MPCS 52060: Parallel Programming Assignment 3

Justin Cohler

April 24, 2019

SAQ 1

(Ch.1 Exercise 1) For each of the following, state whether it is a safety or liveness property. Identify the bad or good thing of interest.

1. Patrons are served in the order they arrive.

   (Safety) - Sequential ordering is a safety concern.

2. What goes up must come down.

   (Liveness) - Eventual correctness is a liveness concern.

3. If two or more processes are waiting to enter their critical sections, at least one succeeds.

   (Liveness) - This is essentially what deadlock freedom denotes.

4. If an interrupt occurs, then a message is printed within one second.

   (Safety) - A sequence of events occurs correctly (a message printed in time).

5. If an interrupt occurs, then a message is printed.

   (Liveness) - Eventual correctness of a message being printed is a liveness concern.

6. The cost of living never decreases.

   (Safety) - Sequential correctness is a safety concern.

7. Two things are certain: death and taxes.

   (Liveness) - Eventually, both of these things are realized, therefore this is a liveness concern.

8. You can always tell a Harvard man.

   (Safety) - Correctness in this case regards safety.

SAQ 2

(Ch.1, Exercise 7) Running your application on two processors yields a speedup of S2. Use Amdahls Law to derive a formula for Sn, the speedup on n processors, in terms of n and S2.

Using Ahmdal's Law:

$$s_2 = 1/(1 - p + p/n) = 1/(1 - p + p/2) = 1/(1 - p/2)$$

$$s_2 = 1/(1 - p/2)$$

$$s_2 - s_2 * p/2 = 1$$

$$s_2 * p/2 = s_2 - 1$$

$$p = 2 * (s_2 - 1)/s_2$$

Now we've solved p in terms of s_2. Let's plug s_2 back in for s_n:

$$s_n = 1/(1 - 2(s_2 - 1)/s_2 + 2(s_2 - 1)/s_2/n)$$

$$= 1/(1 - 2(s_2 - 1)/s_2 + 2(s_2 - 1)/ns_2)$$

$$s_n = 1/(1 - 2(s_2 - 1)/s_2 + 2(s_2 - 1)/ns_2)$$

SAQ 3

(Ch.1, Exercise 8) You have a choice between buying one uniprocessor that executes five zillion instructions per second, or a ten-processor multiprocessor where each processor executes one zillion instructions per second. Using Amdahls Law, explain how you would decide which to buy for a particular application. Using Amdahls Law, we can define our speedup for ten-processors as follows:

$$Speedup = 1/(1 - p + p/n) = 1/(1 - p + p/10)$$

Setting the speedup equal to our single processor speed (5 ZOPS) will determine the break-even point for multi-processing.

$$1Z/s * Speedup = 5Z/s$$

$$= 1 * 1/(1 - p + p/10) = 5$$

$$\rightarrow 1 > 5 - 5p + p/2$$

$$\rightarrow -4 > p * (-4.5)$$

$$\rightarrow p > 4/4.5$$

$$\rightarrow p > 8/9$$

If our application can have a parallelization ratio of greater than 8/9 (meaning 8/9ths of the application can be distributed to cores in parallel), then it becomes worthwhile to buy the ten-processor unit.

SAQ 4

(Ch.2 Exercise 14) The L-exclusion problem is a variant of the starvation-free mutual exclusion problem. We make two changes: as many as L threads may be in the critical section at the same time, and fewer than L threads might fail (by halting) in the critical section. An implementation must satisfy the following conditions:

- **L-Exclusion: At any time, at most L threads are in the critical section.**
- **L-Starvation-Freedom: As long as fewer than L threads are in the critical section, then some thread that wants to enter the critical section will eventually succeed (even if some threads in the critical section have halted).**

Modify the n-process Bakery mutual exclusion algorithm to turn it into an L-exclusion algorithm. Do not consider atomic operations in your answer. You can provide a pseudo-code solution or written solution.

Below, I've written pseudo-Java that replaces the boolean flag array with an int flag array. Otherwise this is very similar to the Bakery Algorithm provided in the slides.

```
class LExclusion implements Lock {
    int[] flag;
    Label[] label;
    int l;

    public LExclusion (int n, int l) {
        this.l = l
        flag = new boolean[n-l];
        label = new Label[n-l];
        for (int i = 0; i < n-l; i++) {
            flag[i] = 0; label[i] = 0
        }
    }

    public void lock() {
        for (int i = 1; i < n-l; i++) {
            flag[i] = i;
            label[i] = max(label) + 1;
            while ((label[i],i) > (label[k],k))){
                for (int j = 0; j < n; j++) {
                    if (flag[j] > this.l+1)
                        break;
                }
            }
        }
    }
```

```
    public void unlock() {
        flag[i] = 0;
    }

}
```