CpSc 418 **Final Exam** December 4, 2018

Graded out of **100 points**.

Five questions.

0. **Who are you?** (2 points)

   (a) What is your name? Mark Greenstreet

   (b) What is your student number? 00000000

1. **Matrix-Multiplication** (20 points)
   Consider matrix multiplication, $Z = X\,Y$ on a message passing machine with P processors. Let X, Y, and Z be $N \times N$ matrices. Furthermore, we assume that P is a perfect square and that N is a multiple of P. Each matrix has N^2 elements with N^2/P elements stored on each processor. In particular, we will have each processor store a $\frac{N}{\sqrt{P}} \times \frac{N}{\sqrt{P}}$ submatrix (i.e. "tile") of each of the X, Y, and Z matrices. We will write $\widehat{X}_{I,J}$ to indicate the submatrix of X that is stored on processor $p(I,J)$

$$\widehat{X}_{I,J}(i,j) \quad = \quad X((N/\sqrt{P}) * I + i, (N/\sqrt{P}) * J + j), \quad 0 \leq I, J < \sqrt{P},\ 0 \leq i, j < N/\sqrt{P}$$

   and likewise for Y and Z.

   (a) **(2 points)** How many multiply-add operations are required to compute the product of two $N \times N$ matrices?
   Your answer should be a formula using the variable N.
   N^3

   (b) **(2 points)** How many multiply-add operations are required to compute the product of two $\frac{N}{\sqrt{P}} \times \frac{N}{\sqrt{P}}$ matrices?
   Your answer should be a formula using the variables N and/or P.
   $\left(\frac{N}{\sqrt{P}}\right)^3$

   In a parallel implementation:

   - Each processor, $p(I,J)$, sends its submatrix, $\widehat{X}_{I,J}$, to each processor in its "row", i.e. to processors $p(I,0)$, $p(I,1), \ldots, p(I, \sqrt{P}-1)$. Likewise, $p(I,J)$, sends its submatrix, $\widehat{Y}_{I,J}$, to each processor in its "column": i.e. to processors $p(0,J), p(1,J), \ldots, p(\sqrt{P}-1, J)$.
   - After these matrices have been sent (and received), each processor, $p(I,J)$, has matrices $\widehat{X}_{I,K}$ and $\widehat{Y}_{K,J}$ for $0 \leq K < \sqrt{P}$.
   - Each processor, $p(I,J)$, computes

$$\widehat{Z}_{I,J} \quad = \quad \sum_{K=0}^{\sqrt{P}-1} \widehat{X}_{I,K}\,\widehat{Y}_{K,J}$$

     Assume that the matrix-adds can be done as part of the fused multiply-adds for the matrix multiplications (they can).

   (c) **(4 points)** How many fused multiply-adds does each processor (e.g. $p(I,J)$) perform to compute its part of the product (e.g. $\widehat{Z}_{I,J}$)?
   Your answer should be a formula using the variables N and/or P.

$$\sqrt{P}\left(\frac{N}{\sqrt{P}}\right)^3 \quad = \quad \frac{N^3}{P}$$

(d) **(4 points)** If we ignore communication time, then the time to compute XY is just the time for the fused multiply-adds. Ignoring communication time, what is the speed-up when computing the product $Z = XY$ using P processors?
Your answer should be a formula using the variables N and/or P.

$$\mathit{SpeedUp} \;=\; \frac{T_{seq}}{T_{par}} \;=\; \frac{N^3}{N^3/P} \;=\; P$$

(e) **(4 points)** Now, consider communication time. Assume that it takes time $\lambda + Mt_0$ to send (and receive) a message of M matrix-elements from one process to another process. With the algorithm described above, each processor sends (and receives) $\sqrt{P} - 1$ messages to (and from) the other processors in its row, and another $\sqrt{P} - 1$ messages to (and from) the other processors in its column. Each of these messages conveys one of the $\widehat{X}$ or $\widehat{Y}$ matrices. What is the time spent to send (and receive) messages? Your answer should be a formula using the variables N, P, λ, and/or t_0.

Each process holds a $\frac{N}{\sqrt{P}} \times \frac{N}{\sqrt{P}}$ block of each matrix. The process sends its block of X to the $\sqrt{P} - 1$ other processes in its row and receives blocks from them. Each exchange takes time $\lambda + \frac{N^2}{P}t_0$. Likewise, each process sends its block of Y to the other processes in its column and receives a clock for each of them. This takes additional time of $\lambda + \frac{N^2}{P}t_0$. The total time for communication is

$$T_{comm} \;=\; 2(\sqrt{P} - 1)\left(\lambda + \frac{N^2}{P}t_0\right)$$

Note: in practice, it is "impossible" to specify communication cost in enough detail that there aren't multiple, plausible interpretations. Alternatively, I could write a full page spelling out the details, but that is too much reading to process on an exam. Some solutions may charge for the send and receive separately. That would have a total cost of $4(\sqrt{P} - 1)(\lambda + (N^2/P)t_0)$. Such solutions will get full credit.

(f) **(4 points)** If the communication time is less than or equal to the computation time, we'll assume that the communication can be fully overlapped by the computation. Let $\lambda = 10000$ and $t_0 = 10$, and $P = 64$. How large must N be for the communication time to be roughly the same as the computation time?
Your answer should be a number.

Hint: You don't need to figure out the exact answer. 75% credit if you work out the equation in N to solve no matter what your value is for N. 100% credit if you work out the equation in N, estimate N to within a factor of 2, and give a one or two-sentence justification for your answer.

Solve for N in

$$\begin{array}{rrcll}
 & \frac{N^3}{P} & = & 2(\sqrt{P} - 1)\left(\lambda + \frac{N^2}{P}t_0\right) & \\
\Rightarrow & N^3 & = & 2(\sqrt{P} - 1)\left(\lambda P + t_0 N^2\right) & \\
\Rightarrow & N^3 - 2(\sqrt{P} - 1)t_0 N^2 & = & 2(\sqrt{P} - 1)\lambda P & \\
\Rightarrow & N^3 - 140 * N^2 & = & 8.96 * 10^6, & \lambda = 10000, t_0 = 10, P = 64
\end{array}$$

N will need to be a little larger than $\sqrt[3]{8.96 * 10^5} \approx 210$. I'll try $N = 240$: $\frac{N^3}{P} = 21600$; $(\sqrt{P} - 1)\left(\lambda + \frac{N^2}{P}t_0\right) = 266000$. The communication time is larger than the computation time (by about 23%). Hey – that's good enough according to the problem statement!
With more tries, we find that computation time dominates communication time for any $N \geq 266$. That means any answer between 133 and 532 gets full credit, and we don't have to make special cases for answers that differ in their solution to Question 1e from the "official" one by a factor or two.

2. **Convolution** (40 points)
Often, when computing the convolution of sequences x and y, it is convenient to think of x as being a sequence

with 0-based indices: $x = x_0, x_1, \ldots, x_{n-1}$, and y being a sequence with indices *centered* at 0: $y = y_{-k}, y_{-(k-1)}, \ldots, y_{-1}, y_0, y_1, \ldots, y_k$. Let $z = x\#y$ denote the convolution of x and y, and we have:

$$z_i \quad = \quad \sum_{j=-k}^{k} x_{i-j} y_j$$

where we treat x_{i-j} as 0 if $i - j < 0$ or $i - j \geq n$.

Although z_i could have non-zero elements for any i with $-k \leq i < (n-1) + k$, we often just want the values of z_i for $0 \leq i < n$. That way, z is an array of the same size as x. Figure 1 shows a CUDA implementation of this convolution computation.

Simple Example:

(a) **(2 points)** Let $n = 5$, $k = 1$, $x_i = i$, $y_{-1} = -1$, $y_0 = 2$, $y_1 = 1$. Let $z = x\#y$.

i. What is z_0?

$z_0 = x_0 y_0 + x_1 y_{-1} = 0 * 2 + 1 * (-1) = -1$

ii. What is z_2?

$z_2 = x_1 y_1 + x_2 y_0 + x_3 y_{-1} = 1 * 1 + 2 * 2 + 3 * (-1) = 2$

Control flow:

(b) **(2 points)** What is the purpose of the `if`-statement on line 6 in Figure 1? In other words, what could go wrong if that test were omitted?

This `if` makes sure that the CUDA thread corresponds to an in-bounds element of `x` and `z`. This is because the number of threads in a block is "rounded-up" to accommodate various CUDA details such as warp size.

(c) **(4 points)** What is thread divergence? Give a short (i.e. two or three sentence) explanation.

Thread divergence occurs when different threads in the same warp take different execution paths – for example, some take the then-branch of an if-statement and others take the else-branch.

(d) **(4 points)** Give two examples of statements that could cause thread divergence in Figure 1. For both of those statements, give conditions that cause the divergence – for example,

"If <state values for some variables>, then <state what happens>"

1. The `if`-statement at line 6 – threads that are out-of-range for the `x` and `z` arrays will do nothing while the other threads in the warp perform the convolution.
2. The `if`-statement at line 18 – threads corresponding to elements of `z` near the ends of the array will skip the multiply-add for elements of `x` that are off the end of the array and treated as if they were 0.

__syncthreads:

(e) **(4 points)** What does the `__syncthreads()` function do? Give a short (i.e. two or three sentence) explanation.

`__syncthreads()` implements a barrier. All threads in a block must reach this barrier before any can continue after it.

(f) **(4 points)** The code in Figure 1 needs a call to `__syncthreads()`. State where the call should be added and give a short explanation (no more than five sentences).

A call to `__syncthreads()` is needed after the `x_sh` and `y_sh` arrays have been loaded (i.e. after line 14), and before the values in these arrays are used (i.e. before line 16). This is because threads in the block access read values from these shared memory arrays that were written by other threads in the block.

Detail: a correct implementation of the kernel would include a '}' to match the '{' on line 6 so all threads in the block will reach the barrier. There would then need to be a new '`if(myId < n) {`' after the call to `__syncthreads()`. The question didn't ask for this level of detail; so, we won't require it in solutions.

Memory:

(g) **(1 points)** Is the reference to `x[myId]` on line 8 in Figure 1 a reference to global memory or shared memory?
global

(h) **(3 points)** If the reference to `x[myId]` on line 8 in Figure 1 is a reference to global memory, is it a *coalesced* reference? If it is a reference to shared memory, does it cause a bank-conflict? Give a one or two sentence justification for your answer.
Coalesced. Consecutive threads in a warp have consecutive values of `myId` because of the '`+ threadIdx.x` on line 4.

(i) **(1 points)** Is the reference to `y_sh[i]` on line 19 in Figure 1 a reference to global memory or shared memory?
shared

(j) **(3 points)** If the reference to `y_sh[i]` on line 19 in Figure 1 is a reference to global memory, is it a *coalesced* reference? If it is a reference to shared memory, does it cause a bank-conflict? Give a one or two sentence justification for your answer. No bank conflict. Threads in the same warp have the same value of `i` because they execute in lockstep. They have the same value for `k` because it is the same for all threads in the kernel. Thus, all threads in the warp are reading from the same location in shared memory.

Kernel launch:

(k) **(3 points)** Complete the kernel launch on line 32 in Figure 1. In particular, write in your the CUDA code that should replace `your_answer(Question1(j)i)` here:
`conv_kernel<<<ceil(n/1024.0),1024>>>(n, k, dev_x, dev_y, dev_z);`

CGMA:

(l) **(3 points)** What is the CGMA for `conv_kernel` when `n=1024` and `k=4`?

Number of floating point operations: `n*(2*k+1) - k*(k-1)`. The $k * (k-1)$ are for the operations that are "skipped" when `ii<0` or `ii >= n`. With $n = 1024$ and $k = 4$, there are 9204 floating point operations.
The code reads `n` values from `x`, reads `2*k+1` values from `y`, and writes `n` values to `z`. Thus, there are `2*(n+k)+1` memory references. With $n = 1024$ and $k = 4$, this means there are 2057 memory references.
The CGMA is $4084/2057 \approx 4.474$.

(m) **(3 points)** Write a formula for the CGMA for `conv_kernel` in terms of `n` and `k`.

Most of my derivation is in my answer to Question 1(k)i:

$$\begin{aligned} \text{CGMA} &= \frac{n(2k+1)-k(k-1)}{2*(n+k)+1} \\ &= \frac{2nk+n-k^2+k)}{2n+2k+1} \\ &= k+\frac{1}{2}-\frac{3k^2+k+1/2}{2(n+k)+1} \end{aligned}$$

(n) **(3 points)** What is the CGMA for `conv_kernel` when

i. `n=1024` and `k=90`?
55.3

ii. `n= 256` and `k=90`?3cm
79.6 – nearly the CGMA=80 that we need to keep a GTX 1060 fully utilized.

3. **Sorting on a GPU** (15 points) Consider sorting on an array of `floats` on a GPU using the bitonic sort algorithm. For its artistic value, Figure 2 shows the sorting network for bitonic sort with 8 inputs.

(a) (**1 point**) How many compare-and-swap operations are performed by the 8-input bitonic sort? 24 – just count them in the figure

(b) (**1 point**) What is the span for the 8-input bitonic sort? 6

(c) (**2 points**) How many compare-and-swaps are performed by a N-input bitonic sort? Check one of the following:

- ☐ N
- ☐ $N \log_2 N$
- ☒ $\frac{N}{4}(\log_2 N)(1 + \log_2 N)$
- ☐ $\frac{N}{2}(\log_2 N)^2$
- ☐ $\frac{N^2}{2}$
- ☐ $N!$

(d) (**4 points**) A compare-and-swap of `x` and `y` can be performed as `min(x,y)` and `max(x,y)`. Assume that `min(x,y)` counts as a single floating point operation, and likewise for `max(x,y)`. Thus, a compare-and-swap counts as two floating point operations. How many floating operations are performed by bitonic sort for an input with 2^k values?
$\frac{N}{2}(\log_2 N)(1 + \log_2 N)$

(e) (**2 points**) If we load 2^k values from the global memory into shared memory, sort them, and then write the sorted result back to the global memory, what is the total number of global memory accesses performed by the sort? 2^{k+1}

(f) (**2 points**) On a GPU, we will store the data to be sorted in the shared memory of an SM. A block can have at most 48 Kbytes of shared memory, and a `float` has a size of 4 bytes. What is the largest integer k such that 2^k `floats` can be stored in the shared memory of one bloc?

$k = 13$.
$48\,\text{Kbytes}/(4\,\text{bytes}/\texttt{float}) = 12288\,\texttt{float}s$. The largest power of 2 that is less than or equal to 1228 is $8192 = 2^{13}$.

(g) (**2 points**) For the value of k from your answer to Question 1(l)vi, what is the CGMA?
Note: this is the CGMA for a single block. We could perform a sort of more than 2^k elements by sorting groups of 2^k as described here, and launching a subsequent kernel or two to merge those results. That would make this question way more complicated. Furthermore, the CGMA for a single block is pretty close to the CGMA for the entire computation – adding more blocks scales up the number of floating point operations and the number of memory accesses at the same rate.

$$\begin{aligned}
\text{CGMA} &= \frac{\#\text{floating}-\text{pointoperations}}{\#\text{globalmemoryaccesses}} \\
&= \frac{2^{k-1}k(k+1)}{2^{k+1}}, && k = \log_2 N \\
&= \frac{k^2+k}{4} \\
&= 43, && k = 13
\end{aligned}$$

(h) (**3 points**) For GPUs like the GTX 1060s in the `linXX` machines, is the critical bottleneck for bitonic sort memory bandwidth or the number of computations performed? Is it likely that another algorithm will be much faster than bitonic sort when run on a GPU? Give a short justification for your answer.

The GTX 1060 requires a CGMA of 80 or more to keep the SPs fully utilized. Bitonic sort achieves about half of that. Thus, memory bandwidth will probably be the critical bottleneck. Any other algorithm will need to read the unsorted data from the global memory and write the sorted result back. Thus, any algorithm will have at least as many global memory accesses as bitonic sort. It is unlikely that another algorithm will run faster than bitonic sort on the GPU.

4. **Bitonic Sequences** (10 points)

(a) **(6 points)** Let $n > 0$, and let $x_0, x_1, \ldots, x_{n-1}$ be a bitonic sequence of n 0s and 1s. You can assume that x is of the form $0^*1^*0^*$. Define y as the sequence with:

$$\begin{array}{rcl} y_0 & = & x_{n-1} \\ y_i & = & x_{i-1}, \quad 1 \leq i < n \end{array}$$

Prove that y is bitonic.

Case $x_{n-1} = 0$: Then x is of the form $pqr0$ where $p, r \in 0^*$ and $q \in 1^*$. We have that $y = 0pqr \in 0^+1^*0^*$. Thus y is bitonic.

Case $x_{n-1} = 1$: Then x is of the form $pq1$ where $p \in 0^*$ and $q \in 1^*$. We have that $y = 1pq \in 10^*1^*$. Thus y is bitonic.

(b) **(4 points)** Let $x_0, x_1, \ldots, x_{n-1}$ be a bitonic sequence of n numbers, and let y be defined as in Question 1(m)i. Show that you can choose x such that y is not bitonic.

Let $x = [0, 3, 2, 1]$. x is bitonic. By the construction for y, $y = [1, 0, 3, 2]$ which is not bitonic.
Observation: From part a, we get that if a sorting network produces a correctly sorted output for any *bitonic* intput, then it generates a correctly sorted output for any cyclic rotation of a bitonic input. In particular, a bitonic-merge will generate a correctly sorted output for any cyclic rotation of a bitonic input. From part b, we get that a cyclic rotation of a bitonic sequence is not necessarily bitonic. But, a bitonic merge will sort it correctly anyway!

5. **Other Stuff** (15 points)

(a) Fused multiply-add **(4 points)**

i. **(1 point)** What is a fused multiply-add?
A fused multiply add is a hardware optimization that allows a multiply followed by an add to be performed as a single operation.

ii. **(1 point)** When counting the number of floating point operations performed by a computation, how many floating point operations is one fused multiply-add?
2

iii. **(1 point)** Give an example of an expression consisting of one multiply and one add where the operations **can** be combined into a single fused multiply-add operation.
`a*x + b`

iv. **(1 point)** Give an example of an expression consisting of one multiply and one add where the operations **cannot** be combined into a single fused multiply-add operation.
`a*(x + b)`

(b) **(6 points)** Consider a problem whose sequential version takes time $N^{3/2}$. A parallel implementation running on P takes time time $\frac{N^{3/2}}{P} + (\sqrt{N} + \lambda)\log_2 P$. Let $N = 10^6$ and $\lambda = 10^4$. Calculate the speed-up for

i. $P = 2^6 = 64$

$$\begin{array}{rcll} SpeedUp & = & \frac{T_{seq}}{T_{par}} & \\ & = & \frac{N^{3/2}}{(N^{3/2}/P)+(\sqrt{N}+\lambda)\log_2 P} & \\ & = & \frac{P}{1+(1+\lambda/\sqrt{N})P(\log_2 P)/N} & \\ & = & \frac{P}{1+1.1*10^-5P\log_2 P}, & N = 10^6, \lambda = 10^4 \\ & \approx & 63.73, & P = 64 \end{array}$$

ii. $P = 2^{10} = 1024$
920

iii. $P = 2^{14} = 16384$2cm
4650

(c) True or false **(4 points)**

true If a cache line in the MESI protocol is in state M, then no other cache has data for this address.

false The cross-section width of a $N \times N$ 2D-mesh is $\Theta(N \log N)$.

true Using energy scaling, it is possible to have a parallel algorithm that performs more operations in less time and using less energy than the best sequential algorithm.

false All blocks of a CUDA grid must execute at the same time.

(d) Confused multiply-add **(1 point)**.
Give an example of a confused multiply-add. Extra credit will be considered for creative answers.
`banana*aardvark + boat = 42`

```
__shared__ float x_sh[1024];
__shared__ float y_sh[1024];

// convolution kernel
__global__ void conv_kernel(uint n, int k, float *x, float *y, float *z) {
   uint myId = blockDim.x*blockIdx.x + threadIdx.x;
   uint k2p1 = 2*k+1;
   if(myId < n) {
      // load x
      float xx = x[myId];
      x_sh[myId] = xx;
      // load y
      if(myId < k2p1) {
         float yy = y[myId];
         y_sh[myId] = yy;
      }
      float sum = 0.0f;
      for(int i = -k; i <= k; i++) {
         int ii = myId - i;
         if((0 <= ii) && (ii < n))
            sum += x_sh[ii] * y_sh[i+k];
      }
      z[myId] = sum;
   }
}

// wrapper function to call from C
void conv(uint n, int k, float *x, float *y, float *z) {
   size_t sz_x = n*sizeof(float);
   size_t sz_y = k*sizeof(float);
   cudaMalloc((void**)(&dev_x), sz_x);
   cudaMalloc((void**)(&dev_y), sz_y);
   cudaMalloc((void**)(&dev_z), sz_x);
   cudaMemcpy(dev_x, x, sz_x, cudaMemcpyHostToDevice);
   cudaMemcpy(dev_y, y, sz_y, cudaMemcpyHostToDevice);
   conv_kernel<<<your_answer(Question1(j)i),1024>>>(n, k, dev_x, dev_y, dev_z);
   cudaMemcpy(z, dev_z, sz_x, cudaMemcpyDeviceToHost);
   cudaFree(dev_x);
   cudaFree(dev_y);
   cudaFree(dev_z);
}
```

Figure 1: CUDA implementation of convolution

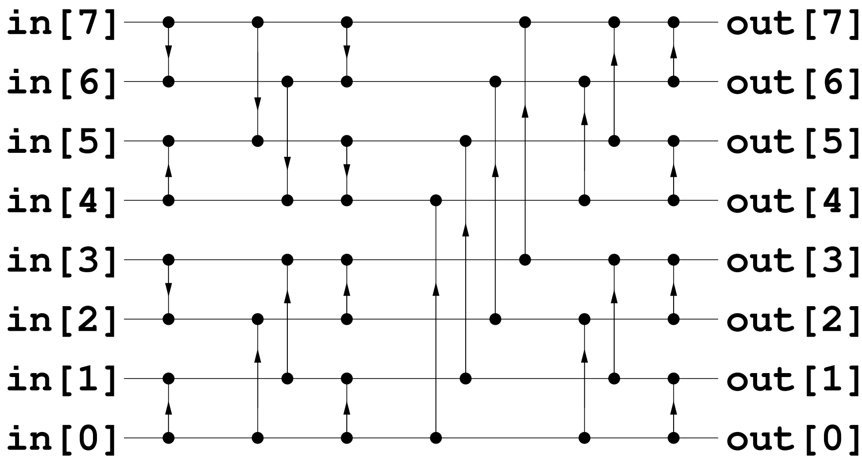

Figure 2: Bitonic Sort with 8 inputs