CpSc 418 **Midterm** October 22, 2018

Graded out of **62 points**

Time for the exam: 50 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, as a calculator, or any other purpose. Likewise, the use of computers, tablets, etc. is not permitted during the exam.

Q	Pts
0.	
1.	
2.	
3.	
4.	
5.	
Total	

No Test books: We have included space with each question for you to write your answers. There are a few blank pages at the end that you can use if you need more space. Note that the space that we provide is intended to be generous.

Bug bounties are in effect and given in midterm points. Only report a suspected error if it affects your ability to complete the exam. To report an error, raise your hand. Due to Mark's hearing difficulties, he may need to step out in the hall to hear you – he doesn't understand whispers. We will post any corrections to the whiteboard.

Minor spelling, grammar, and similar errors should be posted to piazza after the exam. If the "error" does not impact your understanding of what question is being asked of how to solve it, then it is a minor error.

Good luck!

CpSc 418 **Midterm** October 22, 2018

Answer all of the questions: question 0 and all five of questions 1 – 5.
Graded out of **62 points**.

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

   (a) Your name: _____

   (b) Your student number: _____

1. **Speed-Up** (15 points)
   This problem examines the speed-up of a typical reduce problem. The input consists of N data values. The sequential version takes T_1 operations per data value. The data is available to the sequential processes at the start of execution.

   The parallel version uses P worker processes. Each worker process has its part of the data stored locally at the start of the execution. Each worker process combines N/P values. Then, these results are combined using the binary-tree approach that we have used all term. Sending a message between processors takes the same amount of time as λ operations – this is the time from the send to completing the receive. For all of the questions below:

   - $N = 131,072 = 2^{17}$.
   - $P = 16 = 2^4$.
   - $\lambda = 16,384 = 2^{14}$.
   - T_{seq} is the time to execute the sequential program.
   - T_{par} is the time to execute the parallel program with P processors.

   We will consider two values for T_1: 32 and 8. Each question specifies which value of T_1 it uses.

   (a) (**3 points**) Let $T_1 = 32$. What is T_{seq}?

   (b) (**3 points**) Let $T_1 = 32$. What is T_{par}?

   (c) (**2 points**) Let $T_1 = 32$. What is the speed-up?

(d) (**4 points**) With a better compiler, we get $T_1 = 8$. What is the speed-up when $T_1 = 8$?

(e) (**1 point**) Of the four versions, sequential and parallel with $T_1 = 8$ or $T_1 = 32$, which one is the fastest (i.e. executes in the least time)?

(f) (**2 points**) Which parallel version ($T_1 = 8$ or $T_1 = 32$) has the greater speed-up? Give a one or two sentence explanation for the trend you observe between T_1 and speed-up.

2. **Message-Passing Architectures** (14 points)
Consider multiplying two $N \times N$ matrices, X and Y, to produce a matrix Z. We can compute Z by computing each element separately: Element $Z(I, J)$ is the dot-product of row I of matrix X with column J of matrix Y. This algorithm performs N^3 "operations" where an operation consists of a multiplication and an addition. For all questions about "time" in this problem, we consider the time for one multiply-and-add to be one time unit. For example, a sequential implementation takes N^3 time units.

We want to compute a parallel version using P message-passing processors. The P processors are arranged as a $\sqrt{P} \times \sqrt{P}$ mesh. Each link in the mesh consists of two, directed edges. Each edge can convey one matrix element per time unit. All edges can convey data in parallel. Furthermore, the network supports *broadcast* and *multicast* messages. If processor p_0 sends the same message to processors p_1, p_2, and p_3, the network can route that one message across links that are common to the paths for these three destinations – it doesn't send the same message three times.

We first consider an algorithm where each processor stores N/P rows of X and Y in its local memory. Each processor will compute N/P rows of Z. To do so, each processor sends *all* of its rows of Y to *all* of the other processors. Likewise, each processor receives all of the other rows of Y from the other processors. As described above, this is done with broadcast messages. After receiving Y, each processor performs N^3/P operations as described above and thus requires N^3/P time for *computation*. In the rest of this problem, we consider the time for *communication*.

(a) (**1 point**) What is the bisection width of the $\sqrt{P} \times \sqrt{P}$ mesh?
Note: this mesh has a total of $\sqrt{P} * \sqrt{P} = P$ nodes.

(b) (**3 points**) How much time is required to transmit the elements of matrix Y to all processors using this mesh? For this problem, only consider the cross-section bandwidth. Because the network supports broadcast, we only need to consider how much data crosses the bisection (i.e. how many matrix elements) even though this data is delivered to many processors on each side of the bisection.

(c) (**3 points**) Assume that each processor can transfer at most one element to and/or from the network per time unit. This is a constraint on the processor-to-network bandwidth. How long does it take to deliver the elements of Y to each processor?

(d) (**2 points**) Which constraint, cross-section bandwidth or processor-to-network bandwidth determines the communication time?

(e) (**4 points**) What is the value for P such that the communication time is equal to the computation time?

(f) (**1 point**) If P is greater than the answer to Question 2e, which will be greater, the computation time or the communication time?

3. **Dependencies** (5 points)

(a) (**1 points**) What is a read-after-write (RAW) dependency?

(b) (**2 points**) Dependencies can lower performance by preventing parallel execution of tasks. Consider read-after-write (RAW), write-after-write (WAW), write-after-read (WAR), and control (CTRL) dependencies. Renaming removes or reduces the impact of which of these types of dependencies?

(c) (**2 points**) Branch-prediction removes or reduces the impact of which of these types of dependencies?

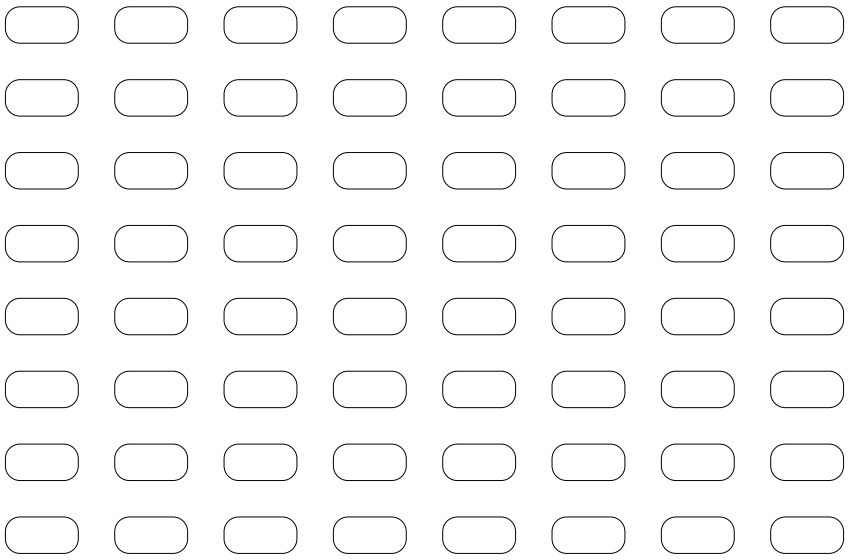
4. **The Work-Span Model** (18 points)

(a) (**4 points**) Consider the following code (in C) for computing the sum of the elements in a vector:

```
// vec_add0(v, n) – compute the sum of the elements of v.
// n is the number of elements of v.
void vec_add0(double *v, int n) {
  int i;
  double sum = 0.0;
  for(i = 0; i < n; i++)
    sum += v[i];
  return(sum);
}
```

Draw a work-span graph for this computation when `n==4` by labeling the vertices, and drawing arrows. All edges should go from a earlier row down to a later row, and each operation should be put in the earliest row allowed by the dependencies. For your convenience, I've provided labels you can use for the vertices. The template has more rows and columns of "vertex boxes" than needed for a correct solution. Only draw edges for RAW dependencies.

label	operation
`v[0]`	load `v[i]`, when `i=0`
`sum.0`	`sum+=`, when `i=0`
`inc.0`	`i++`, when `i=0`
`v[1]`	load `v[i]`, when `i=1`
`sum.1`	`sum+=`, when `i=1`
`inc.1`	`i++`, when `i=1`
`v[2]`	load `v[i]`, when `i=2`
`sum.2`	`sum+=`, when `i=2`
`inc.2`	`i++`, when `i=2`
`v[3]`	load `v[i]`, when `i=3`
`sum.3`	`sum+=`, when `i=3`
`inc.3`	`i++`, when `i=3`



(b) (**1 point**) What is the work for the Work-Span graph from Question 4a?

(c) (**1 point**) What is the span for the Work-Span graph from Question 4a?

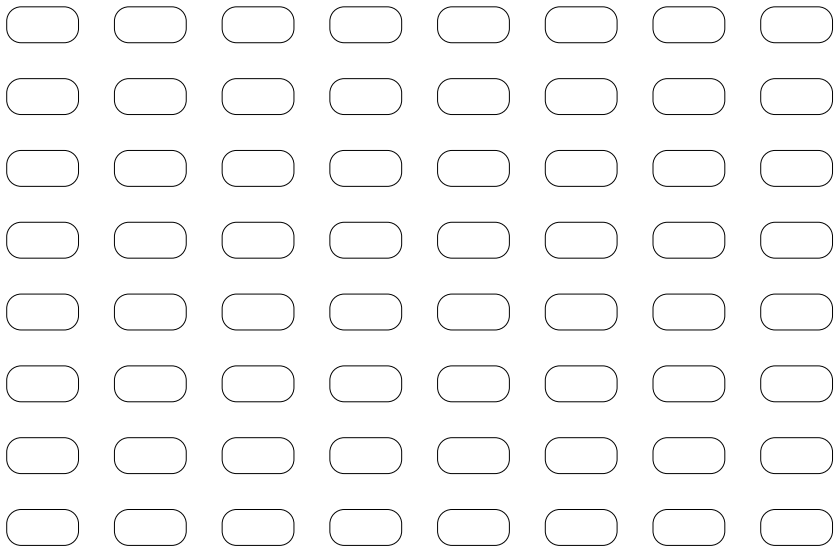
(d) (**1 point**) What is maximum speed-up for the Work-Span graph from Question 4a?

(e) (**4 points**)

```
// vec_add1(v, 1) – compute the sum of the elements of v.
// Assume: n is even.
void vec_add1(double *v0, int n) {
  int i, n2 = n/2;
  double sum0 = 0.0, sum1 = 0.0;
  double *v1 = &v0[n2];
  for(i = 0; i < n2; i++) {
    sum0 += v0[i];
    sum1 += v1[i];
  }
  return(sum0+sum1);
}
```

Draw a work-span graph for this computation when `n==4` (thus `n2=2`) by labeling the vertices, and drawing arrows. All edges should go from a earlier row down to a later row, and each operation should be put in the earliest row allowed by the dependencies. For your convenience, I've provided labels you can use for the vertices. The template has more rows and columns of "vertex boxes" than needed for a correct solution. Only draw edges for RAW dependencies.

label	operation
`v0[0]`	load `v0[i]`, when `i=0`
`v1[0]`	load `v1[i]`, when `i=0`
`sum0.0`	`sum0+=`, when `i=0`
`sum1.0`	`sum1+=`, when `i=0`
`inc.0`	`i++`, when `i=0`
`v0[1]`	load `v0[i]`, when `i=1`
`v1[1]`	load `v1[i]`, when `i=1`
`sum0.1`	`sum0+=`, when `i=1`
`sum1.1`	`sum1+=`, when `i=1`
`inc.1`	`i++`, when `i=1`



(f) (**1 point**) What is the work for the Work-Span graph from Question 4e?

(g) (**1 point**) What is the span for the Work-Span graph from Question 4e?

(h) (**1 point**) What is maximum speed-up for the Work-Span graph from Question 4e?

(i) (**4 points**) I tried running the `vec_add0` and `vec_add1` functions on my laptop with `n==1000` and 1,000,000 trials (a total of 1,000,000,000 additions). Using `vec_add0`, the total time was 1.03 seconds. Using `vec_add1`, the total time was 0.53 seconds. Why is `vec_add1` almost twice as fast as `vec_add0`? A short answer with two or three sentences is sufficient.

5. **Three fibs** (11 points)
The `fib1(N)` and `fib2(N)` functions below both compute the N^{th} Fibonacci number:

```
fib1(0) -> 0;
fib1(1) -> 1;
fib1(N) when is_integer(N), N > 1 -> fib1(N-1) + fib1(N-2).

% fib2a is a helper function for fib2
fib2a(0) -> {0,1};
fib2a(N) ->
  {FibN_1, FibN_2} = fib2a(N-1),
  {FibN_1 + FibN_2, FibN_1}.

fib2(N) when is_integer(N), N >= 0 ->
  {Ans, _} = fib2a(N),
  Ans.
```

Using `time_it:t` I got the following timing data:

N	time for `fib1(N)`	time for `fib2(N)`	time for `fib3(N)`
10	5.2 μs	2.2 μs	2.0 μs
15	39.5 μs	2.2 μs	2.1 μs
20	435.7 μs	2.4 μs	2.3 μs
25	5.0 ms	2.5 μs	2.4 μs
30	53.8 ms	2.6 μs	2.5 μs
35	579.1 ms	2.7 μs	2.6 μs
40	6.6 s	2.9 μs	2.7 μs
50	timeout	3.1 μs	2.9 μs
100	timeout	4.8 μs	4.3 μs
200	timeout	9.4 μs	8.4 μs
300	timeout	14.3 μs	12.8 μs
400	timeout	18.9 μs	17.3 μs
500	timeout	25.1 μs	21.7 μs

(a) (**1 point**) Which is faster, `fib1` or `fib2`?

(b) (**2 points**) Give a one or two sentence explanation for why one is faster than the other. Your answer should address the big-$\mathcal{O}$ difference between `fib1` and `fib2`. Which is faster, `fib1` or `fib2`?

(c) (**2 points**) Is `fib2a` head-recursive or tail-recursive?

(d) (**5 points**) Write a third implementation of computing the $\mathtt{N}^{th}$ Fibonacci number, call it `fib3`. `fib3` should call a helper function `fib3a`. `fib3` should have the same big-$\mathcal{O}$ complexity as `fib2`, and `fib3a` should be head-recursive if `fib2a` is tail-recursive, and vice-versa.
Hint: I included the run-times for my implementation of `fib3`.

```
fib3a(                          ) ->
```

```
fib3(N) ->
```

(e) (**1 point**) True, false, or poetry (any non-empty answer gets full credit): "For loops are evil."

☐ True

☐ False

☐ Other – please justify an answer of "Other" with a limerick or haiku.