# WRITTEN

1.1 $q_1$: fraction of work that must be sequential: $\frac{\text{SEQ. WORK}}{\text{TOTAL WORK}}$

$q_2$: fraction of work that can use @ most 2 processors: $\frac{\text{2 proc. WORK}}{\text{TOTAL WORK}}$

$\underbrace{(1-q_1-q_2)}_{q_3}$: fraction of WORK that can use unbounded # of processors:

I think this might be parallel?

ⓐ Provide the best possible (smallest) upper bound on the Speedup as a function of $q_1$ and $q_2$.

We know that speedup $\leq \frac{1}{q_1}$.

Let $q_3 = (1-q_1-q_2)$.

Let $Q_1 = q_1 \cdot \text{TOTAL WORK}$.

Let $Q_2 = q_2 \cdot \text{TOTAL WORK}$.

Let $Q_3 = q_3 \cdot \text{TOTAL WORK}$.

We know that $CPL = Q_1 + \frac{Q_2}{2} + \frac{Q_\infty}{\infty}$ because $q_1$, $q_2$, and $q_3$ are disjoint.

We know that $\text{speedup} = \frac{\text{total WORK}}{CPL}$.

We can then say that

$$\text{speedup} = \frac{\text{total WORK}}{Q_1 + \frac{Q_2}{2} + \frac{Q_\infty}{\infty}}.$$

Putting this in terms of $q_1$ and $q_2$, we get

$$\text{speedup} = \frac{\cancel{\text{total WORK}}}{(q_1 \cdot \cancel{\text{TOTAL WORK}}) + \frac{(q_2 \cdot \cancel{\text{TOTAL WORK}})}{2} + \frac{(q_3 \cdot \cancel{\text{TOTAL WORK}})}{\infty}}$$

$$= \frac{1}{q_1 + \frac{q_2}{2} + \cancel{\frac{1 - q_1 - q_2}{\infty}}}$$

anything divided by $\infty$ is 0.

$$= \boxed{\frac{1}{q_1 + \frac{1}{2} q_2}}$$

b)

By definition, upper bound on speedup assumes that work can be divided into sequential and parallel portions.
This is the upper bound because we said that $q_2$ has exactly two processors and is thus perfectly parallel.

Case 1: $q_1 = 0$
↳ exclusively parallel

$$\text{speedup} \leq \frac{1}{\frac{q_2}{2}} = 1 \cdot \frac{2}{q_2} = 2 \cdot \frac{1}{q_2}$$

Case 2: $q_2 = 0$
↳ exclusively sequential

$$\text{speedup} \leq \frac{1}{q_1}$$

Case 3: $q_1 = 1$
↳ exclusively sequential

$$\text{speedup} \leq \frac{1}{q_1}$$

Case 4: $q_2 = 1$
↳ exclusively parallel

$$\text{speedup} \le \frac{1}{\frac{1}{2}q_2} = 2 \cdot \frac{1}{q_2}$$

In case 1 vs. Case 2 and case 3 vs. case 4, it logically makes sense that the speedup will be 2x as large when the sequential work is divided between two processors.
When $q_1 = 1$ and $q_2 = 0$, we get speedup $\le 1$.
When $q_1 = 0$ and $q_2 = 1$, we get speedup $\le 2$.

If the program can use at most two processors the program cannot be spedup by more than a factor of 2.
This proves that my speedup is the correct upperbound.

☐

## 1.2

```
count0 = 0;
accumulator a = new accumulator(SUM, int.class);
finish(a) {
    for(int i=0; i ≤ N-M; i++) async {
        int j;
        for(  j=0; j<M; j++) {
            if(text[i+j] != pattern[j]) {
                break;
            }
        }
        if (j==M) {     if hasn't broken, its a match! increment counts
            count0++;
            a.put(1)
        }
        count1 = a.get();
    }
}
count2 = a.get();
```

Count0 = [0,...,Q]. Assuming that the program works correctly, we know that the if statement on line 11 will pass Q times. Since count0 is a data race, we are not guaranteed the same value of count0 on each execution. We know that the absolute most that count0 can be is Q. Given that Q is greater than zero, the lowest value that count0 could have is one (because count0 is incremented by one). If Q is 0, then count0 can have the value of zero. Since Q is the maximum number of patterns found, count0 does not rely on N or M.

Count 1 = 0 because the accumulator get returns the initial value when called inside the finish. Since count1 cannot be anything but zero, count1 does not rely on N, M, or Q.

Count 2 = Q because count2 properly uses and increments the accumulator, thus causing no data races and always getting the same correct output, Q. Since the correct answer is Q, count2 does not rely on N or M.