Parallel Computing
Assignment-6
Sharan Girdhani
800960333

Q1)

a) Chain Network Structure:

	Reduce Star	**Reduce Chain**	**Reduce Tree**
Data Transit Link/Most Loaded	On link i-i+1, where i is [0,P-2], Data transit is O(P-(i+1)) Most Loaded Link: 0-1	All links have equal O(1) data transit. All links are most loaded.	**Wrong Algorithm**
Data Sent/Received Node/Most data	O(1) for Nodes (1,P-1) O(P-1) for Node 0 Hence, Node 0 sends/receives the most data.	O(1) for all the nodes. Hence all nodes communicates most data.	**Wrong Algorithm**
Longest Chain	O(P-1)	O(1)	**Wrong Algorithm**

- **Reduce Chain Algorithm is the best algorithm for this network structure.**

b) Clique Network Structure:

	Reduce Star	**Reduce Chain**	**Reduce Tree**
Data Transit Link/Most Loaded	On any link 0-i, where i is [1,P-1], Data transit is O(1) Rest all the links are not used at all. Most Loaded Link can be all the links associated to Node 0.	On any link i-i+1, where i is [0,P-2], Data transit is O(1) Rest all the links are not used at all. Most Loaded Link can be all the links having O(1) data transit.	**Wrong Algorithm**
Data Sent/Received Node/Most data	O(1) for Nodes (1,P-1) O(P-1) for Node 0 Hence, Node 0 sends/receives the most data.	O(1) for all the nodes. Hence all nodes communicates most data.	**Wrong Algorithm**
Longest Chain	O(1)	O(1)	**Wrong Algorithm**

- **Among the given 3, Reduce Star Algorithm is the best algorithm for this network structure.**

c) Hierarchical Network Structure:

	Reduce Star	**Reduce Chain**	**Reduce Tree**
Data Transit Link/Most Loaded	On any link between level 0 and level 1: Data transit is O(P/2) On any link between level 1 and level 2: Data transit is O(P/4)… and so on.. On the last level and level above that, Data transit will be O(1). Here number of levels will be off the form O(logP) Most Loaded Link will be 0-1 & 0-2.	Let h be the height of a node. For the rightmost link between every consecutive 2 levels: Data transit can be defined as 2*h-1 For every other link Data transit is 2*h. Most Loaded Link will be link 0-1.	**Wrong Algorithm**
Data Sent/Received Node/Most data	O(1) for Nodes (1,P-1) O(P-1) for Node 0 Hence, Node 0 sends/receives the most data.	O(1) for all the nodes. Hence all nodes communicates most data.	**Wrong Algorithm**
Longest Chain	O(logP)	O(2logP)	**Wrong Algorithm**

- **None of the given algorithms are the best for it. The best algorithm can be the one which communicates in a tree like manner i.e. parent-child node direct communication.**

Q2)

a) Round Robin:

I. **Algorithm**:

Let c be the current Node and h^{k-1} be the previous iteration data.

Compute_heat_RoundRobin(N,c,P, h^{k-1})

```
{
    int curr = c;

    while(curr < N)
    {
        if(curr == 0)
        {
            send h^{k-1}[curr] to c+1;
            recv h^{k-1}[curr+1] from c+1;
            h^k[curr] = (2* h^{k-1}[curr] + h^{k-1}[curr+1])/3;
        }
        else if(curr == P-1)
        {
            send h^{k-1}[curr] to c-1;
            recv h^{k-1}[curr-1] from c-1;
            h^k[curr] = (2* h^{k-1}[curr] + h^{k-1}[curr-1])/3;
        }
        else
        {
            if(c == 0)
            {
                send h^{k-1}[curr] to P-1;
                send h^{k-1}[curr] to c+1;
                recv h^{k-1}[curr-1] from P-1;
                recv h^{k-1}[curr+1] from c+1;
            }
            else if(c == P-1)
            {
                send h^{k-1}[curr] to c-1;
                send h^{k-1}[curr] to 0;
                recv h^{k-1}[curr-1] from c-1;
                recv h^{k-1}[curr+1] from 0;
            }
            else
```

```
                        {
                                send h^(k-1)[curr] to c-1;
                                send h^(k-1)[curr] to c+1;
                                recv h^(k-1)[curr-1] from c-1;
                                recv h^(k-1)[curr+1] from c+1;
                        }
                        h^k[curr] = (h^(k-1)[curr-1] +  h^(k-1)[curr] + h^(k-1)[curr+1])/3;
                }
                curr += P;
        }
        return;
}
```

II. Communication per iteration:

For each element, 2 communications are happening except for element 0 & N-1(1 communication).
Hence,
Total Communications: O(2N-2)
Communication per link: O(N/P) in a circular fashion i.e. i-((i+1)%P), where i is [0,P-1]
Communication Per Node: O(2N/P)

b) Block:

I. Algorithm:

Let c be the current Node and h^{k-1} be the previous iteration data.

```
Compute_heat_Block(N,c,P, h^(k-1))
{
        start = c*(N/P);
        end = c+1*(N/P);

        If(c == 0)
        {
                send h^(k-1)[end-1] to c+1;
                recv h^(k-1)[end] from c+1;
        }
        else if(c == P-1)
        {
                send h^(k-1)[start] to c-1;
                recv h^(k-1)[start-1] from c-1;
        }
        else
```

```
        {
                send h^(k-1)[start] to c-1;
                send h^(k-1)[end-1] to c+1;
                recv h^(k-1)[start-1] from c-1;
                recv h^(k-1)[end] from c+1;
        }

        for(i = start; i< end; i++)
        {
                if(i == 0)
                {
                        h^k[i] = (2*h^(k-1)[i] +  h^(k-1)[i+1])/3;
                }
                if(i == N-1)
                {
                        h^k[i] = (2*h^(k-1)[i] +  h^(k-1)[i-1])/3;
                }
                else
                {
                        h^k[i] = (h^(k-1)[i-1] +  h^(k-1)[i] + h^(k-1)[i+1])/3;
                }

        }
        return;
}
```

II. Communication per iteration:

For each node, 2 communications are happening except for node 0 & P-1(1 communication).
Hence,
Total Communications: O(2P-2)

Communication per link: For all the nodes from [1,P-1], there is 2 communication between last element of 1 node and the first element of the next node & vice versa. For node 0 & P-1, 1 communication only. Hence, O(1).
No link communication otherwise.

Communication Per Node: 2 for Nodes [1,P-2], 1 for 0 & P-1. So O(1).

I would use Block Data partition as it contains less communication between nodes.

Q3)

a) **Horizontal:**

- **Algorithm:**

```
Dense_Horizonatal(N, c, P, A, x)
{
        start = c*(N/P);
        end = c+1*(N/P);
        count = 10;
        while(count--)
        {
                // computing y = Ax
                for(i = start; i<end;i++)
                {
                        y[i]=0;
                        for(j = 0;j<N;j++)
                        {
                                y[i] += A[i][j]*x[j];
                        }
                        x[i] = y[i]; // computing x = y
                }
        }
        return;
}
```

- **Memory Required**: O(N*N/P + N + N/P) [A+x+y]
- **No communication required here.**

b) **Vertical:**

- **Algorithm:**

```
Dense_Vertical(N,c,P,A,x)
{
        start = c*(N/P);
        end = c+1*(N/P);
        count = 10;
        while(count--)
        {
                // computing y = Ax
                for(i = 0; i<N;i++)
                {
                        if(c == 0)
                        {
```

```
                        y[i] = 0;
                    }
                    else
                    {
                        recv y[i] from c-1;
                    }

                    for(j = start;j<end;j++)
                    {
                        y[i] += A[i][j]*x[j];
                    }

                    if(c == P-1)
                    {
                        x[i] = y[i]; // computing x = y
                    }
                    else
                    {
                        send y[i] to c+1;
                    }
                }
            }
            return;
        }
```

- **Memory Required**: O(N*N/P + N/P + N) [A+x+y]
- Communications happens in a chain like form here i.e. from link j-j+1 for every i , i=[0,N-1] and j=[0,N-2]
  Total communication: O(N*N-1) or $O(N^2)$
  Communication per link: O(N) for the links mentioned above, 0 otherwise
  Communication per Node:O(N)

c) Block:

- **Algorithm:**

Dense_Block(N,c,P,A,x)

```
{
    startx = (c%sqrt(P))*(N/sqrt(P));
    endx = (c%sqrt(P)+1)*(N/sqrt(P));

    starty = (c/sqrt(P))*(N/sqrt(P));
    endy = (c/sqrt(P)+1)*(N/sqrt(P));
```

```
    count = 10;
    while(count--)
    {
        // computing y = Ax
        for(i = startx; i<endx;i++)
        {
            if(c%sqrt(P) == 0)
            {
                y[i] = 0;
            }
            else
            {
                recv y[i] from c-1;
            }

            for(j = starty;j<endy;j++)
            {
                y[i] += A[i][j]*x[j];
            }
            if(c%sqrt(P) == sqrt(P)-1)
            {
                x[i] = y[i]; // computing x = y
            }
            else
            {
                send y[i] to c+1;
            }
        }
    }
    return;
}
```

- **Memory Required**: O(N/sqrt(P)*N/sqrt(P) + N/sqrt(P) + N/sqrt(P)) = O(N*N/P + 2N/sqrt(P)) [A+x+y]
- **Total communication**: O(N*N-1) or $O(N^2)$
  Communication per link: O(N/sqrt(P)) for links j-j+1, for all i, where i=[0,N-1] and j=[0,N-2], j+1%sqrt(P) != 0, zero otherwise
  Communication per Node: O(N/sqrt(P)), for every Node i, where i%sqrt(P) != sqrt(P) -1
