CpSc 418 **Final Exam** April 20 2016

100 points

Time for the exam: 2 hours, 30 minutes.

Open book: anything printed on paper may be brought to the exam and used during the exam. This includes the textbook, other books, printed copies of the lecture slides, lecture notes, homework and solutions, and any other material that a student chooses to bring.

Calculators are allowed: no restriction on programmability or graphing. There are a few simple calculations needed in the exam, a calculator will be handy, but the fancy features will not make a difference.

No communication devices: That's right. You may not use your cell phone for voice, text, web-surfing, or any other purpose. Likewise, the use of computers, iPods, etc. is not permitted during the exam.

Write your name and student number on your exam book for two points of extra credit. If you don't have them written at the end of the exam and need to write them after we collect (or try to collect) your exam, we will mark your book as "late" and you will not get the extra credit.

Do all five questions.

CpSc 418 **Final Exam** April 20, 2016

Graded out of **100 points**

Write your name and student number on your exam book for two points of extra credit. You **must** do this **before** the end of the exam to get the extra credit. Then, do all **five** questions below.

1. The zero-one principle (**20 points + 5 points Extra Credit**)
   Let $X[0..N-1]$ be a *bitonic* array of N elements, where N is even, and every element of X is either 0 or 1.

   (a) (**5 points**) Explain what it means that an array is bitonic. Use at most three sentences.

   (b) (**10 points**) Let Y be an array with N elements such that:

   $$\begin{array}{rcll} Y[i] & = & \min(X[i], X[i+(N/2)]), & \text{if } 0 \le i < N/2; \\ & = & \max(X[i-(N/2)], X[i]), & \text{if } N/2 \le i < N. \end{array}$$

   Show that either

   - $Y[i] = 0$ for $0 \le i < N/2$, or
   - $Y[i] = 1$ for $N/2 \le i < N$.

   Both may hold for particular choices of X, but you just need to show that at least one of these conditions holds.

   Hint: Consider the following cases:

   - case X is all 0s.
   - case X has at least one 1. In this case, let i_{lo} be smallest integer in $[0, \ldots, N-1]$ such that $X[i_{\text{lo}}] = 1$, and let i_{hi} be largest integer in $[0, \ldots, N-1]$ such that $X[i_{\text{hi}}] = 1$. By definition (you may use these facts without proving them):,

     $$\begin{array}{l} 0 \le i_{\text{lo}} \le i_{\text{hi}} < N \\ X[i] = 0, \quad \text{if } 0 \le i < i_{\text{lo}} \text{ or } i_{\text{hi}} < i < N \\ X[i] = 1, \quad \text{if } i_{\text{lo}} \le i \le i_{\text{hi}} \end{array}$$

     We have three cases:
     - case $N/2 \le i_{\text{lo}}$.
     - case $i_{\text{hi}} < N/2$, you should be able to say "this is analagous to the previous case.
     - case $i_{\text{lo}} < N/2 \le i_{\text{hi}}$, this is the interesting case.

     That's it. Three cases. If you need another hint, draw a picture for what X looks like for each case (e.g. with $N = 8$).

   (c) (**5 points**) Let Y be defined as in question 1b. Show that $Y[0..(N/2)-1]$ is bitonic and that $Y[(N/2)..(N-1)]$ is bitonic.

   Hint: After you prove this for $Y[0..(N/2)-1]$, you should be able to say "The proof for $Y[(N/2)..(N-1)]$ is analogous."

   (d) (**5 points, Extra Credit**) Let A be a $N \times M$ array, where N is even, and $A[i, j]$ denotes the element of A in row i and column j. Let $A[i, 0..M-1]$ denote a row of A. Assume that every element of A is either 0 or 1, and that for $0 \le i < N$, row $A[i, 0..M-1]$ is sorted into ascending order of i is even and into descending order if i is odd.

   Let B be the $N \times M$ array where for $0 \le j < M$, column $B[0..N-1, j]$ is $\mathsf{sort}(A[0..N-1, j])$, where sort sorts the elements of the column into ascending order.
   Prove that at least $N/2$ rows of B are either all 0s are all 1s. For example, if $N = 16$, and B has 6 rows that are all 0s and three rows that are all 1s, then the claim is satisfied.

   Hint: consider *pairs* of rows of A and the results from previous parts of this problem.

2. CUDA (**20 points**) Let's say that I have an array of

$$n = 2^{27} = 2^{17} \cdot 2^{10} = 131,072 \cdot 1,024 = 134,217,728$$

`floats`, and I want to compute their sum using my blazing-fast GPU.

(a) (**10 points**) I've written three approaches to compute the sum below. Each is either incorrect or inefficient. Identify the problem with each, giving a short (at most three sentences) explanation for each one describing why it is wrong or why it is slow.

i. Create one block with 134,217,728 threads executing the kernel:

```
0  __global__ void sum1(float *x, int n) {
1      // assume n is a power of 2
2      int myId = blockDim.x * blockIdx.x + threadIdx.x;
3      if(myId < n) {
4          for(int stride = 2; stride < n; stride *= 2) {
5              __syncthreads();
6              if((myId % stride) == 0)
7                  x[myId] = x[myId] + x[myId + stride]
8          }
9      }
10     // the result is in x[0]
11 }
```

The kernel launch is:

```
sum1<<<1, 134217728>>>(x, 134217728)
```

ii. The same kernel as for version [i], but this time the kernel launch is:

```
sum1<<<131072, 1024>>>(x, 134217728)
```

iii. Create one block with 1024 threads executing the kernel where each thread computes the sum of `n/1024` elements before the threads perform a reduce:

```
0      __global__ void sum3(float *x, int n, int m) {
1          // assume n and m are powers of 2
2          int myId = blockDim.x * blockIdx.x + threadIdx.x;
3          if(myId < n) {
4              // compute the sum for my part of the array
5              float sum = 0.0;
6              for(int i = 0; i < m; i++)
7                  sum += x[myId + i];

8              // reduce
9              for(int stride = 2; stride < blockDim.x; stride *= 2) {
10                 __syncthreads();
11                 if((myId % stride) == 0)
12                     x[myId] = x[myId] + x[myId + stride]
13             }
14         }
15         // the result is in x[0]
16     }
```

The kernel launch is:

```
sum3<<<1, 1024>>>(x, 134217728, 131072)
```

(b) See the next page.

(b) (**10 points**) There are many ways to improve the performance of the slow kernel. Pick one such as coalescing global memory references, or using shared memory, or avoiding bank conflicts, or reducing thread divergence, or ....

- (**2 points**) State what optimization you are performing.
- (**4 points**) Write a short (no more than three sentence) description of what the optimization does.
- (**2 points**) Write the revised kernel. You can just write the lines that you change compared with the kernels I wrote above, and write
  lines 05-08 of sum2
  or similar to refer to lines from the kernels above.

3. GPUs (**20 points**)
Earlier this month, nVidia™ announced the GP100 GPU. It has 3584 single-precision floating point cores with a peak 10.6 teraflops of single-precision floating point computation (1 teraflop = 1000 gigaflops = 10^{12} floating point operations per second). The interface to off-chip, global memory has a bandwidth of 720 GBytes/sec.

   (a) (**10 points**) How many floating point operations must the GP100 perform on each `float` loaded from global memory if it is to achieve its peak floating point operation rate (and not by limited by memory bandwidth)?

   (b) (**10 points**) Give two reasons that a GPU needs thousands of active threads to fully utilize the processors. For each, you can give its name (e.g. the nVidia™ jargon word or phrase) and a short description (at most four or five sentences for each of your two reasons).

4. **Speed-up** Let's say that I spend all day running four programs in sequence on my x86 linux machine:

   - Program-A runs for 1 minute on the x86.
   - Program-B runs for 1 minute on the x86.
   - Program-C runs for 1 minute on the x86.
   - Program-D runs for 1 minute on the x86.

   Each program depends on the results of the one that came before it; so, I can't run them in parallel ☹. After Program-D finishes, I go back and start the sequence with Program-A again.

   (a) (**5 points**) Assuming that I work 8 hours per day and get paid $1 each time I complete all four tasks, how much do I earn per day?

   (b) (**5 points**) An nVidia™ sales person drops by my office and tells me that with a new, GP100 GPU, I can get the following speedups:

   - Program-A: speed-up = 100.0;
   - Program-B: speed-up = 10.0;
   - Program-C: speed-up = 1.0;
   - Program-D: speed-up = 0.1.

   If I sell my linux machine and move all four of my computing tasks to the GPU, long will it take me to complete all four tasks? What is the overall speed-up?

   (c) (**5 points**) Maybe selling my linux box wasn't such a good idea. Let's say I run Program-D on my linux machine, and I run the other three programs on the GPU. For simplicity, we'll ignore the time for communication when switching between running one task on the GPU and the next on the CPU (or the other way around). Because the tasks have sequential dependencies, I can't use the CPU and the GPU at the same time. With this combined CPU + GPU approach what is the speed-up compared with running everything on the CPU?

   (d) (**5 points**) What is Amdahl's law? Write the mathematical formula, and write two or three sentences to explain the intuition it gives for the speed-up results in the previous two sections.

5. **Other stuff (20 points)**
   Answer each question below with 1-3 sentences. Points may be taken off for long answers.

   (a) (**4 points**) What is a pipeline bypass?

   (b) (**4 points**) What is a fused multiply-add?

   (c) (**4 points**) What is the cross-section bandwidth of a message-passing multiprocessor?

   (d) (**4 points**) What is a "straggler" in a map-reduce computation?

   (e) (**4 points**) How do "back-up tasks" mitigate the problems caused by stragglers?