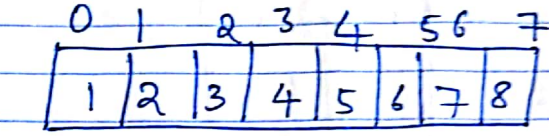
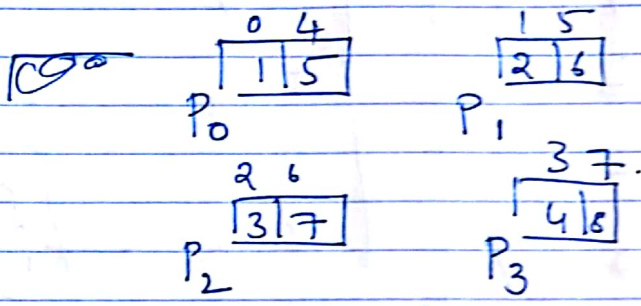
Assignment - 8

(1) Heat-Equation :-

(1.1) Round-Robin Decomposition :-

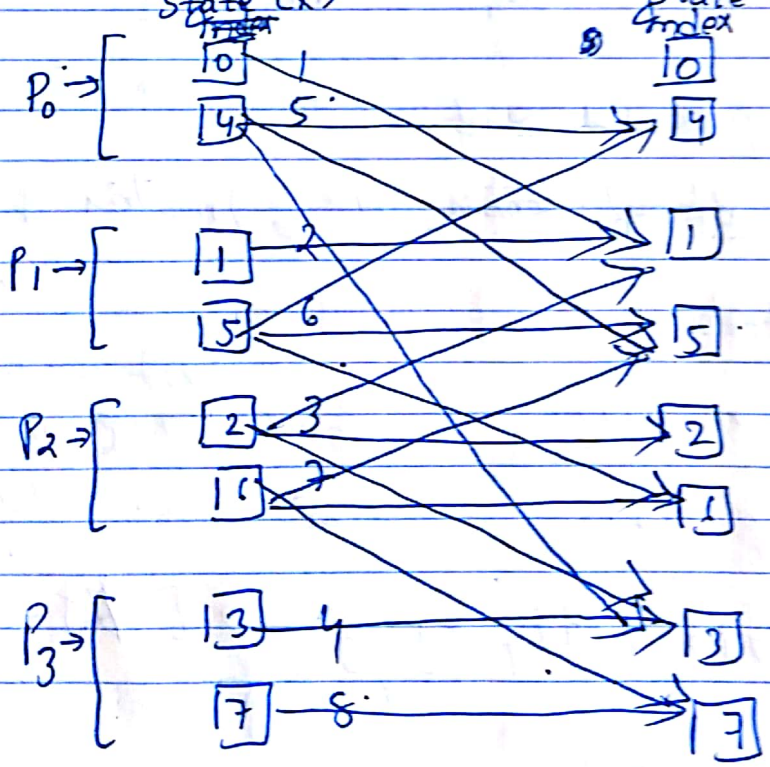


Let say we have P = 4
P (no of Processors)
N = 8



Every index has its global rank & local_rank.

(Assuming the Network is a clique)



∫ I have a gen_function to generate data Locally ∫

// Pseudo mpi-program for Block wise decomposition

```
MPI_Init(&argc,&argv);
int send_value, recv_value;

int my_rank, total_P;
~~int* arr0 = new int[n/p];~~ ~~int* arr1 = new int[n/p];~~
MPI_Comm_rank(my_rank, MPI_COMM_world);
MPI_Comm_Size(total_P, MPI_Comm_world);
// arr0, arr1 have size(n/total_P)
// Every Processor Contains (N/p Size of array)

~~for(int k=0; k<n; k++)~~
int P=0
for(int i=my_rank; i<n; i=i+total_P)
{
   arr0[P]= gen(i);
      P++;
}.                          // arr0 denotes curr state
int send, recv;             // arr1 denotes next state

for(int k=0; k<n/total_P; k++)
{
    if(my_rank==0 && k==0)
    {  send=arr0[k];
       MPI_Send(send, my_rank+1, MPI_comm_world);
       MPI_Recv(recv, my_rank+1, MPI_comm_world);
       arr1[k]= (2 * arr0[k]+recv)/3.
    }
                                   [P-1,0] ->
else
~~{~~ if(~~my_rank= total_P-1~~ && k==(n/total_p)~~-1~~)
  { ~~// last element.~~
```

else {

if (my_rank == (total_P - 1) && k = (n/total_P - 1))

{ //last element

MPI_Send (send, arr[k], my_rank-1)

MPI_Recv (recv, my_rank-1);

Compute. arr1 k

arr1[k] = 2*k arr0[k] + Recv / 3.

}

else.

{ //here you will see. 2 sends & 2 recv for middle process

int send_rank1 = my_rank-1;

if (send_rank < 0).

send_rank1 = total_P-1

int send_rank2 = (my_rank+1) % P

MPI_Send (arr0[k], send_rank1);

MPI_send (arr0[k] , send_rank 2);

int recv1, recv2;

MPI_Recv (recv1, send_rank1)

MPI_Recv (recv2, send_rank 2).

Compute arr1 k

arr1[k] = arr0[k] + recv1 + recv2 / 3

} }

Communication in Round-Robin:-

Except first & last every process perform 2 sends & 2 Recv.

Total Comm^n = first process Comm^n + last process comm^n + rest-all other Proc

$$= \left[\left(\frac{N}{P} - 1\right) * 2 + \frac{N}{P} * 1\right] 2 + \frac{N}{P}$$

$$(total\_P - 2) * \left(\frac{N}{total\_P}\right) * 2 + 2 * \left[\left(\frac{N}{total\_P} - 1\right) * 2 + 1\right]$$

↓ other process (middle process)

↓ first & last process

So, here first & last process also sends 2 sends + 2 Recv except for {(i == 0) in my_rank = 0} & {(i == n) in my_rank = total_P - 1}

Per_link = $\frac{N}{total\_P} * 2$ Every time we are passing 2 sends & 2 Recv through the same Link.

Per_Node :- $\frac{N}{total\_P} * 2$ for all nodes except first & last.

for first $\left[\left(\frac{N}{total\_P} - 1\right) * 2 + 1\right]$ → sends.

for last $\left[\left(\frac{N}{total\_P} - 1\right) * 2 + 1\right]$ → sends.

(1.2) Block-wise decomposition

In Block-wise You need to store $\frac{n}{p}$ elements in the each local-array of the process. & output array to store $\frac{n}{p}$ elements of next state.

```
int my_rank, P;
MPI_Init(&argc, &argv);
int K=0;
for(int i = my_rank * (N/P) ; i<(my_rank+1) * N/P ; i++)
{
  arro[K] = gen(i);      // Block wise decomposition.
      K++;
}int send, recv;
if(my_rank == 0)
{
 for(int i=0; i<N/p; i++)
 {
   if(i==N/p-1)
  {  send=arro[i];
     MPI_Send(arro[i], my_rank+1);
     MPI_Recv(recv, my_rank+1);
     Compute arr1[i] = arro[i]+ arro[i-1] + recv;
                                    3
  }
else {  if(i==0)
        {arr1[i]= 2*arro[i]+arro[i+1];
                    3
        }
} , else{ arr1[i]= arro[i]+arro[i-1]+arro[i+1]
```

our 1st processor here we perform only one Send & one Recv

```
else if (my_rank == P-1).
{
  for(int i=0; i< N/p; i++)
  {
    if (i==0).
    {
      MPI_Send(arro[i], my_rank-1);
      MPI_Recv(arro[i], recv, my_rank-1);
      Compute heat eqn
      arr1[i] = recv + arr0[i] + arro[i+1]
                ---------------------------
                            3
    }
    else {
      if (i == (N/p) - 1)
      {
        arr1[i] = 2*arro[i] + arro[i-1];
                  ------------------
                         3
      }
      else {
        arr1[i] = arr0[i-1] + arr0[i] + arro[i+1]
                  ---------------------------
                              3
      }
    }
  }
}
```

```
else {
    // All the procss in the middle.
    for(int i=0; i<N/p ; i++)
    {
      if (i==0)
      {
        MPI_Send (arro[i], my_rank-1);
        MPI_Recv (recv, my_rank-1);
        arr1[i] = (arro[i]+recv+arro[i+1]) / 3;
      }
      else if (i==(N/p)-1).
      {
        MPI_Send (arro[i], my_rank+1);
        MPI_Recv (recv, my_rank+1);
        arr1[i] = (arro[i]+arro[i-1]+recv) / 3;
      }
      else {
        arr1[i] = (arro[i]+arro[i-1]+arro[i+1]) / 3
      }
    }
}
```

Communication in Block-wise decomposition:-

Total-comm = first_proc + middle_proc + last_proc.

= 1 + (P-2) * 2 + 1 (Per iteration)

= 2 + (P-2) * 2. = 2 + 2P - 4 = 2P - 2.

Per link = 2. {on an Average} per iteration.

Per Node = Every node on average performs 2 sends & 2 recv.'s whereas 1st & last proc make 1 Send & 1 Recv

(2A) 1-D. partitioning (horizontal):-

(a) In horizontal partitioning all the matrix is going to be cut $\frac{N}{P}$ pieces horizontally

If you have a $N \times N$ → matrix we cut into

P pieces of $\left[\frac{N}{P} \times N\right]$ * and vector of size $\frac{N}{P}$ of P-Pe

$N \times N$ → Sized matrix
N → " " vector
P → Total no of Procss.

```
for(int i=0; i< N/P; i++)
{
  for(int j=0; j<N; j++)
  {
    a[i][j] = Gen((my_rank)* N/P + i, j)
  }
}
// Curr_state vector
for(int i=0; i< N/P; i++)
  { curr_state[i] = Gen(my_rank * N/P + i);
  }
```

// So Every Processor has $\frac{N}{P}$ part of Vector and it requires $(P-1) * \frac{N}{P}$ data from other processors to Compute the matrix-vector multiplication..

// So, the current process should. its $(\frac{N}{P})$ Part of Vector to all other process. (Broad-cast).

and Recv. the $(P-1)*(\frac{N}{P})^{th}$ Part of the data from all other Process.

Compute the matrix-Vector multiplication locally and save. the result.

Copy the result in to required $(\frac{n}{p})^{th}$ curr_state Vector &.

Broad Cast $(\frac{N}{P})^{th}$ vector to all other in next iteration

Continue for 10 iterations.

Total-Comm^n :- (Horizontal)

(2) Memory each node Consist are :-

$[\frac{N}{P} \times N]$ → To store matrix.

$[\frac{N}{P}]$ → Size Array to store a Vector

$(P-1)*(\frac{N}{P})$ → Size Array to store. Recv-values

(2) Total-comm^n :-

Every proc perform / Send & $(\frac{N}{P})^{th}$ data & $(P-1)*\frac{N}{P}$ recv-data. So, $\left[\underbrace{\frac{N}{P} \times (P-1)}_{\text{Sends}} + \underbrace{\frac{N}{P} \times (P-1) \times P}_{\text{Recv}}\right]$ ← Total-comm^n

Horizontal-Cut :-

Total Communication per Link = 1 Send of $\frac{N}{P}$ + 1 Recv of $\frac{N}{P}$

{ Since N/w is clique }

Communication per Node = $(P-1)$ Sends of $\frac{N}{P}$ + $(P-1)$ Recv of $\frac{N}{P}$.

(2.2) Vertical - Decomposition :-

In vertical decomposition we have $N \times N$ matrix Cut to $\frac{N}{P}$ pieces Vertically.

// So, Total matrix of $N \times N$ is chopped & locally Generated in to Each process of size $\left(N \times \frac{N}{P}\right)$ and matrix of size $\left(\frac{N}{P}\right)$ and output array of size "N" to store the output.

// once the data is Generated Locally we perform the matrix vector b/w $\left(N \times \frac{N}{P}\right) \times \left(\frac{N}{P} \times 1\right)$ Bro, the results are stored as Colum vector Or vector that Contains $\left(\frac{N}{P}\right)^{th}$ Part of original answer. So, to get right Answer we need to Perform All reduce. So, that Every process has the right Vector of 1st iteration.

All reduce = { All Broadcast + Reduce on Each process
(P-1) Sends + (P-1) Recv.

So, Now we take the required (my_rank)** Part to Curr - state & perform next iteration.

Continue until 10 ; iteration

$N \times N = 3 \times 3 \quad P = 3.$

Eg :- $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 1 \\ 4 \\ 7 \end{bmatrix} [1] \quad \begin{bmatrix} 2 \\ 5 \\ 8 \end{bmatrix} [2]$

$\begin{bmatrix} 3 \\ 6 \\ 9 \end{bmatrix} [3]$

$P_1 \quad P_2 \quad P_3.$

$\Rightarrow \begin{bmatrix} 1 \\ 4 \\ 7 \end{bmatrix} \begin{bmatrix} 4 \\ 10 \\ 16 \end{bmatrix} \begin{bmatrix} 9 \\ 18 \\ 27 \end{bmatrix}$

After all-reduce Every Procs Contains $\begin{bmatrix} 14 \\ 32 \\ 50 \end{bmatrix}$

Total - Commⁿ :-

Memory :- $\left(\frac{N \times N}{P}\right)$ → matrix.

$\left(\frac{N}{P}\right)$ → size Vector to Store curr_state

N-sized Vector → Next state result of $\left(\frac{N \times N}{P}\right) + \left(\frac{N}{P}\right)$

$O\left(\frac{N^2}{P} + \frac{N}{P} + N\right) = O\left(\frac{N^2}{P}\right)$

Communication : $P \times ((P-1) \text{ sends of } \frac{N}{P} + (P-1) \text{ recv of } \frac{N}{P})$

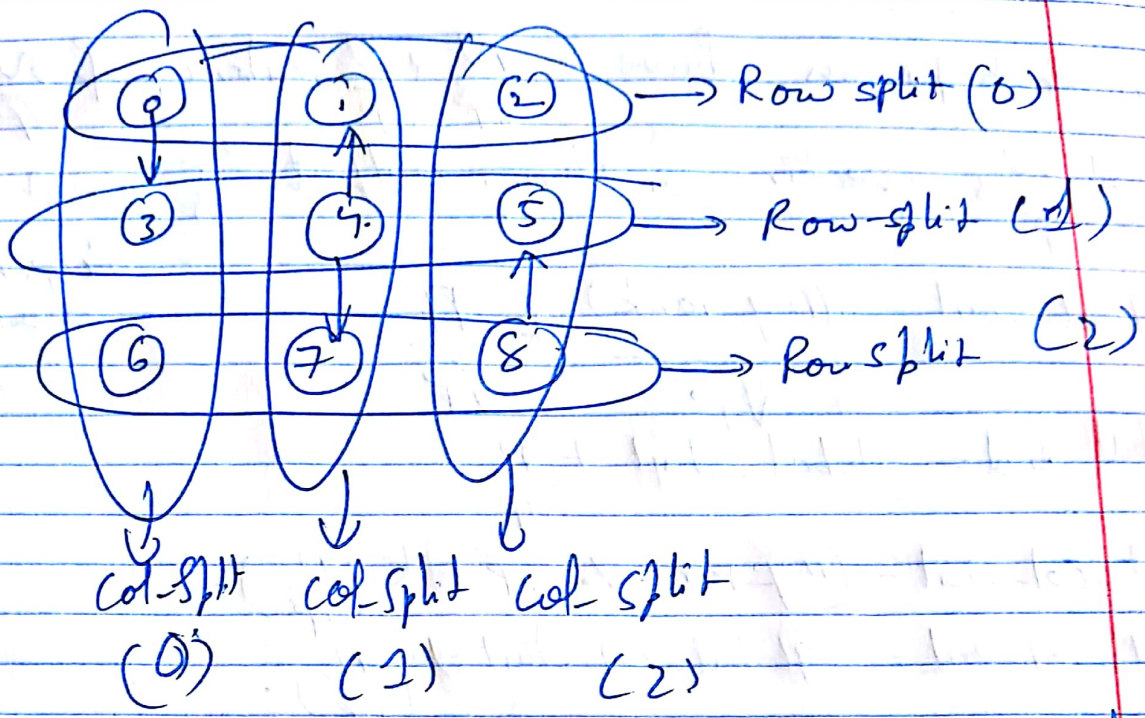
$= P \times \left((P-1)\frac{N}{P} + (P-1)\frac{N}{P}\right) = 2 * (P-1) * P * \frac{N}{P}$

$= 2 * (P-1) * P * \frac{N}{P}.$

Per link = 1 Send $\frac{N}{P}$ & 1 Recv $\frac{N}{P}$.

Per Node = $(P-1)$ send $\frac{N}{P}$ & $(P-1)$ Recv of $\frac{N}{P}$.

(2.3) In Block wise decomposition.



(1) In Block-wise decomposition we store NXN matrix in different processor with a matrix of size $\frac{N}{\sqrt{p}} \times \frac{N}{\sqrt{p}}$ and a vector of size $\frac{N}{\sqrt{p}}$ & output vector of size $\frac{N}{\sqrt{p}}$.

(2) we compute Local matrix Vector multiplication of $\frac{N}{\sqrt{p}}$.

// Essentially we compute global row_start & end, global start & col_end.

global_row_start = (my_rank) / Processor in each colm

Block-decomposition:-

In Each process have a $\frac{N}{\sqrt{P}} \times \frac{N}{\sqrt{P}}$ matrix, & $\frac{N}{\sqrt{P}}$ size vector and $\frac{N}{\sqrt{P}}$ size vector for storing output. { genA is a function generate data }

global_start = $\frac{(my\_rank)}{\sqrt{P}} * \frac{n}{\sqrt{P}}$.

global_end = global_start + $\frac{n}{\sqrt{P}}$.

global_col_start = $(my\_rank \% \sqrt{P}) * \left(\frac{n}{\sqrt{P}}\right)$.

global_col_end = global_col_start + $\left(\frac{n}{\sqrt{P}}\right)$.

// Based on global_

for(int i = global_start ; i < globalend ; i++)

{ for(int j = global_col_start; j < global_col_end ; j++)

{ grid[k] = genA(i, j);

} }

// by generate a vector.

Perform a row wise split based on $(my\_rank / \sqrt{P})$ and Col_wise split based on $(my\_rank \% \sqrt{P})$

so, perform All reduce in row_split_wise

so, for next iteration we do Column wise BroadCast. from the diagonal PP assors for next iteration continue for 10 iterations

~~Total comm~~:

Memory:- $\left(\frac{N}{\sqrt{p}} \times \frac{N}{\sqrt{p}}\right)$ matrix & $\frac{N}{\sqrt{p}}$ → vector for i/p & o/p

$$= O\left(\frac{N^2}{p} + \frac{N}{\sqrt{p}} + \frac{N}{\sqrt{p}}\right) = O\left(\frac{N^2}{p}\right).$$

Total comm^n = row_wise + col_wise.

If row_wise comm^n performs {all_reduce}

$$\text{row-wise} = \underbrace{\left[\underbrace{\left(\frac{N}{\sqrt{p}} \times (\sqrt{p}-1)\right)}_{\text{sends}} * \sqrt{p} + \underbrace{\left(\frac{N}{\sqrt{p}} \times (\sqrt{p}-1)\right)}_{\text{recv}} * \sqrt{p}\right]}_{\downarrow \text{1 rows}} \times \underbrace{\sqrt{p}}_{\downarrow \sqrt{p}\ \text{row}}$$

Col-wise:

$$\text{1 Column take} = \left(\underbrace{\frac{N}{\sqrt{p}} \times (\sqrt{p}-1)}_{\text{sends}} + \underbrace{\frac{N}{\sqrt{p}} * (\sqrt{p}-1)}_{\text{Recv}}\right) \times \sqrt{p}$$

$$\therefore \text{Total comm}^n = \text{row\_wise} + \text{col\_wise}.$$

$$= 2N * \sqrt{p}(\sqrt{p}-1) + 2N * (\sqrt{p}-1)$$

$$= 2N(\sqrt{p}-1)\cdot(\sqrt{p}+1) = 2N * (p-1)$$

$$= .$$

On-link:- 1 send $\frac{N}{\sqrt{p}}$ & 1 Recv $\frac{N}{\sqrt{p}}$.

most-loaded node:- The Central processor in above example at (4) in 3×3 grid of proc

Performs $(\sqrt{p}-1)$ recv's of $\frac{N}{\sqrt{p}}$ data

& $(\sqrt{p}-1)$ Sends of $\frac{N}{\sqrt{p}}$ data

If we perform only diagonal reduce in processors instead of all reduce we can avoid lot of communication.

1 row can have $\left(\frac{N}{\sqrt{p}} \times (\sqrt{p}-1) \text{ recv.} + \frac{N}{\sqrt{p}} \text{ send.}\right)$

Total row comm^n $= \left(\frac{N}{\sqrt{p}} \times (\sqrt{p}-1) + \frac{N}{\sqrt{p}}\right)\sqrt{p}$.

Colum $= \left(\frac{N}{\sqrt{p}} \times (\sqrt{p}-1) \text{ sends} + \frac{N}{\sqrt{p}} * (\sqrt{p}-1) \text{ recv}\right) * \sqrt{p}$

$= \frac{2N}{\sqrt{p}} * (\sqrt{p}-1) * \sqrt{p} + N(\sqrt{p}-1) + N.$

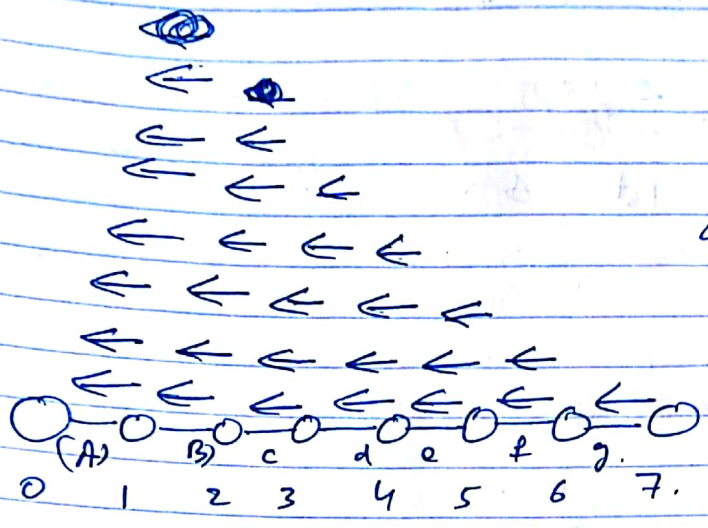
$O(N\sqrt{p}) = 2N(\sqrt{p}-1) + N(\sqrt{p}-1) + N$

$= N(2\sqrt{p}-2+\sqrt{p}-1+1) = N(3\sqrt{p}-2)$

(3) Reduction:-

Reduce-Star on-chain :-

In Reduce star all the results are send to zero.
Let say $p = 8$.



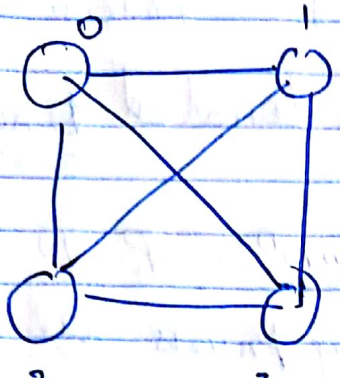
In the figure beside.
The most loaded link is (A) with 7 messages.
In terms of P.
$(P-1) * 1$.

The most loaded node.
'1' it is perform.
=
6 sends & 6 Recv
"P-2" sends & "P-2" Recv.
$\Rightarrow 2P-4$ Communication

The longest chain of Comm^n happen from Processor 0 to proc 7
The " " " " is $(P-1)$ in above Ex: 7

Total Comm^n $= 1 + 2 + 3 + \cdots + P-1 = \frac{P(P-1)}{2} = O(p^2)$

Reduce - star on clique :-



Since the underlying N/w is a clique. Every ~~link~~ node has a link to Every other Processor.

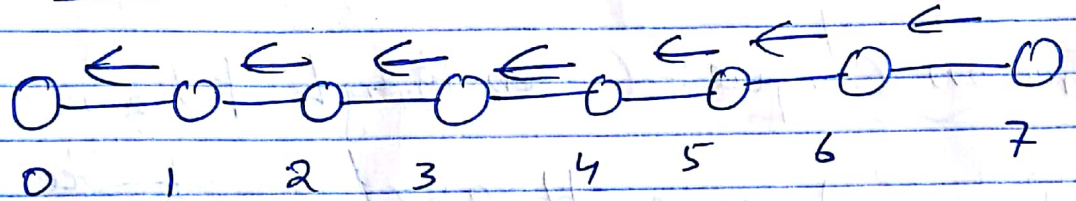
Total Comm^n = P − 1.

bcz. Every link has one - Value to send.

Most loaded node is zero. because. It is receiving (P−1) recv from all other Procs

Comm^n is O(P) Longest Comm^n is 1

Reduce - chain :- {N/w is chain}



Every node performs Exactly one Send & one Recv. Except the last & first nodes. (because the last node did one send) first node recv one - value).

Total Comm^n = P − 1. = O(P).

longest Chain of Comm^n = P − 1 i.e (0 − 7)

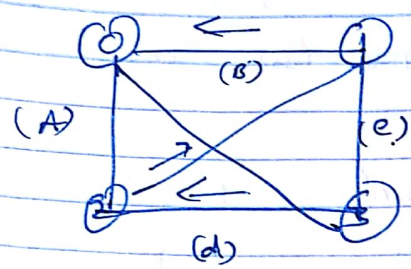
Just look at the arrows.

data on the most loaded link = 1 data throughout the N/w on each Link.

data on most loaded Node = [0 − 6] {1 − P − 2} rank

Processor perform 1 Send & 1 Recv.

Reduce - chain. (clique).



here 1 & 2 Procn are the most loaded Ones

Total comm^n = $(P-1) = O(P)$

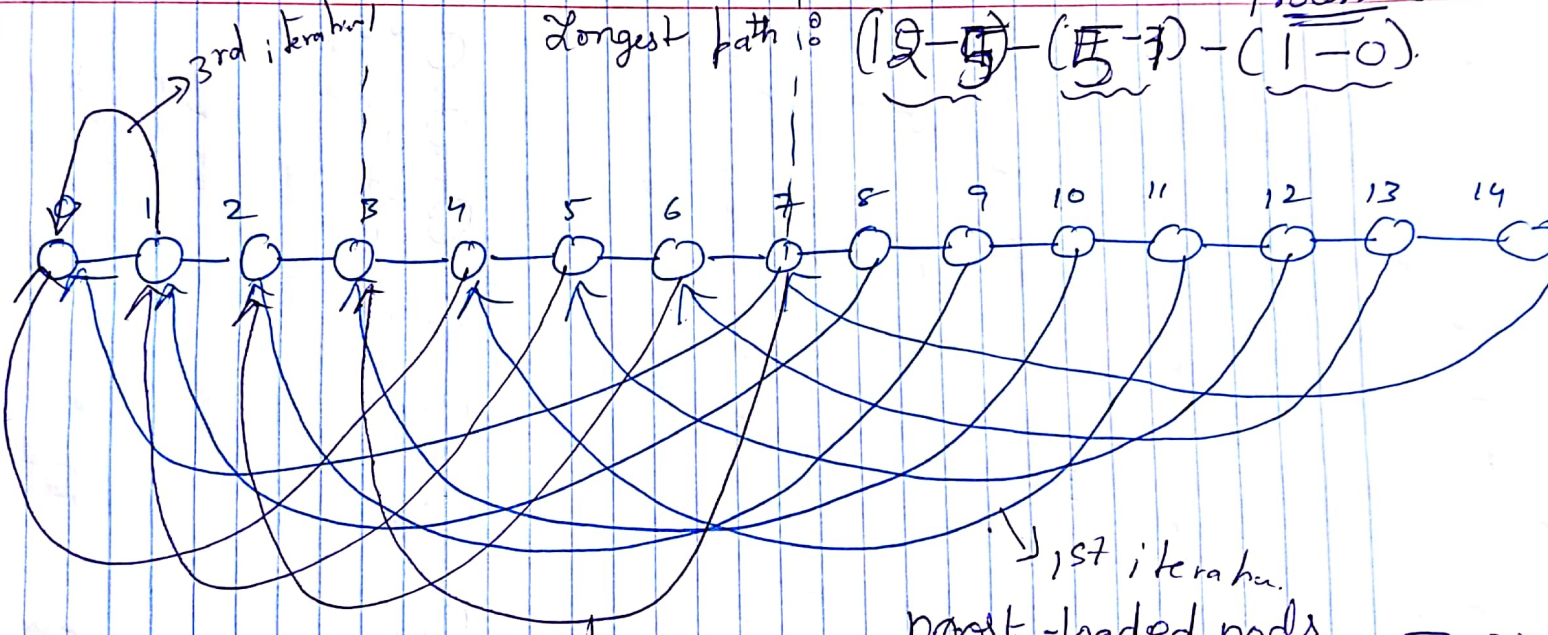
Because every process is sending & its data to its previous processes.

longest comm^n = $P-1 = O(P)$.

most loaded link. : 1 Send.

most loaded nodes :- {1 - - - P-2} ranks performs 1 Send & 1 Recv.

Reduce Tree on clique:- (Even though the N/w is clique) 4ts hard to express on 16 processors



most-loaded nods $[0 - \frac{p}{2^{n-1}}]$ → ra

Every procn has 4ts link to other processors.

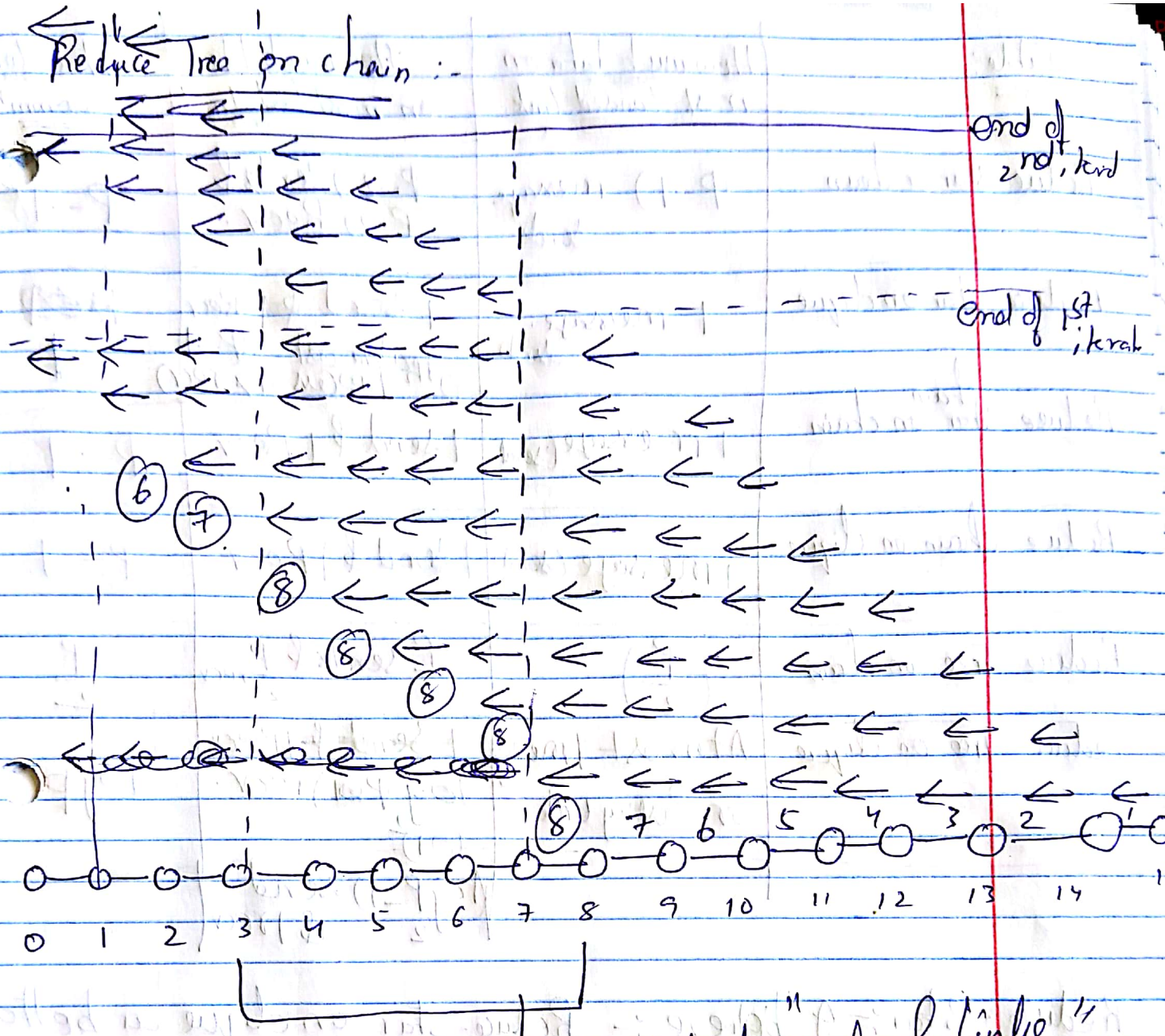
data on most loaded link is 1 send (one message)

data on most loaded node is [0 - 4] (4) rank processor & → ranks

where n = iterations they perform (2 recv & one send)

Critical path = O(log p) = here 4 is 3 = (12-5)-(5-1) (1-0)

Reduce Tree on chain :-



most loaded "node of Linke"

Whose rank are b/w $\left[\frac{P}{4} - \frac{P}{2}\right]$ in above Example.

longest chain of comm^n is $\left(\frac{P}{2}\right)$.

data on most loaded node is $\left(\frac{P}{2}\right)$.

Table:-	How much data on most loaded link.	How much data on most loaded node	How long is Comm^n
Reduce-star on chain	$(P-1)$ message sends	$(P-2)$ sends & $(P-2)$ Recv.	$(P-1)$
Reduce-star on clique.	1 message send	1 send & 1 Recv. (P-1 processors recv)	1
Reduce-chain on chain.	1 message (send)	1 send & 1 Recv.	$P-1$
Reduce chain on clique.	1 message (send)	1 send & 1 Recv.	$P-1$.
Reduce Tree on chain	$\left(\frac{P}{2}\right)$	$\frac{P}{2}$ sends & $\frac{P}{2}$ recv	$\frac{P}{2}$
Reduce Tree on clique.	Atmost 1 message on Every link.	$(\log_2 P)$ recv. (or) $(\log_2 P-1)$ send & 1 recv	$\log P$

Reduce-Star:- I believe :- Reduce-star on clique is better because Comm^n is less.

Reduce-chain I believe : Reduce-chain on chain is better because clique is very large N/w in real. Time is complex to model & handle.

Reduce-Tree: I believe :- Reduce-Tree on clique is better because The Comm^n is very less Compare to Chain.