COL 380
Homework 3 Solutions

Author : Aman Bhatia
Dated : 22 April 2016

Problem 1: Bcast

Ans(i) We will provide as an argument the **srcId**, and simply take the XOR of myId with srcId. Also we will take XOR of src, dest with srcId. This way, we will be able to provide any source node. Hence, the algorithm will be,

Algorithm : Bcast Algo Any Source

```
BCAST_any_source(d, srcId, myId, msg) {
    myId := myId ⊕ srcId
    mask := $2^d - 1$
    for i:= d-1 to 0 {
        mask := mask ⊕ $2^i$
        if myId && mask == 0 {
            if myId && $2^i$ == 0 {
                dest := myId ⊕ $2^i$
                dest := dest ⊕ srcId
                send(msg, dest)
            } else
                src := myId ⊕ $2^i$
                src := src ⊕ srcId
                recv(msg, src)
            }
        }
    }
}
```

<u>Ans(ii)</u> We will provide as an argument **n, number of nodes.** Then while sending, we will just check that if destination_id is less than n, then only we will continue. We do not need to check for recieving messages because recieve anyway happens from lower src_id's. This way, we will be able to provide any number of nodes. Hence, the algorithm will be,

Algorithm : Bcast Algo n nodes

```
BCAST_n_nodes(srcId, n, msg) {
  d := ceil(log n)
  mask := $2^d-1$
  for i:= d-1 to 0 {
       mask := mask $\oplus$ $2^i$
       if myId && mask == 0 {
            if myId && $2^i$ == 0 {
                 dest := myId $\oplus$ $2^i$
                 if (dest < n)
                      send(msg, dest)
            } else
                 src := myId $\oplus$ $2^i$
                 recv(msg, src)
            }
        }
    }
}
```

Problem 2: Prefix Scan

(Ans) Below is an implementation of Hypercube Prefix Sum algorithm. It takes as input,
d : the dimension of the hypercube **(d = log p)**
myId : id of the process
data : data that node contains

It returns the prefix sum of that node.

Algorithm : Hypercube Prefix Sum

```
Hypercube_prefix_sum(d, myId, data) {
  msg := data
  res := data
  for i:= 0 to d-1 {
       commId := myId ⊕ 2^i
       send and recieve data to and from commId
       store recieved data in rcvdData
       msg := msg + rcvdData
       if (commId < myId){
            res := res + rcvdData
       }
   }
   return res
}
```

Timing Analysis :-

One exchange of data between two nodes in parallel takes time for 2 additions and non blocking messeage sends. So, it takes $2t_a + (t_s + t_w)$ amount of time for one exchange of data. Now, exchange between nodes happen in parallel and so the total time is no. Of dimensions times time for one exchange i.e.

Total time = $\log p * (2t_a + (t_s + t_w))$, where p is no. of processes.

Problem 3: Butterfly Communication

(Ans) Below is an implementation for calculating sum in a butterfly structured communication. It takes as input,
myId : id of the process
n : total number of process (assumption, n is power of 2)
data : data that process contains

It returns the total sum of data in all the nodes.

Algorithm : Butterfly Communication Sum

```
Butterfly_communication_sum(myId, n, data) {
  mask := 1
  while(mask < n){
       commId := myId ⊕ mask
       send and recieve data to and from commId
       store recieved data in rcvdData
       data := data + rcvdData
       mask := mask << 1
    }
    return data
}
```

Timing Analysis **:-**

One exchange of data between two nodes takes time of 1 addition and a non blocking messeage sends. So, it takes $t_a + (t_s + t_w)$ amount of time for one exchange of data. Now, total time will be no. of iterations times time for one exchange i.e

total time = $\log n * (t_a + (t_s + t_w))$

--------------------END OF FILE--------------------