NWEN 303 Assignment 4

Daniel Braithwaite

October 5, 2016

1 Question 1

The simplest way to achieve this is to have node 5 always send the value of X to node 10 when ever it is changed, and wait for node 10 to read it. Then node 10 will attempt to receive a value of X when ever it needs to use it.

However this causes a large bottle neck and if one of the nodes stops using X then the other is stuck for ever. Perhaps a better option is to run a background process on each of the nodes. On node 5 this background process will be checking for updates to the variable X and when it detects one it will send the variable to node 10. The background process on node 10 will be constantly listening for updates and when it receives one it will write that change. We can implement this using locks and condition variables to avoid any race conditions. First we define the background process doing the sending.

```
1: cond OKU, OKC
2: procedure SENDER
3:     while true do
4:         OKU.await()
5:         send(10, X, "X")
6:         OKC.signal()
```

Here we have two condition variables, OKU (ok update) and OKC (ok continue). When a change is made to X the main process will signal OKU and wait for OKC to be signaled. The sender process will be woken and will attempt a send, this send will block until the receiver gets it. Now we define the receiver process.

```
1: using ← false
2: condOKC
3: procedure RECEIVER
4:     while true do
5:         receive(5, "X'")
6:         aquire()
7:         X ← X'
8:         unlock()
9: procedure LOCK
10:    if usingistrue then
11:        OKC.await()
12:    using ← true
13: procedure UNLOCK
14:    using ← false
15:    OKC.signal()
```

When ever the main process is going to use X it must first use the lock method, and when it is finish it must use the unlock method. Here we attempt to receive the variable from node 5, this will block until there is one to receive, we store this update in a temporary variable. We then use the lock function to get exclusive access to the variable we want to update, now we can update the value of X and then run the unlock function to release our exclusive access.

2 Question 2

For this problem we can use non blocking sends and receives. Here we assume that a non blocking receive will be queued (rather than failing) if there is current message to be received. The solution will work by the coordinator initiating a non blocking receive for each of the rooms. Then when a message arrives at the coordinator we can activate the corresponding receive and a sub routine will flash the corresponding room and finally initiate a new non blocking receive for that room.

```
1: procedure CORDINATOR
2:     for i = 1 to 199 do
3:         receiveA(i, "flash")
```

```
1: procedure SENSOR
2:     while true do
3:         movement_detected()
4:         sendA(0, true, "flash")
```

So now we define our sub routine which is run when an async receive is activated. We assume that the node id number, and the received variable are bound within the scope of the procedure.

```
1: procedure RECEIVE(node_id)
2:     flash(node_id)
3:     receiveA(node_id, "flash")
```

At the end of a receive sub routine, we submit another async receive for the given node.

3 Question 3

The parallel implementation might have a large overhead, which when run on a large N isn't such a problem as it is outweighed by the speedup. However on a single node the overhead of the parallelisation could significantly overshadow the time it takes to actually do the computation. Using another implementation of the program which is serial means its (hopefully) as fast as it can be on a single computer so we are getting a better comparison.

4 Question 4

Averaging a list of numbers is simple, first we sum up the list then divide by the total number of elements. So we can simply divide (scatter) the list of numbers up over the N nodes, the N nodes can sum up the numbers they are given. Then finally the coordinator can gather the individual sums from each of the N nodes, add them all together and compute the average. There is no dependency between any of the N workers, also there is very little communication overhead as the workers don't need to communicate. This is why I would say this problem is embarrassingley parallel.

```
1: procedure MANAGER(S)
2:     for i from 1 to N do
3:         a ← portion i of S
4:         send(node_i, a)
5:     sum ← 0
6:     for i from 1 to N do
7:         receive(node_i, result)
8:         sum ← sum + result
9:     avg ← sum / len(S)
```

Now we will define the worker

```
1: procedure WORKER
2:     recieve(manager, work)
3:     sum ← 0
4:     for n in work do
5:         sum ← sum + n
6:     send(manager, sum)
```