LSU EE 7700-2 Homework 1 Due: 12 March 2012

For the following assignment read Chapter 2 in the CUDA C Programming Guide, which is linked to the NVIDIA documentation page,
`http://developer.nvidia.com/nvidia-gpu-computing-documentation`, and which can also be found in /usr/local/cuda/doc in a standard Linux CUDA toolkit installation. Also refer to the documentation for `cuobjdump`, which describes the NVIDIA machine language and to the class notes on the GF3 at `http://www.ece.lsu.edu/gp/notes/set-study-gf3.pdf`.

Source code and accounts for this assignment may be made available.

Problem 1: A CUDA program, a variation on the "dots" demo shown in class, operates on a 1000000-element array using a kernel which operates on a single element. (See the code fragments below.)

(*a*) Suppose the block size is set to 64. How many blocks should be launched? Modify the code below so that the kernel is launched with this block size and number of blocks. Note that only an assignment for the `x` component of the grid configuration is shown; add the others. *Note: This is an easy problem, and something close to the solution can be found in other examples. Please solve it on your own.*

```
__host__ void dots_launch() {
 dim3 dg, db;
 // Answer below.
 dg.x =



  dots<<<dg,db>>>();
}

__global__ void dots() {
  int idx = threadIdx.x + blockIdx.x * blockDim.x;
  if ( idx >= array_size ) return;
  b[idx] = v0 + v1 * a[idx].x + v2 * a[idx].y;
}
```

(*b*) Suppose that the code above is run on a Tesla C2050. How many blocks will there be per multiprocessor? *Note: To solve this problem yo need to look characteristics of the C2050 in the Programming Guide.*

(*c*) Suppose the kernel (`dots`) consists of 21 instructions. How many cycles would it take for the kernel to finish on a Tesla C2050 assuming that:

- No instruction has to wait, even for loads.
- All instructions use CUDA cores. (None of them use the special functional units).

Problem 2: The `dots` kernel and the NVIDIA CC 2.0 machine code for the `dots` kernel appears below.

```
__global__ void dots() {
  int idx = threadIdx.x + blockIdx.x * blockDim.x;
  if ( idx >= array_size ) return;
  b[idx] = v0 + v1 * a[idx].x + v2 * a[idx].y; }
```

```
code for sm_20
        Function : _Z4dotsv
/*0000*/ MOV R1, c [0x1] [0x100];
/*0008*/ S2R R2, SR_Tid_X;
/*0010*/ S2R R0, SR_CTAid_X;
/*0018*/ IMAD R0, R0, c [0x0] [0x8], R2;
/*0020*/ ISETP.LT.AND P0, pt, R0, c [0x2] [0xc], pt;
/*0028*/ @!P0 BRA.U 0xa0;
/*0030*/ @P0 MOV R3, c [0x2] [0x10];
/*0038*/ @P0 IMUL.HI R2, R0, 0x8;
/*0040*/ @P0 MOV R6, c [0x2] [0x18];
/*0048*/ @P0 IMAD R8.CC, R0, 0x8, R3;
/*0050*/ @P0 MOV R5, c [0x2] [0x4];
/*0058*/ @P0 IMUL.HI R4, R0, 0x4;
/*0060*/ @P0 IADD.X R9, R2, c [0x2] [0x14];
/*0068*/ @P0 IMAD R6.CC, R0, 0x4, R6;
/*0070*/ @P0 LD.E R2, [R8];
/*0078*/ @P0 LD.E R3, [R8+0x4];
/*0080*/ @P0 IADD.X R7, R4, c [0x2] [0x1c];
/*0088*/ @P0 FFMA R2, R5, R2, c [0x2] [0x0];
/*0090*/ @P0 FFMA R0, R3, c [0x2] [0x8], R2;
/*0098*/ @P0 ST.E [R6], R0;
/*00a0*/ EXIT;
```

(*a*) Briefly describe what each instruction does referring to the `dots` code. For example, for the instruction at `0018` one might say "Compute `idx` = `blockIdx.x` * `blockDim.x` + `threadIdx.x`".

(*b*) Show the equivalent GF3 machine code based on the description given in the class notes. Note that the GF3 has input values automatically delivered to registers and results automatically moved to the next stage. Be sure to take' advantage of GF3 vector operations (though they aren't a perfect match).

(*c*) Comment on the difference in code size and possible differences in execution time on systems with the same number of floating-point units.