Functional and Parallel Programming Final Exam

Date: Tuesday, July 20th, 2021
Instructor: Rachata Ausavarungnirun

Problem 1 (30 Points):	
Problem 2 (30 Points):	
Problem 3 (25 Points):	
Problem 4 (25 Points):	
Total (100+5 points):	

Instructions:

1. This is a 30-hour exam. Any grade above 100 will be converted to your extra credit at the rate of 50% (to the maximum of 105).
2. Submit your work as a zip file on Canvas.
3. Because this is a 30-hour exam, I expect everyone to `test your code`. Hence, you must **submit the test cases for all the coding questions** along with the code and explain what each test case is used for. Test cases, along with the explanation of how you test your code, will factor into 10% of all the questions.
4. If not specified, input and output types are a part of the question. Please use appropriate input and output types that make sense for the purpose of the question.
5. Please clearly comment your code, especially if your code do not work perfectly,
6. Clearly indicate your final answer for each conceptual problem.
7. Our typical restriction for Scala and Rust packages are similar to Assignments 3 and 4.
8. **DO NOT CHEAT.** If we catch you cheating in any shape or form, you will be penalized based on **my plagiarism policy** ($N*$ 10% of your total grade, where N is the number of times you plagiarized previously).

Tips:

- **Read everything.** Read all the questions on all pages first and formulate a plan.
- **Be cognizant of time.** The submission site will close at 00.01AM on Saturday.
- **Show work when needed.** You will receive partial credit at the instructors' discretion.

1. Potpourri [30 points]

For this questions, please put your answers here and submit the pdf, or put an answer in a separated pdf named `potpourri.pdf`. Please be as brief as possible. Long answer that has a mixture of correct and incorrect statement can result in a lower grade than a brief but correct statement. Be fundamental.

(a) **Parallelism** [**5 points**]

Why is immutability generally good for parallelism?

(b) **Lending vs. Borrowing** [**10 points**]

Please list the differences between lending and borrowing, and provide one example each for how this can be done.

(c) **Parallel Programming Model [5 points]**

What are the key differences between the shared memory model and data parallel model?

(d) **What is a Stream? [5 points]**

Please define stream and give three examples.

(e) **Work and Depth Analysis [5 points]**

What is the key benefits of using work and depth to analyze the parallelism of your program?

2. Mystery Code [30 points]

For this question, we will look at our mysterious code (Credit for the code goes to P'Pitapat Chairoj, MUIC'59)

(a) What does the code below do? Please provide the answer below.

```
fn mystery(seq: &[u64], ret: &mut[u64], offset: u64) -> u64 {
    let mut r = offset;
    for i in 0..seq.len() {
        let t = seq[i];
        ret[i] = r;
        r += t;
    }
    r
}
```

Your friend suggest a parallel version of the mystery code above with the following fixes.

```
fn sequential_reduce(seq: &[u64]) -> u64 {
    let mut r= seq[0];
    for j in 1..seq.len() {
        r += seq[j]
    }
    r
}
```

```
fn parallel_mystery_util(seq: [u64], ret: &mut [u64], offset: u64) -> u64
{
    let n: usize = seq.len();
    \\ BEGIN HERE
    let l: usize = num_blocks(n, BLOCK_SIZE);
    if l <= 2 {
        return mystery(seq, ret, offset);
    }
    let sums: Vec<u64> = vec_init(l, &|i: usize, _| {
        let s = i * BLOCK_SIZE;
        let e = min((i+1) * BLOCK_SIZE, seq.len());
        sequential_reduce(&seq[s..e])
    }, BLOCK_THRESHOLD);
    \\ END HERE
    let mut tmp: Vec<u64> = vec_no_init(l);
    let total = mystery(&sums, &mut tmp, offset);
    sliced_for(seq, ret, BLOCK_SIZE, &|i:
    usize, s_chunk: &[u64], r_chunk: &mut [u64]|
    {
        mystery(s_chunk, r_chunk, tmp[i]);
    });
    return total;
}

pub fn parallel_mystery(xs: &Vec<u64>) -> Vec<u64> {
    let ret : Vec<u64> = vec_no_init(xs.len() + 1);
    let tot = parallel_mystery_util(xs, &mut ret, 0);
    ret[xs.len()] = tot;
    ret
}
```

(b) What is the wrong with the code above and how can you fix it so that they compile? Hint: The number of wrong things here is between 1 and 4.

(c) Now that the code compile, please focus on the code block between `BEGIN HERE` and `END HERE`. What is the purpose of the code in that block and how does it contribute to parallelizing our mystery code?

3. Playing with Rust [25 points]

This question ask you on two unrelated topics.

(a) **isPrime Optimization [15 points]**

Write a function `isPrime(n: u64): bool` that returns true if `n` is a prime number and false otherwise. Please feels free to change the function signature if you need to apply borrowing/leading.

Please save the file with the name `parallelIsPrime.rs`

(b) **Parallel Primes [10 points]**

Implement a function `filterPrimes(input: [u64]): [u64]` that filter out only prime numbers in the vector of unsigned `intinput` **in parallel**. Feels free to use the solution to `parallelIsPrime` in this question. Note: Please feels free to change the function signature if you need to apply borrowing/leading.

Please save the file with the name `parallelFilterPrimes.rs`

4. Sorted List, Again [25 points]

In this question, you are going to implement a more generic sorted list in Rust. **Please feels free to change the function signature if you need to apply borrowing/leading. We intentionally take out the borrowing/lending part of the code to make sure you can apply these correctly.**

Please note that [x] means a vector of type `x`.

Below is the definition of our sorted list:

- A sorted list can either be `Empty` or contain multiple instances of elements that has total ordering property.
- For the purpose of implementing this list, you are not allowed to use the Scala's build-in List.
- For the purpose of getting things done, if you want to stick with a SortedList of Integer, feels free to do so but be aware that you will not be able to get a full credit for this question.

(a) (5 points) Implement an `insertList(x: i64, L: [i64]):[i64]` function that **takes an input value and a sorted vector** (`L`), and insert this value as a new element of the list while ensuring you preserve the SortedList's property.

(b) (5 points) Implement an `searchParallel(x: [i64], L: [i64]): [bool]` function that performs a search in parallel **for all element** `x` **on the input List** `L` and return a vector of boolean, where each elements has a value true if that element in `x` is in `L` and false otherwise.

(c) (5 points) Implement an `filter(x: i64, L: [i64]): [i64]` function that **takes an input** `x` **and return a List that contains only elements that is lower than** `x`.

(d) (5 points) Implement a function `numWrongOrder(L: [i64]): i64` that takes an input vector that does not always have the SortedList property. This function returns the smallest number of elements that should be deleted in order to make the list sorted again.

(e) (5 points) Once everything is all done, please write your own test cases that thoroughly test your code **so that it also works in corner cases**. Your test should shows, as best as possible, that your implementation works as intended.

Please save the file with the name `sortedlist.scala`