Parallel Computing
Assignment-6
Nitin Purohit
800956312

Q1)

a) **Chain Network Structure:**

	Reduce Star	**Reduce Chain**	**Reduce Tree**
Most Loaded Link	The link between '0-1' is the busiest link. On link i to i+1, where i is between [0,P-2], Data transit is O(P-(i+1)). So, for link '0-1', all the data from P-1 nodes will flow. O(P-1).	All links have equal O(1) data transit. All links are most loaded.	Most loaded links are links between nodes 2^(LogP-2)-1 and P/2. The amount of data is O(P/2).
Most Loaded Node	Node 0 is the most loaded node. As, all the nodes send data to node 0. Data communications in node 0 is O(P-1).	O(1) for all the nodes. Hence all nodes communicates most data.	Most loaded nodes are nodes between 2^(logP-2)-1 and P/2. The amount of data they handle is LogP.
Longest Chain	The communication going from node P-1 to Node 0 is longest with length O(P-1).	All nodes communicate with adjacent nodes only. So, all communications are of equal length with length O(1).	Length of longest chain of communication is P/2. This occurs between (P/2) number of pair of nodes.

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

b) Clique Network Structure:

	Reduce Star	**Reduce Chain**	**Reduce Tree**
Most Loaded Link	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 any of the link 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 any the link having O(1) data transit.	On any link i-i+1, where i is [0,P-2], Data transit is O(1).
Most Loaded Node	Hence, Node 0 is the most loaded node with O(P-1) data.	O(1) for all the nodes as all the nodes handles same amount of data.	Most loaded nodes are first (logP-1) nodes. The amount of data they handle is O(LogP).
Longest Chain	All communications are of equal length of O(1).	All communications are of equal length of O(1).	All communications are of equal length of O(1).

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

Q2)

a) Round Robin:

I. **Algorithm**:

Let p be the current Process and h^{k-1} be the previous iteration data. N is size of array and P is total number of processes.

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

```
{
    int curr = p;

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

```
                    recv h^(k-1)[curr-1] from p-1;
                    recv h^(k-1)[curr+1] from p+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)

b) Block:

I. Algorithm:

Let p be the current Process and h^{k-1} be the previous iteration data. N is size of array and P is total number of processes.

Compute_heat_Block(N,p,P, h^{k-1})

```
{
        start = p*(N/P);
        end = (p+1)*(N/P);

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

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

Q3)

a) Horizontal:

- **Algorithm:**

Dense_Horizonatal(N, p, P, A, x)

```
{
        start = p*(N/P);
        end = (p+1)*(N/P);
        count = 10;
        while(count>0)
        {
                // 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
            }
            count--;
        }
        return;
}
```

- **Memory Required**: O(N*N/P + N + N/P)   [A+x+y]
- **No communication required here, as there is no exchange of data between rows.**

b) Vertical:

- **Algorithm:**

Dense_Vertical(N,p,P,A,x)

```
{
        start = p*(N/P);
        end = (p+1)*(N/P);
        count = 10;
        while(count>0)
        {
            // computing y = Ax
            for(i = 0; i<N;i++)
            {
                if(p == 0)
                {
                    y[i] = 0;
                }
                else
                {
                    recv y[i] from p-1;
                }

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

```
                if(p == P-1)
                {
                        x[i] = y[i]; // computing x = y
                }
                else
                {
                        send y[i] to p+1;
                }
            }
            count--;
        }
        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,p,P,A,x)
{
        startx = (p%sqrt(P))*(N/sqrt(P));
        endx = (p%sqrt(P)+1)*(N/sqrt(P));

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

        count = 10;
        while(count>0)
        {
            // computing y = Ax
            for(i = startx; i<endx;i++)
            {
                if(p%sqrt(P) == 0)
                {
                        y[i] = 0;
                }
```

```
                else
                {
                        recv y[i] from p-1;
                }

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