Assignment : Extracting Parallelism

The purpose of this assignment is for you

- to develop insight about how to extract parallelism from simple codes.
- to recognize cases where some form of parallelism may not be correct
- to acknowledge that sometimes adding work is necessary

Note: Often when talking about algorithms, the weights of tasks are denoted with their complexity.

Note: 1 and 2 are exercise/warm up. 3, 4 and 5 are harder. All problems are independent; so if you get stuck on one of them, try the other ones.

The thresholds for this assignments are: $A >= 75$; $B >= 60$; $C >= 50$; $D >= 35$

1 Transform (15 pts)

Consider the transform function:

```
void transform (int* a, int* b, int n) {
  for (int i=0; i<n; ++i)
    b[i] = f(a[i]);
}
```

Question: Extract the dependencies. Assume the call to `f` cost O(1). Assume calls to `f` are always independent. (Note: Yes this problem is VERY simple!)
Question: What is the width? the critical path? the work?
Question: How does a schedule look like on P processors? (What I mean is that what ever the values of n and P, the schedules have "shapes". What "shape" does any schedule for this problem have? The sketch of a Gantt chart work.)

2 Coin Collection (from Midterm Spring 2018) (20 pts)

The Coin Collection problem is defined as follows:

Several coins are placed on an $n \times m$ board with at most one coin per cell of the board. A robot is initially located at the upper left cell of the board. The robot can only move to the right or down; it can not move up or left. When the robot visits a cell where a coin is located, it picks it up. At most, how many coins can the robot collect?

This problem can be solved by the following method:

```
void RobotCoin(int n, int m, //size of the board
               int C[n][m] //Is there a coin in (i,j)
               ) {

  int F[n][m]; //How many coins can be collected while on (i,j)

  F[0][0] = C[0][0];

  for (int k=1; k<m; ++k) {
    F[0][k] = F[0][k-1] + C[0][k];
  }

  for (int i=1; i<n; ++i) {
    F[i][0] = F[i-1][0] + C[i][0];
    for (int j=1; j<m; ++j) {
      F[i][j] = max (F[i-1][j], F[i][j-1]) + C[i][j];
    }
  }

  return F[n-1][m-1];
}
```

Question: What is the complexity of this function?
Question: Extract and represent the graph of dependencies of `RobotCoin`. (Hint: make one task per possible k and (i, j) pair.)
Question: What is the work, width and critical path of this graph?

3 Reduce (20 pts)

Consider the reduce function:

```
template<typename T, typename op>
T reduce (T* array, size_t n) {
  T result = array[0];
  for (int i=1; i<n; ++i)
    result = op (result, array[i]);
  return result;
}
```

Do not be scared by the syntax! In C++, templates allow you to replace types and values in a piece of code by a type or a value known at compilation time. This is similar to generics in Java.

So if you define `T` as `int` and `op` as `sum`, it boils down to computing the sum of the array. You could use `op` as `max` and compute the maximum value of the array.

3.1 int, sum

Consider first the `int, sum` case which computes the sum of an array of integers.
Question: Extract the dependencies of this problem. What is the width? the critical path? the work?
Question: Assuming you have P processors, rewrite the code to introduce one local variable per processor to store partial computation. Extract the dependencies now. What is the width, critical path and work ?
Question: What does a schedule look like on P processors?

3.2 Variants

Question: Would that parallel version be correct for `int, max`? Why?
Question: Would that parallel version be correct for `string, concat`? Why?
Question: Would that parallel version be correct for `float, sum`? Why?
Question: Would that parallel version be correct for `float, max`? Why?

4 Prefix sum (20 pts)

Prefixsum is an algorithm that has many uses in parallel computing. The algorithm computes $pr[i] = \sum_{j<i} arr[j], \forall 0 \leq i \leq n$ and is often written sequentially:

```
void prefixsum (int* arr, int n, int* pr) {
  pr[0] = 0;
  for (int i=1; i<=n; ++i)
    pr[i] = pr[i-1] + arr[i-1];
}
```

Question: Extract the dependencies of this problem. What is the width? the critical path? the work?
Question: How can you make it parallel? (Hint: It is one of the "creative" cases where the algorithm needs to change. You have to add some work without changing the complexity in Big-Oh notation. A single pass on the array is not enough.)

5 Merge Sort (25 pts)

Question: Recall the merge sort algorithm. (Give the algorithm.)
Question: Extract dependencies on the merge sort algorithm. Do all tasks have the same processing time? What is the critical path, work, and width? (Hint: instead of using loop iterations as a task, you can use function calls and function return as tasks. Think that merge sort is recursive!)
Question: How does the schedule of such an algorithm look like when $P = 4$? (What I mean is that what ever the values of n, the schedules have "shapes". What "shape" does any schedule for this problem have? The sketch of a Gantt chart work.)
Question: Can you make it more parallel by rewriting the code? (Hint: You may need to increase the amount of work slightly. This is a "creative case" where the algorithm need to somewhat change.)