CpSc 418 **Homework 4** Due: Mar. 22, 2017, 11:59pm
Early Bird: Mar. 20, 2017, 11:59pm

**140 points.**

Please submit your solution using the handin program. Submit your solution as
cs418 hw4
Your submission should consist of two files:

- `hw4.cu`: CUDA source code for the coding parts your solution.
- `hw4.pdf`: PDF for the written response parts of your solution

Some C functions for questions 3 and 4 are available at
http://www.ugrad.cs.ubc.ca/~cs418/2016-2/hw/4/src/hw4.cu.
If your code does not compile, we might give you zero points for the problem. Points will be deducted for compiler warnings.

1. **Sorting Networks: clean and dirty sequences (35 points)** This problem explores a substructure that is commonly used in sorting networks. Let $\mathcal{S}$ be a sorting network with $N$ inputs and outputs. Let the inputs be `x[0]`, `x[1]`, ..., `x[`$N-1$`]`, and let the outputs be `y[0]`, `y[1]`, ..., `y[`$N-1$`]`. The sorting network performs a compare-and-swap between `x[`$i$`]` and `x[`$(N-1)-i$`]` for $0 \le i < (N/2)$. In other words,

$$\begin{aligned} \texttt{y[0]} &= \min(\texttt{x[0]}, \texttt{x}[N-1]) \\ \texttt{y}[N-1] &= \max(\texttt{x[0]}, \texttt{x}[N-1]) \\ \texttt{y[1]} &= \min(\texttt{x[1]}, \texttt{x}[N-2]) \\ &\dots \end{aligned}$$

   (a) **(5 points)** Draw this sorting network for the case $N = 10$.

   Of course, we will use the 0-1 principle to reason about this network. We say that a sequence of 0s and 1s is "clean" if it consists entirely of 0s or entirely of 1s. We say that a sequence of 0s and 1s is "dirty" if it has at least one 1 and at least one 0. Note that any clean sequence is trivially monotonic or bitonic.

   For the remainder of this problem, assume that `x` consists only of 0s and 1s; assume that inputs `x[`$0, \dots, (N/2)-1$`]` are sorted into ascending order; and assume that inputs `x[`$N/2, \dots, N-1$`]` are sorted into ascending order. It is not necessarily the case that `x` is fully sorted; in particular, `x[`$(N/2)-1$`]` may be greater than `x[`$N/2$`]`.

   (b) **(10 points)** Prove that either the sequence `y[`$0, \dots, (N/2)-1$`]` is clean or that the sequence `y[`$(N/2), \dots, (N-1)$`]` is clean (or possibly both).

   (c) **(10 points)** Prove that `y[`$0, \dots, (N/2)-1$`]` and `y[`$N/2, \dots, N-1$`]` are both bitonic sequences. **Hint:** as shown in part (b), one of these sequences is clean, and therefore bitonic. You just need to show that the other sequence is bitonic.

   (d) **(10 points)** Prove that for $0 \le i < (N/2)$ and $(N/2) \le j < N$, `y[`$i$`]` $\le$ `y[`$j$`]`. In English, this says that every element of the lower half of `y` is less-than-or-equal-to any element of the top half.

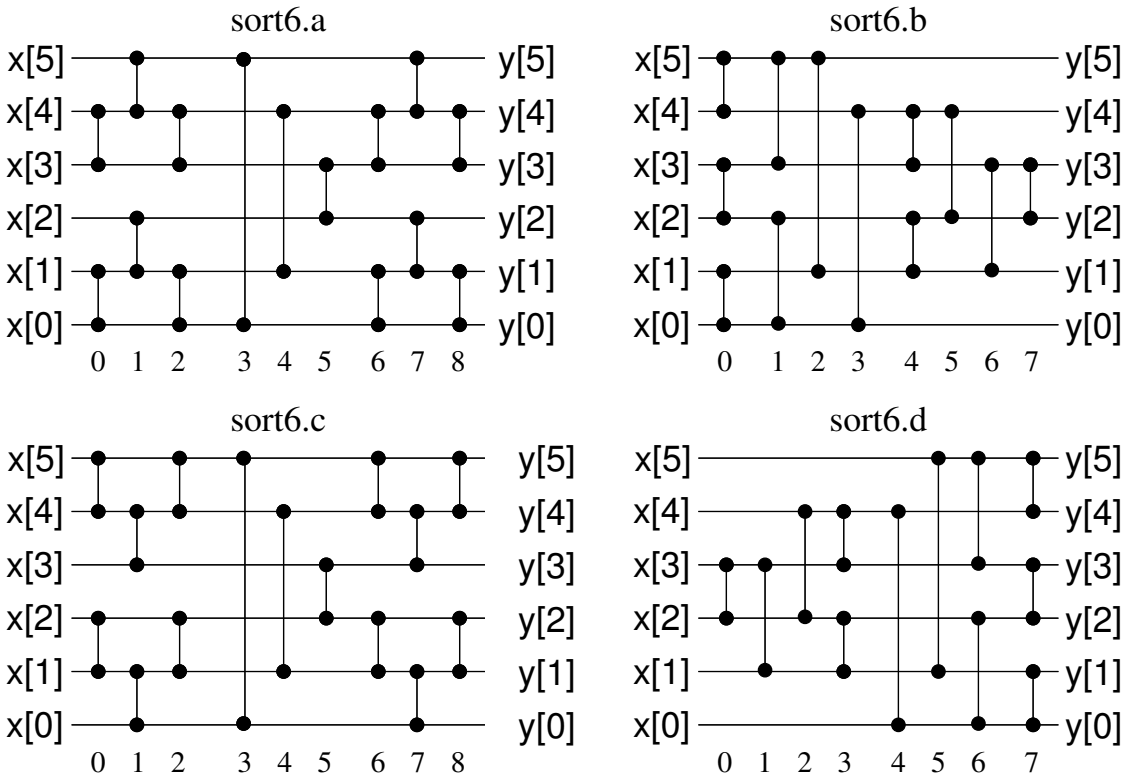


Figure 1: Sorting Networks

2. **Sorting Networks: examples (50 points)** Figure 1 shows four sorting networks with six inputs: sort6-a, sort6-b, sort6-c, and sort6-d. At least one of these networks sorts correctly, and at least one does not.

   (a) **(5 points)** Identify a subnetwork of one of these sorting networks that sorts two elements. To specify a subnetwork, you can make a copy of the figure, circle the compare-and-swap elements of the subnetwork, and label the circuit, or you can do it textually as described below.
   Here's an example of a textual description of a subnetwork:

   ```
   from sort6.b % the parent network of your subnetwork
      column 1:  (0,2), (3,5)
      column 2:  (1,5)
   ```

   This refers to three compare-and-swap elements from sort6.b: the compare-and-swap in column 1 spanning rows 0 and 2; the compare-and-swap in column 1 spanning rows 3 and 5; and the compare-and-swap in column 2 spanning rows 1 and 6.
   **Hint:** Yes, this is a very simple problem, nearly "five points just for reading".

   (b) **(5 points)** Identify a subnetwork of one of these sorting networks that sorts three elements.

   (c) **(5 points)** Identify a subnetwork of one of these sorting networks that sorts four elements.

   (d) **(5 points)** Identify a subnetwork of one of these sorting networks that has $M$ inputs for $M \leq 6$, and combines them as described in Question 1.

(e) **(10 points)** Identify one of the four networks that **does not** sort correctly. Label the inputs with values of **0** or **1** that the network does not sort correctly. Label the output of each compare-and-swap with its value for this input, and label the outputs of the sorting network with their values.

You can use a screenshot of the figure in your solution and annotate it with your counter-example. I have also provided the original `.pdf` at
http://www.ugrad.cs.ubc.ca/~cs418/2016-2/hw/4/src/sort6.pdf.

(f) **(5 points)** Identify one of the networks that **does** sort correctly. Explain why if at least one of `x[`$0, \ldots, (N-1)$`]` is a 1, then `y[5]` is a 1.

(g) **(10 points)** For the same network that you chose for the previous question, use the 0-1 principle to prove that it sorts correctly. Of course, you may use your answers to the previous parts of this question and the results from question 1 in your proof.

(h) **(5 points):** State whether or not the remaining two networks sort correctly – that means the networks that you didn't choose for Questions 2e or 2g.

3. **GPU GFlops (35 points)**
Let

$$f(x) \quad = \quad \tfrac{5}{2}(x^3 - x)$$

Given $x_0$, we'll define a sequence $x_1$, $x_2$, ... where

$$x_{i+1} \quad = \quad f(x_i), \quad i \geq 0$$

If $|x_0| > \sqrt{7/5} \approx 1.1832$, then the sequence of $x_i$ diverges as $i \to \infty$; so, we'll restrict our attention to the case where $|x_0| \leq 1$. That's because 1 is a nice number, and it gives us a little "safety margin". Let's say that we start with a vector of $n$ values for $x_0$ and want to compute $x_m$ for each of these values. In other words we compute $m$ steps of the recurrence for each initial value of $x$.

(a) (20 points): Write a CUDA implementation of this recurrence Your code should provide a function,

```
void f(float *x, float *xm, uint n, uint m);
```

where `x` is an array of `n` initial values; `xm` is an array of `n` floats into which to write the final result; and `m` is the number of iterations to compute. When `f(x, xm, n, m)` returns,

`xm[i]` $\quad = \quad f^m($`x[i]`$)$

should hold for $0 \leq$ `i` $< n$.

Define the *throughput* of your implementation as `n`$*$`m`$/t$ where $t$ is the time to execute `f(x, xm, n, m)`. Your goal is to make a *fast* implementation on one of the `linXX` machines – i.e. `lin01.ugrad.cs.ubc.ca`, `lin02.ugrad.cs.ubc.ca`, ..., `lin01.ugrad.cs.ubc.ca`. You can pick the values of `n` and `m` that maximize the throughput for an execution that takes at most 0.1 seconds to execute. In your solution, specify the values of `n` and `m` that you used and which machine you ran your code on for timing measurements.

(b) (5 points): How many Gflops (billions of floating point operations per second) does your kernel achieve? Provide data from executing your program that shows how the number of blocks, number of threads per block, and any other critical design decisions affect the performance of your kernel.

(c) (5 points): I've provided a C implementation of `f` called `f_cpu` in hw4.cu. Measure the run-time for `f_cpu` using the same parameters as you used for Question 3b. What speed-up do you get by using the GPU and CUDA?

(d) (5 points): Given the observations from Question 3b, describe how you took these performance trade-offs into account when writing your implementation of the function `f`. What observations can you make about writing efficient CUDA programs?

Your answer doesn't need to be long. Anywhere from five to twenty sentences should be fine. Grading will be based on how your explanations make the design of your code clear, not on length.

4. **saxpy (20 points)**
This is similar to Question 3, but we'll use `saxpy` instead of the recurrence described in the previous problem.

(a) (5 points): Compile and run the `saxpy(uint n, float a, float *x, float *y)` example from hw4.cu where `x` and `y` are arrays of `n` initial values; and `a` is a floating point number. When `saxpy(x_in, x_out, n, a, y)` returns,

```
y'[i]  =  a*x[i] + y[i]
```

where `y'[i]` is the value of from the array `y` *after* the call to `saxpy`, and `y[i]` is the value before the call.

Define the *throughput* of your implementation as $n/t$ where $t$ is the time to execute `saxpy(x_in, x_out, n, a, y)`. Your goal is to make a *fast* implementation. You can pick a value for `n` that maximizes the throughput for an execution that takes at most 0.1 seconds to execute. As in Question 3, run your code on one of the `linXX` machines, state what machine you ran your code on, and state the value for `n` that you used to maximize throughput.

(b) (10 points): write a C implementation of `saxpy` called `saxpy_cpu`. Measure the run-time for `saxpy_cpu` using the same parameters as you used for Question 4a. What speed-up do you get by using the GPU and CUDA?

(c) (5 points): What are the limiting factors for the performance of your CUDA implementation of `saxpy`. Provide timing measurements to support your conclusion. You might try making modifications to `saxpy` (including changing what it computes) to test your ideas for what the performance bottlenecks are.

# Why?

**Question 1**: The concept of creating "clean" sequences occurs frequently in sorting networks, including at least one sorting network from Question 2. This question is to introduce you to the concept and make it easier to recognize the pattern.

I'll add that Questions 1 and 2 were motivated in part by questions from piazza and office hours. In the process of writing these questions, I found some great slides on sorting networks – see
https://hoytech.github.io/sorting-networks/

**Question 2**: This explores sorting networks with some specific examples. The goal is to help you recognize common structures in sorting networks and how these structures are connected with the counting arguments used to show that a sorting network sorts correctly.

Typically, I would ask a more concrete question like this one before the more general question like Question 1. In this case, the properties of combining sorted sequences as described in Question 1 are helpful in Question 2; so, I asked Question 1 first. Of course, you can solve the questions in either order. Please submit your final solution in the same order as the questions were asked to make grading simpler. Furthermore, you can use the properties that you are supposed to prove in Question 1 in this question whether or not you complete Question 1.

**Question 3**: This question lets you write a fairly simple CUDA kernel and see the performance trade-offs. The problem is deliberately open-ended. You should be able to get pretty good performance with a moderate amount of effort. You can also use this as a way to explore the trade-offs and quirks of GPU programming and find ways to squeeze more performance out of the GPU.

I will post a throughput that is acceptable to get 100%. I will note what my solution gets. Solutions that are faster than mine will receive extra credit.

**Question 4**: This is an even simpler GPU programming problem than Question 3 – you can get all or most of the code from the assigned reading. That's valuable as it makes sure that everyone gets across the "hello world" threshold of getting something to work in CUDA. Once your first program works, the others can be easier. The "interesting" part about the problem is comparing it with the results you got in Question 3. That's why I asked the other question first and asked for a comparison here.