CpSc 418 **Midterm – Take Home Version** 2017W2

- Write your name and student number at the top of this page.
- Complete up to **four** out of questions 1–5. If you write solutions for all questions, please indicate which four should be included in your grade. If you do not indicate which to include, the graders will choose which to include at their own convenience.
- Answer the questions in the space provided. **Do not** use additional sheets. For questions involving code, you are welcome to write, compile and test your code on a computer first, but the solution you submit must be written within the space provided on these pages.
- Normal course rules regarding collaboration and plagiarism apply. In particular, you may ask questions of the TAs or instructor and do not need to cite them; however, you should not look at the solution of another student or make your solution available as an aid to others. That includes code: You should not share code (handwritten, typed or otherwise) with anyone.
- For each question that you answer, we will grade it out of 25 according to the midterm rubric. If your score on that question in the in-class midterm on October 25 was $x$ and your score on this take-home version is $y$, then you will receive $\max(0, 0.2(y - x))$ as a bonus mark for that question on your midterm score. For example, if you scored 15/25 on the in-class ($x = 15$) and then got full points on the take-home ($y = 25$), you would receive $0.2(25 - 15) = 2$ bonus points, for a final score of 17/25.
- You need not choose the same four questions on the take-home as on the in-class. In many cases, the best strategy is to answer the questions on which you scored lowest during the in-class version (including the question that you skipped). If you scored 0/25 on a question during the in-class (including the question that you skipped), you can score up to $0.2(25 - 0) = 5$ bonus points.
- Your answers should be submitted in hardcopy at the beginning of class (2pm) on Monday November 6. Late submissions will not be accepted.
- Sign below to confirm that you understand these instructions. Submissions which are not signed will not be graded.

I, _____, confirm that I have read and understood the take-home exam instructions listed above.

1. **Erlang** (25 points) Erlang provides bitwise operation on integers. For example, `X bor Y` computes the bitwise-OR of integers `X` and `Y`; the result is an integer. Here are a few examples – in Erlang `2#0101` is the way we can write an integer in binary, and `2#0101 == 5`.

   ```
   2#0101 bor 2#1100 -> 2#1101.
   2#0001 bor 2#1000 -> 2#1001.
   ```

   The following function computes the bitwise-OR of all of the elements of a list:

   ```
   bit_or([]) -> 0;
   bit_or([Hd | Tl]) -> Hd bor bit_or(Tl).
   ```

   (a) **(5 points)** Is `bit_or` head-recursive or tail-recursive? Briefly justify your answer.

(b) **(5 points)** Write a second implementation that uses the other style – i.e. if `bit_or` is head-recursive, write a tail-recursive version. If it is tail-recursive, write a head-recursive version.

(c) **(10 points)** Write an implementation of `bit_or` using `lists:foldl`. Here's a template you can complete:

```
bit_or(List) ->

  lists:foldl(_____________________________________________________

    ___________________________________________________________

    ___________________________________________________________

    ___________________________________________________________

    ___________________________________________________________).
```

Hint: we provided more space for the answer than you will probably need.

(d) **(5 points)** The *cummulative-OR* of a list replaces each element of a list with the bitwise-OR of everything up-to-and including that element. For example,

```
cummulative_bor([2#0001, 2#1000, 2#1100, 2#0101]) ->
                [2#0001, 2#1001, 2#1101, 2#1101].
```

Write an implementation of `cummulative_bor` using `lists:mapfoldl`. Here's a template you can complete:

```
cummulative_bor(List) ->

  ___________ = lists:mapfoldl(_____________________________________

    ___________________________________________________________

    ___________________________________________________________

    ___________________________________________________________

    ___________________________________________________________),

  _____________________________________.
```

| element number $i$ | element bits | element integer | parity $0 \ldots i$ bits | parity $0 \ldots i$ integer |
|---|---|---|---|---|
| 1 | 10101010 | 170 | 10101010 | 170 |
| 2 | 00001111 | 15 | 10100101 | 165 |
| 3 | 11111111 | 255 | 01011010 | 90 |
| 4 | 00000000 | 0 | 01011010 | 90 |
| 5 | 10000000 | 128 | 11011010 | 218 |
| 6 | 00111100 | 60 | 11100110 | 230 |

Table 1: Example of computing the parity of a sequence of bytes for Question 2

2. **Reduce / Scan** (25 points)

   The *parity* of a sequence of bits is 1 if the number of ones in the sequence is odd, otherwise it is 0. Put another way, if the parity is treated as an extra bit at the end of a sequence, the augmented sequence will always have an even number of ones. The notion of parity can be extended to sequences of bytes (8 bits) by separately computing the parity of all of the first bits, all of the second bits, ..., all of the eighth bits (the parity of a sequence of bytes will be a byte). Table 1 shows an example sequence of bytes $i = 1 \ldots 6$ (in both their bitwise and integer representation) and the parity bytes computed over elements $0 \ldots i$ (in both their bitwise and integer representation).

   The parity of a sequence of bits can be computed with the *exclusive or* (often called "xor") operator. In Erlang, we can do this to larger collections of bits—such as bytes—using the `bxor` operator. For example, `170 bxor 15 = 165` and `165 bxor 255 = 90`. Compare these results to the integer columns of the first three rows of the sequence in table 1. Exclusive or and Erlang's `bxor` are both associative and commutative.

   You will complete the helper functions for `compute_parity(W, ListKeyIn, ListKeyOut)`, where

   - `W` is a list of worker nodes arranged in a tree.
   - `ListKeyIn` is a Key with which each worker node can retrieve its local portion of the input list from its `ProcState`. The input list will be bytes of data represented as integers in the range $[0, 255]$.
   - `ListKeyOut` is a Key under which each worker node should store its local portion of the output list in its `ProcState`. The output list will be parity bytes represented as integers in the range $[0, 255]$.

   The return value of `compute_parity()` should be the parity byte of the entire list represented as an integer in the range $[0, 255]$. We implement `compute_parity()` with the following call to `wtree:scan()`:

```
compute_parity(W, ListKeyIn, ListKeyOut) ->
  wtree:scan(W,
    fun(ProcState) -> cp_leaf1(wtree:get(ProcState, ListKeyIn)) end,
    fun(ProcState, AccIn) ->
      ListIn = wtree:get(ProcState, ListKeyIn),
      ListOut = cp_leaf2(ListIn, AccIn),
      wtree:put(ProcState, ListKeyOut, ListOut)
    end,
    fun(Left, Right) -> cp_combine(Left, Right) end,
    0).
```

   Assuming that there are $N$ elements in the input list spread evenly across $P$ workers in the worker pool `W` (where $P \ll N$), your implementation should achieve a speedup close to $P$. Your implementation should fail if there is no value associated with `ListKeyIn` or if the value associated with `ListKeyIn` is not a list of integers.

   You may call any Erlang built-in functions, and any functions from the `lists`, `misc`, `workers`, or `wtree` modules.

As an example, imagine that `W` contains two workers with the following data (the same data as in Table 1):

- On worker 1, `ProcState` contains the Key/Value pair `{listIn, [ 170, 15, 255, 0 ]}`.
- On worker 2, `ProcState` contains the Key/Value pair `{listIn, [ 128, 60 ]}`.

Then the call to `compute_parity(W, listIn, listOut)` will return the value `230` and the following additional data will be stored on the workers after the scan completes:

- On worker 1, `ProcState` will now contain the Key/Value pair `{listOut, [ 170, 165, 90, 90 ]}`.
- On worker 2, `ProcState` will now contain the Key/Value pair `{listOut, [ 218, 230 ]}`.

(a) (4 points) Briefly describe the segment summary that you plan to use.

(b) (5 points) Your code for `cp_combine(Left, Right)`:

(c) (8 points) Your code for `cp_leaf1(ListIn)`:

(d) (8 points) Your code for `cp_leaf2(ListIn, AccIn)`. Note that the calls to `wtree:get()` and `wtree:put()` are already done for you in `compute_parity()`.

3. **Architecture and other stuff** (25 points)

   (a) **(5 points)** The statements below describe the contents of caches for a shared memory multiprocessor using the MESI protocol. Mark each statement below T for true or F for false.

   _____ Only one cache can have valid data for a particular memory location at any given time.

   _____ Multiple caches can have read-only copies of the data for the same memory location at the same time.

   _____ Multiple caches can have write-only copies of the data for the same memory location at the same time.

   _____ Multiple caches can have read-and-writable copies of the data for the same memory location at the same time.

   _____ If a value in a cache is valid (i.e. can be read or written by the cache's processor), then it matches the value in main memory.

   (b) **(7 points)** Mark each statement below T for true or F for false.

   _____ A parallel implementation of an algorithm can be faster than the sequential version or use less energy, but not both as the same time.

   _____ A parallel implementation of an algorithm can be faster than the sequential version and use less energy.

   _____ Parallel algorithms are always faster than their sequential counterparts.

   _____ Parallel algorithms are always slower than their sequential counterparts.

   _____ A CPU can consume less energy per operation when operating at a lower frequency.

   _____ Power is energy per unit time.

   _____ Knowledge is power.

   (c) **(5 points)** For each message-passing network below, write whether the cross-section bandwidth grows as 1, $\log N$, $N$, $N^2$, or $N^3$. We're ignoring constant factors.

   _____ A $N \times N$ mesh (i.e. two dimensional)

   _____ A $N \times N \times N$ torus (i.e. three dimensional)

   _____ A $N$ processor ring.

   _____ A binary tree with $N$ processors at the leaves – the non-leaf nodes

   _____ A $d$-dimensional hypercube with $N = 2^d$ processors.

(d) **(8 points)** Answer each of the questions below with a single sentence or short phrase.

- **(2 points)** What does acronym "RAM" mean in the RAM model?

- **(2 points)** What does acronym "PRAM" mean in the PRAM model?

- **(2 points)** What does $\lambda$ represent in the CTA model. Note: the CTA model is the model we use most often in class.

- **(2 points)** Should chip manufacturers pay a fine if they don't comply with Moore's Law?

☺

4. **Performance** (25 points)

A common operation in iterative numerical computing (such as image processing, solving differential equations, etc.) is to take an array of data and update each element of the array with a function that depends on the values of that element and its neighbors. For this question we will consider the simplest form of such an update. We will work on a 2D square array of size $N \times N$, so there are $N^2$ elements. Each element will be updated using the values of that element and the four neighbours above, below, left and right of that element (the value of any neighbor beyond the edge of the array can be treated as zero). If we define a unit of time to be the time necessary to perform an update to a single element (assuming that the values of all four neighbours are known locally), then a serial implementation takes $N^2$ time to update all of the elements of the array.

Now we will consider creating a parallel version on $P$ processors. We will assume a fully connected network, so each processor can communicate directly with every other processor. To complete an update, each processor will need a copy of the data for any elements neighboring its own elements. Assume that sending or receiving $K$ elements takes time $\lambda + K$, but that messages between different pairs of processors can be overlapped and each link can carry messages in both directions at once; for example, if processor 1 sends a message of length $K_{1\to2}$ to processor 2 and a message of length $K_{1\to3}$ to processor 3 while receiving a message of length $K_{2\to1}$ from processor 2 and a message of length $K_{3\to1}$ from processor 3, the total time spent by processor 1 will be $\lambda + \max(K_{1\to2}, K_{1\to3}, K_{2\to1}, K_{3\to1})$.

(a) (5 points) Our first implementation will divide the array up by columns, so each processor will get $N/P$ complete and contiguous columns (each with $N$ elements).

i. For a processor working on some interior columns of the array (in other words, it has neighbors on both sides), how much data must it send and receive before all the processors can perform their updates?

ii. How much time will be spent for this communication?

iii. How much time will be spent on computation to complete the update?

(b) (5 points) Our second implementation will divide the array up into square blocks so that each processor gets $B = N^2/P$ elements arranged in a square (you may assume that $B$ is a perfect square).

i. For a processor working on some interior block of the array (in other words, it has neighbors on all four sides), how much data must it send and receive before all the processors can perform their updates?

ii. How much time will be spent for this communication?

iii. How much time will be spent on computation to complete the update?

(c) (3 points) Are there any conditions on $N$, $P$ and/or $\lambda$ under which the column partition would outperform the block partition? If yes, give the condition. If no, briefly explain why not.

(d) (4 points) Compute the speedup for the **block partitioned** version with:

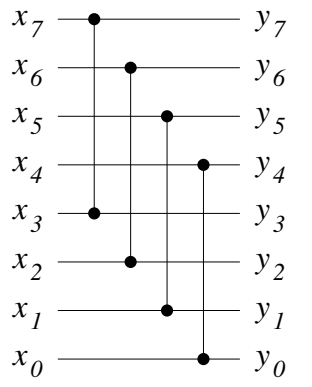
i. $N = 200$, $P = 100$, $\lambda = 100$.

ii. $N = 1000$, $P = 400$, $\lambda = 1000$.

(e) (4 points) Will the **block partitioned** algorithm display the behavior described by Amdahl's law? Briefly explain.

(f) (4 points) Will the **block partitioned** algorithm display the behavior described by Gustafson's law? Briefly explain.




a. The Sorting Network

| i. $j < i + (N/2)$ | ii. $j = i + (N/2)$ | iii. $j > i + (N/2)$ |
|---|---|---|
| *i=3, j=6* | $i =$ __, $j =$ __ | $i =$ __, $j =$ __ |
| $x_7 = 0,\ y_7 = 1$ | $x_7 =$ __, $y_7 =$ __ | $x_7 =$ __, $y_7 =$ __ |
| $x_6 = 0,\ y_6 = 0$ | $x_6 =$ __, $y_6 =$ __ | $x_6 =$ __, $y_6 =$ __ |
| $x_5 = 1,\ y_5 = 1$ | $x_5 =$ __, $y_5 =$ __ | $x_5 =$ __, $y_5 =$ __ |
| $x_4 = 1,\ y_4 = 1$ | $x_4 =$ __, $y_4 =$ __ | $x_4 =$ __, $y_4 =$ __ |
| $x_3 = 1,\ y_3 = 0$ | $x_3 =$ __, $y_3 =$ __ | $x_3 =$ __, $y_3 =$ __ |
| $x_2 = 0,\ y_2 = 0$ | $x_2 =$ __, $y_2 =$ __ | $x_2 =$ __, $y_2 =$ __ |
| $x_1 = 0,\ y_1 = 0$ | $x_1 =$ __, $y_1 =$ __ | $x_1 =$ __, $y_1 =$ __ |
| $x_0 = 0,\ y_0 = 0$ | $x_0 =$ __, $y_0 =$ __ | $x_0 =$ __, $y_0 =$ __ |

b. Examples (you need to fill in ii and iii)

Figure 1: Sorting Network for Question 5b

5. **Bitonic Sequences** (25 points)

   (a) **(6 points)** Consider the six sequences below. Which of them are **bitonic**? Your answer can just list the numbers for the bitonic ones (e.g. *i*, *ii*, etc.). Note: For this part we are using the same sequences as the in-class midterm, but we are requesting you to identify the **bitonic** sequences, **not** the monotonic sequences.

   i. [0, 0, 1, 1, 1, 1, 1, 0]
   ii. [0, 1, 0, 1, 0, 1, 0, 1]
   iii. [1, 1, 1, 1, 1, 1, 0, 0]
   iv. [1, 1, 1, 1, 1, 1, 1, 1]
   v. [1, 2, 3, 4, 3, 2, 1, 2]
   vi. [0, 1, 1, 2, 3, 5, 8, 0]

   Answer: _____

For the remainder of this problem, let $i$, $j$, and $N$ be integers with $0 \leq i \leq j \leq N$. Let $x_0$, $x_1$, ..., $x_{N-1}$ be the bitonic sequence with

$$\begin{aligned} x_k &= 1, \quad \text{if } i \leq k \text{ and } k < j \\ &= 0, \quad \text{otherwise} \end{aligned} \tag{1}$$

In English, $x$ is a sequence of the form zero or more 0s followed by zero or more 1s followed by zero or more 0s; $i$ is the index of the first 1; and j is the index of the first 0 following the 1s.

Let $y_0$, $y_1$, ..., $y_{N-1}$ be the sequence defined by:

$$\begin{aligned} y_k &= \min(x_k, x_{k+(N/2)}), \quad \text{if } 0 \leq k < (N/2) \\ &= \max(x_{k-(N/2)}, x_k), \quad \text{if } (N/2) \leq k < N \end{aligned} \tag{2}$$

This is the generalization of Figure 1.a for arbitrary, even $N$.

   (b) **(6 points)** For parts ii and iii of Figure 1.b, choose values of $i$ and $j$ that satisfy the given constraint – for these examples $N = 8$. Then, fill in the values for $x$ and $k$ according to Equations 1 and 2 above. We've completed part i of the figure to give you a worked example. If you want to save time writing, you can just write tall, skinny 0s and 1s that span several elements of a sequence for $x$ or $y$.

(c) **(8 points)** Show that if $x$ is bitonic, then either $y_k = 0$ for $0 \leq k < (N/2)$ or $y_k = 1$ for $(N/2) < k < N$ (or both). It is sufficient to prove this for $j \leq i + (N/2)$ and $j > i + (N/2)$. We've provided the proof for the $j \leq (N/2)$ case as an example so you know that you don't need any more detail than what we wrote. The $j > i + (N/2)$ case is similar.

Case $j \leq i + (N/2)$: I'll show that $y_0 \ldots y_{(N/2)-1}$ are all 0. Consider $k$ with $0 \leq k < (N/2)$. If $k < i$ then $x_k = 0$ and $y_k = \min(x_k, x_{k+(N/2)})$ is 0. Otherwise,

$$j - (N/2) \;\leq\; i \;\leq\; k < (N/2),$$

$x_{k+(N/2)} = 0$, and therefore $y_k = \min(x_k, x_{k+(N/2)}) = 0$.

Case $j > i + (N/2)$:

(d) **(5 points)** Show that the other half of $y$ is bitonic. In other words, if $y_0 \ldots y_{(N/2)-1}$ are all 0, then $y_{N/2} \ldots y_{N-1}$ form a bitonic sequence, and if $y_{N/2} \ldots y_{N-1}$ are all 1, then $y_0 \ldots y_{(N/2)-1}$ form a bitonic sequence. My proof uses the same two cases as in question 5c.