Name _____

EE 7700-2

Final Examination

Monday, 6 May 2013 to Sunday, 12 May April 2013

Work on this exam alone. Regular class resources, such as notes, papers, documentation, and code, can be used to find solutions. Do not discuss this exam with classmates or anyone else, except questions or concerns about problems should be directed to Dr. Koppelman. Answers to questions about the exam may be sent to all students in the class (without divulging who asked the questions). For example, if something in the paper isn't clear, an explanation will be sent to everyone.

Several questions in this exam are based on the paper, "GPUs and the Future of Parallel Computing," by Keckler, Dally, Khailany, Garland, and Glasco, IEEE Micro Magazine, September 2011. The paper is linked to the `http://www.ece.lsu.edu/gp/gpu-descriptions.html` course Web page (near the bottom). If accessed off-campus provide the userid `ee4720` and the password given in class (a warning for those considering climbing a fence near the PMAC).

Problem 1	_____	(15 pts)
Problem 2	_____	(15 pts)
Problem 3	_____	(15 pts)
Problem 4	_____	(15 pts)
Problem 5	_____	(15 pts)
Problem 6	_____	(25 pts)
Exam Total	_____	(100 pts)

Alias _____

Good Luck!

Problem 1: [15 pts] Figure 3 in the paper and the illustration on page 35 of the presentation show that a lane in Echelon has a 32 kiB LM (local SRAM) (consisting of four banks of 8 kiB each). The LM is used for registers, scratchpad (shared) memory, and for a global memory cache (called the L0 cache). The amount allocated to each of these three uses is decided at the equivalent of kernel launch time.

(*a*) Compare the number of registers, amount of shared memory, and amount of L0 or L1 cache space *per thread* available on a CC 2.0 multiprocessor (SM) and an Echelon TOC. For both devices choose an appropriate number of threads to base your answer on. Make intelligent choices for how space in the Echelon LM is divided.

☐ Amount of storage per thread for each device:

☐ How number of threads counted:

(*b*) Comment on which device is better based on the numbers above.

Problem 2: [15 pts] Each TOC is a 3-way VLIW (*Very Long Instruction Word*, called LIW in the paper) processor, meaning a processor that fetches instructions in 3-instruction *bundles*. It is the responsibility of the compiler or assembly language programmer to organize instructions into bundles.

Consider a loop that contains 30 instructions, I_0 to I_{29}, with I_0 following I_{29} in all but the last iteration. The compiler has scheduled the instructions so that dependencies do not slow execution for instructions I_0 to I_{26}, so for this problem assume that there are no dependencies at all between these instructions. But I_{27} depends on I_{26} and cannot execute until 100 cycles after I_{26}. Instructions I_{28} and I_{29} can execute at the same time as I_{27}. Once $I_{27} - I_{29}$ complete the cycle repeats with the execution of I_0.

(*a*) How many threads are needed per Echelon TOC to hide the latency of this loop? Answer this question using a method similar to the one used in Homework 6.

☐ Number of threads per in an Echelon TOC to hide the latency:?

(*b*) Repeat the analysis for a CC 2.0 device. Use the same latency assumptions: the only latency to consider is between I_{26} and I_{27}.

☐ What is the minimum number of threads needed to hide latency in a CC 2.0 multiprocessor?

(*c*) Consider a scientific programmer trying to choose between the two devices based on how many threads his code will have to have (the fewer the better). The programmer would like to choose the system which uses fewer threads for some level of performance. Assume that both systems run at the same clock frequency and use the loop above to analyze the systems.

☐ For programming effort compare the devices based on number of threads per ...

Problem 3: [15 pts] Among other things Figure 2 shows the bandwidth between various parts of Echelon, including the interface to (off-chip) DRAM (which has a bandwidth of 1600 GB/s). Answer the following questions about data bandwidth. *Hint: Once these questions are understood the answers are easy.*

(*a*) What is the maximum rate at which 8-byte load instructions in an Echelon can access data based on the information in Table 3 (on page 13) and assuming that the memory system can keep up? (The question is primarily asking the maximum number of loads per cycle.)

- Assume that each load reads 8 bytes.
- Perform the analysis for a program that has a maximum number of loads possible given the description in the paper.

☐ The maximum data rate due to loads, in GB/s, is:

(*b*) The maximum load instruction bandwidth computed in the last part should exceed the off-chip bandwidth of 1600 GB/s. The only way the loads could execute at their maximum bandwidth would be if they found data in on-chip storage.

Figure 2 (b) shows a bandwidth of 256 GB/s between a TOC and external devices. Based on the data in Figure 2 and elsewhere in the paper, what is the minimum hit ratio in the level-0 cache (the cache using lane memory, see Figure 3), needed so that the maximum load instruction bandwidth can be achieved. (For this part you might assume a 100% hit ratio outside of the TOC.)

☐ Minimum L0 hit ratio to achieve maximum load bandwidth:

(*c*) For this part suppose that every TOC is fully using its 256 GB/s off-TOC bandwidth. What is the minimum hit ratio in the SRAM banks needed so that 64 GB/s connection to the tile crossbar (top of Figure 2 (b)) is not overloaded?

☐ Minimum SRAM bank hit ratio to support 256 GB/s off-TOC bandwidth:

Problem 4: [15 pts] As described on page 14, Echelon uses multi-level scheduling of threads to reduce energy consumption.

(*a*) An active thread is evicted to the on-deck set if its current instruction (the one about to execute) is a consumer of a result from a long-latency instruction. It is up to the compiler to identify such consumer instructions and to do so in a way which helps with performance and energy goals. Assume that evicting a thread costs energy but that not evicting a thread at a consuming instruction could reduce performance (by leading to a situation in which an on-deck thread is ready to execute, while none of the active threads are ready because their current instructions are consumers of results from long-latency instructions).

Appearing below are Fermi instructions based on the Homework 6 vertex transform code, modified for 64-bit registers. Assume that the input data for this is similar to that used in Homework 6, meaning that a large number of vertices is being transformed. Show which instruction(s) should be marked as consumers.

```
LOOP:
      IMAD R8, R0, R10, c [0x2] [0x30];
      IMAD R6, R0, R10, c [0x2] [0x40];
      IADD R0, R0, c [0x2] [0x0];
      LD.E.64 R2, [R8];
      LD.E.64 R3, [R8+0x8];
      ISETP.LT.AND P0, pt, R0, c [0x2] [0x14], pt;
      DFMA R4, R2, c [0x2] [0x8], RZ;
      DFMA R2, R2, c [0x2] [0x18], RZ;
      DFMA R4, R3, c [0x2] [0x10], R4;
      DFMA R5, R3, c [0x2] [0x20], R2;
      ST.E.64 [R6], R4;
      ST.E.64 [R6+0x8], R5;
      @P0 BRA LOOP;
```

☐ Show consumer instruction(s).

☐ Describe assumptions made about the behavior of the load instructions.

☐ Explain how this helps meet the goals mentioned above.

(*b*) When an active thread is evicted, an on-deck thread must be chosen, *promoted*, to take its place. Devise two methods, complex and simple, to choose which thread to promote. The complex method might require a moderate amount of hardware to make its choice, while the simpler method should use less hardware. Show an example in which the complex method outperforms the simple one.

☐ Complex method to select thread to promote.

☐ Simple method to select thread to promote.

☐ Code example that runs better using complex method.

☐ Explanation, including assumptions about data.

Problem 5: [15 pts] Recall that an Echelon lane is 3-way VLIW, the first two slots of the bundle can be arithmetic instructions and the last slot can only contain memory and branch instructions. A CC 2.0 Fermi multiprocessor has 32 CUDA cores (which execute most arithmetic instructions) and 16 load/store units (for memory instructions).

Consider the following two statements:

> Echelon is wasteful because in regions of code requiring only arithmetic instructions $\frac{1}{3}$ of the instructions must b nops, and regions requiring only memory instructions $\frac{2}{3}$ must be nops.
>
> CC 2.0 devices are wasteful because it is impossible to initiate instructions on the 32 CUDA cores and the 16 load/store units simultaneously.

Which statement is more correct? Justify your answer, include assumptions about hardware costs and program characteristics.

Problem 6: [25 pts] The code samples below read an array of $n \times m$ elements and write an n-element array `out_array`. Element `out_array[i]`$=\sum_{j=i\times m}^{(i+1)\times m-1} a_j$, where a_j is element `array[j]`, the input array.

- `grp_size` (m) is known at compile time.
- `array_size` is $m \times n$.
- The block size is a multiple of `grp_size`.
- Arrays `array` and `out_array` are in global memory.

(*a*) Describe why the code below will run inefficiently, especially on Kepler and pre-Fermi GPUs. *Hint: It's breaking a fundamental rule about memory access.*

```
const int tid = threadIdx.x + blockIdx.x * blockDim.x;
const int num_threads = blockDim.x * gridDim.x;
const int idx_start = threadIdx.x * grp_size;

for ( int i=idx_start; i<array_size; i += num_threads * grp_size )
  {
    const int grp_idx = i / grp_size;
    float sum = 0;
    for ( int j=0; j<grp_size; j++ )
      sum += array[i+j];
    out_array[grp_idx] = sum;
  }
```

☐ Inefficient because:

(*b*) Describe two reasons why the code below will not run correctly.

```
for ( int i=tid; i<array_size; i += num_threads )
  {
    const int grp_idx = i / grp_size;
    const int val = array[i];

    if ( i % grp_size == 0 )
      out_array[grp_idx] = val;
    else
      out_array[grp_idx] += val;
  }
```

☐ Reason 1 why code will not execute correctly.

☐ Reason 2 (perhaps reason 1b) why code will not execute correctly.

Problem 6, continued:

(*c*) The code below uses shared memory to avoid problems from the first code sample. It too suffers efficiency problems but should run correctly.

```
__shared__ float s_array[1024];

for ( int i=tid; i<array_size; i += num_threads )
  {
    const int grp_idx = i / grp_size;
    const int val = array[i];

    // All threads load shared memory from global memory.
    //
    s_array[threadIdx.x] = val;

    __syncthreads();

    if ( threadIdx.x < blockDim.x / grp_size )
      {
        const int idx_thread_0 = i - threadIdx.x;
        const int grp_idx_thread_0 = idx_thread_0 / grp_size;
        float sum = 0;
        for ( int j=0; j<grp_size; j++ )
          sum += s_array[ threadIdx.x * grp_size + j ];
        out_array[grp_idx_thread_0 + threadIdx.x] = sum;
      }

    __syncthreads();
  }
```

☐ Identify and fix the problem related to shared memory.