of warps needed to hide latency. Show the number of warps for each generation of CUDA architecture (1.x, 2.0, 2.1, 3.x) needed to hide latency assuming code consisted of just add and multiply instructions (which is sort of the default case). Also show the warps needed under the assumption that the code consists of reciprocal instructions. (Note that this part does not require the Volkov paper.)

(*e*) The impact of instruction latency is not as bad if the instruction reading a register does not immediately follow the instruction writing the register. Find the part of the paper that verifies that this is indeed true. What is the minimum number of warps necessary to avoid stalls on code with suitably distant dependencies, according to the paper?

Problem 2: We know that the `synchthreads()` call should be avoided because it adds overhead. Shown below is an excerpt of the machine code that performs a tree reduction of values within a warp, each group of three instructions adds on a value at a different distance. Notice that each instruction is dependent on the instruction before it.

```
@!P4 LDS R3, [R2+0x40];
@!P4 FADD R0, R0, R3;
@!P4 STS [R2], R0;

@!P0 LDS R3, [R2+0x20];
@!P0 FADD R0, R0, R3;
@!P0 STS [R2], R0;

@!P1 LDS R3, [R2+0x10];
@!P1 FADD R0, R0, R3;
@!P1 STS [R2], R0;

@!P2 LDS R3, [R2+0x8];
@!P2 FADD R0, R0, R3;
@!P2 STS [R2], R0;

@!P3 LDS R3, [R2+0x4];
@!P3 FADD R0, R0, R3;
@!P3 STS [R2], R0;
```

(*a*) Compute the execution time of this code, measured in cycles, on a CC 2.0 device, for the launch of a single block with 1024 threads. Assume that all instruction latencies are 24 cycles. Measure time from the execution of the first instruction (shown as something like $\boxed{\text{I0}}$ in class) to the execution of the last instruction.

(*b*) The code above did not have `synchthreads` between each group. In this part consider code in which there is:

```
@!P4 LDS R3, [R2+0x40];
@!P4 FADD R0, R0, R3;
@!P4 STS [R2], R0;
BAR.RED.POPC RZ, RZ;
@!P0 LDS R3, [R2+0x20];
@!P0 FADD R0, R0, R3;
@!P0 STS [R2], R0;
BAR.RED.POPC RZ, RZ;
@!P1 LDS R3, [R2+0x10];
@!P1 FADD R0, R0, R3;
@!P1 STS [R2], R0;
BAR.RED.POPC RZ, RZ;
@!P2 LDS R3, [R2+0x8];
@!P2 FADD R0, R0, R3;
@!P2 STS [R2], R0;
BAR.RED.POPC RZ, RZ;
@!P3 LDS R3, [R2+0x4];
@!P3 FADD R0, R0, R3;
@!P3 STS [R2], R0;
BAR.RED.POPC RZ, RZ;
```

Assume that the barrier instruction, `BAR`, uses cuda cores for execution (as does the `FADD`), and that its latency is 24 cycles. An instruction following a barrier cannot execute until 24 cycles after *the last thread* executes its barrier instruction. (In contrast, the `FADD` for a thread cannot execute until at least 24 cycles after the `LDS` in the same thread executes.)

Compute the time for the code above for this assumed behavior.