CpSc 418 **Practice Final Exam** April 2017

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.

Maybe we'll ask you do do all five questions. Or, we might ask you to do four out of five. Read the instructions. **No lenience will be shown to those who can't read.**

```
__device__ tr1() {
   uint i = threadIdx.x;
   for(uint j = 0; j < 32; j++) {
      float t1, t2;
      if(j < i) {
         t1 = shBuf[i][j];
         t2 = shBuf[j][i];
         shBuf[j][i] = t1;
         shBuf[i][j] = t2;
      }
   }
}
```

```
__device__ tr2() {
   uint i = threadIdx.x;
   for(uint j = 1; j <= 16; j++) {
      float t1, t2;
      uint jj = (i+j) % 32;
      t1 = shBuf[i][jj];
      t2 = shBuf[jj][i];
      shBuf[jj][i] = t1;
      shBuf[i][jj] = t2;
   }
}
```

Figure 1: Two implementations of transpose

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. **Transposing a matrix**
   The problem of transposing a matrix occurs frequently in parallel programming. We saw it in matrix multiplication. It also occurs in algorithms that have the pattern:

   Each of `P` processes works on a subproblem of size `N/P`.
   Each process "scatters" it's result over the other processes:
   I.e. it sends a *different* message of N/P^2 elements to each of the other processes.
   Each process works on the subproblem consisting of the data it just received.

   For simplicity, we'll assume that the GPU has a warp-size of 32, and that n is a multiple of this warpsize. Let's assume that a 32×32 block has already been loaded into

   ```
   __shared__ float shBuf[32][32];
   ```

   Consider the implementations of transpose shown in Figure 1.

   (a) Figure 2 shows two grids representing `shBuf`. On the left figure, indicate which locations are updated when `j=0`, `j=1`, and `j=30`. For example, you can draw a line through the cells that are updated and label the line "`j=0`". Likewise, on the right figure, indicate which locations are updated when `j=1`, `j=2`, and `j=15`.

   (b) Which implementations is more efficient? Justify your answer. Possible considerations include global memory accesses, shared memory bank conflicts, CGMA, and thread-divergence. When you state a reason, give a short explanation (one to three sentences) of why the two functions are different with respect to this issue.

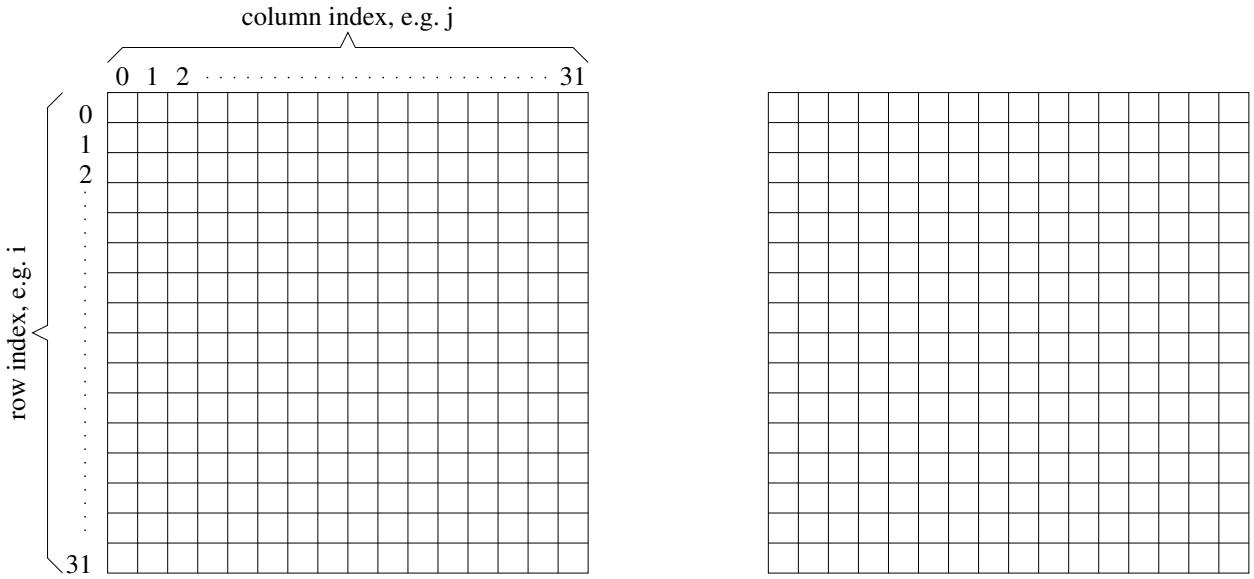

Figure 2: Fill in this figure for question 1a

2. **0-1 Principle** Certainly a candidate for a question.
3. **PReach and model-checking** We had two lectures on this. That's about 5% of the term; so, we could ask one 5-point question such as:
   (a) What is model checking?
   (b) How does PReach distribute work between processes?
   (c) What is a safety property?
   (d) What is mutual exclusion?
4. **CUDA concepts:** We've spent most of the second half of the term working with CUDA. We could ask some short answer questions such as:
   (a) What is data parallelism? Maybe present a piece of sequential code and ask you to describe whether or not it is data parallel, and why.
   (b) What is SIMD? What is SIMT? What's the difference?
   (c) Why do CUDA program have way more threads than there are processors?
   (d) What is CGMA?
   (e) Describe and/or compare global memory, shared memory, and constant memory?
   (f) What does `__syncthreads()` do? Maybe more questions about communication and synchronization between threads in CUDA.
   (g) What corresponds to our beloved λ (for performance analysis) when programming in CUDA?
5. **CUDA code:** Give you a simple piece of sequential code and the skeleton for a corresponding CUDA implementation. Ask you to fill in the template and answer some questions about performance trade-offs.
6. **A bit from the first half of the term,** for example we might ask something about
   (a) A simple example of reduce.

(b) An even simpler example of reduce.

(c) ...