Lec 9

"Variability"

- MPI_Isend (buf, count, datatype, dest, tag, comm, req)
- MPI_Irecv (buf, count, datatype, src, tag, comm, req)
- MPI_Wait (req, status)
- MPI_Waitall (count, req, status)
- MPI_Test : non blocking, tests completion, starts progress.

22:40

* MPIR_CVAR_ASYNC_PROGRESS = 1
  → for same isend / irecv code, time decreases

→ Comm Graph / Matrix

Symmetric matrix $A[i][j] = A[j][i]$

= Total no. of bytes comm b/w ranks i and j irresp of "dir" of comm.

→ Process Mapping / Alloc"

Given network topology and a given comm pattern (virtual topology)

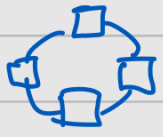
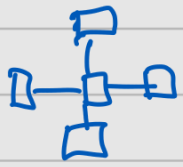
process mapping affects comm. time.

Lec 10

→ ## Attributes of Interconnects

- Topology (connectivity) — structure of network
- Cost — no. of wires
- Diameter — max possible dist
- Bisection width — mincut

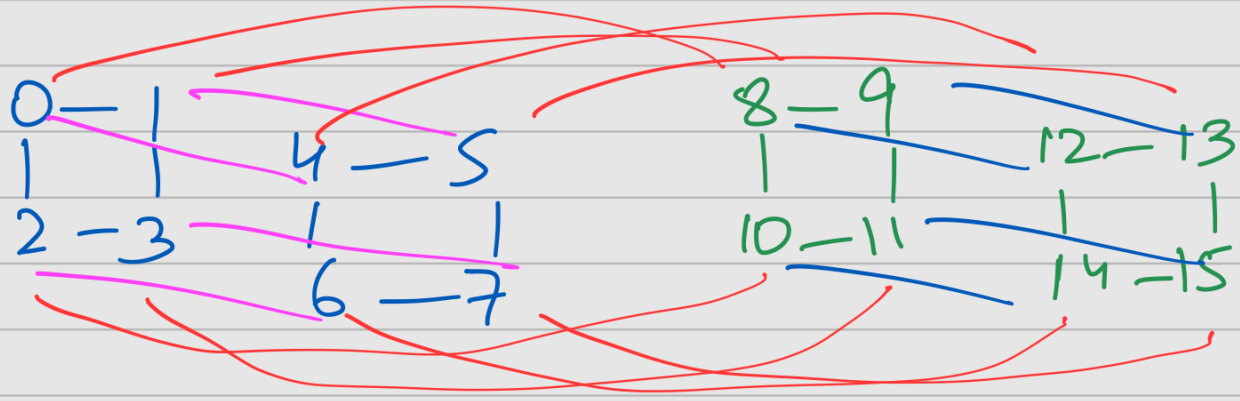
(# nodes = p)

	Topology	Dia	Cost	Bis Wid
	Linear	p-1	p-1	1
	Ring	floor(p/2)	p	2
	Star	2	p-1	—
	(single pt of failure, cong. at central node)			
	2D Mesh	$2(\sqrt{p}-1)$	$2\sqrt{p}(\sqrt{p}-1)$	$\sqrt{p}$
	2D Torus	$\sqrt{p}$ (even) $\sqrt{p}-1$ (odd)	$2p$	$2\sqrt{p}$

3D Mesh $3(p^{1/3}-1)$ $3(p^{1/3})(p^{1/3}-1)$ $p^{2/3}$

3D Torus

Hypercube $\log p$ $\frac{p \log p}{2}$ $p/2$



Tree Networks

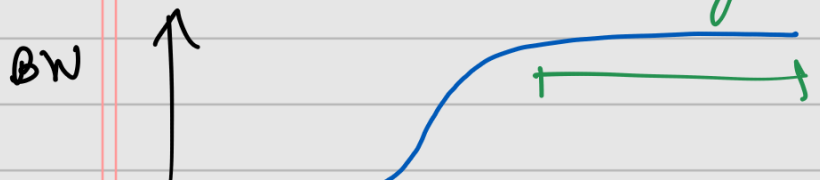
Prob: Comm bottleneck at higher levels

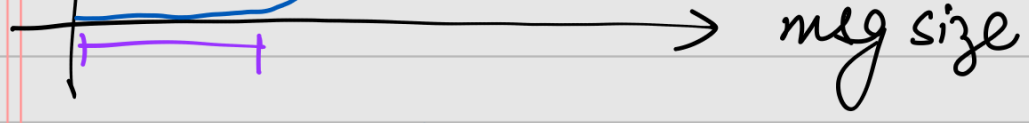
↳ Fat tree Networks

→ ## Comm. Cost

Bandwidth / Throughput = $\frac{\#\ bits}{time}$ bps

* Latency bound region: Small msg size sat^n
* Bandwidth bound region: large msg size sat^n





— for n words on l links

Cut thru $\boxed{t_s + t_a(l) + \frac{n}{B}}$

t_s: startup time (at send/recv nodes)

$t_a(l)$: Node latency (per hop time)

B: bandwidth or t_w: per word transfer time

Store & Fwd $\boxed{t_s + t_a(l) + \frac{ln}{B}}$

Hockney Model / Postal Model

$$T_{comm} = L + \frac{n}{B} \quad \left(L = t_s + l t_a\right)$$

affect effective bandwidth →

1) Bulk comm
2) fewer hops
3) less volume comm
4) network congestion
5) network topology

Realistic model ⇒ overhead, contention

$$L' + \frac{n}{B'}$$

→ MPI Groups

- ordered set of processes.
- contiguous ranks
- MPI_Comm_group (comm, *group)

- MPI_Group_rank, MPI_Group_size
- Union & intersection of groups

→ MPI Communicators

- MPI object cont. a group of processes
- MPI_COMM_WORLD
- represents a comm. domain
- memory prop. to # processes in the group
- several communi^n contexts may co-exist within a single communicator.

MPI_Comm_Split

inp

MPI_Comm_split (oldcomm, color, key, * newcomm)

output

* collective call — everyone in oldcomm has to do this.

* color — division criteria

* color — division criteria
(same col ⇒ same group)

* key — new rank assignment in inc order of key.

- MPI_Group_incl(g_group, N, ranks, & new_group)
- MPI_Comm_create_group(MPI_COMM_WORLD, new_group, tag, & new_comm)

– must be called by all processes in the comm,
– tag must be same for all calls.

Lec 12

→ Collective Comm

- must be called by all processes that are part of the communicator

Barrier (sync)

Bcast Reduce Gather Scatter

Allreduce All-gather Alltoall

- MPI_Barrier :

every rank needs to call this func and caller returns only after all processes have entered the call

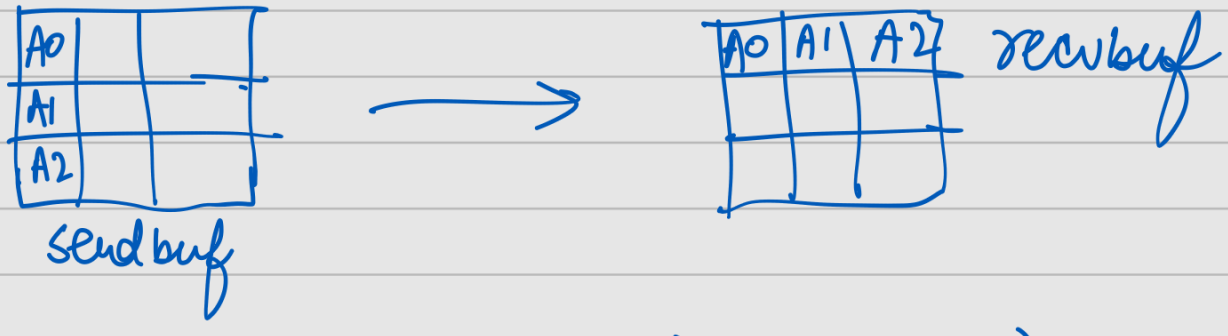
- **MPI_Gather:**

Gathers values from all processes to root.

in comm, all processes ke sendbuf se sendcount elements lake root ke recvbuf me daal dega.

recvcount = size of any single receive

eg P2P — many to one
non-root send → root recv

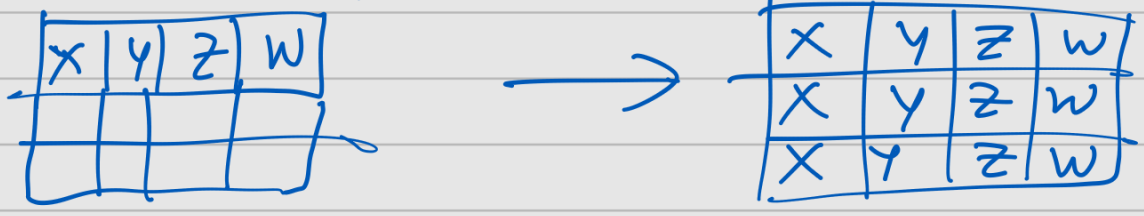


$$T_{naive} = (p-1)(L + n/B)$$

$$T_{opt} = \log p\, L + \frac{n(p-1)}{B}$$

- **MPI_Bcast**

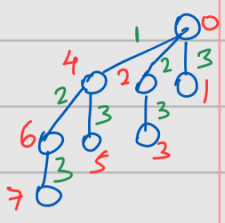
comm ke saare processes to root process buffer ka count elements bhejta, aur unke buffer me rakhega.



eg P2P — One to many. (Naive)

$time = (p-1)\left(L + \frac{n}{B}\right)$ — all 1 hop

optimised : "Binomial Algo" (divide half) (Hypercube)



$Steps = \log_2 p$ $\quad T(p) = \log_2 p \left(L + \frac{n}{B}\right)$

---

col = rank × G/p or rank % G

(P) Comm_Split

Comm_rank (&newr, &newcom)
Comm_size(

int buf[2] p/G − 1

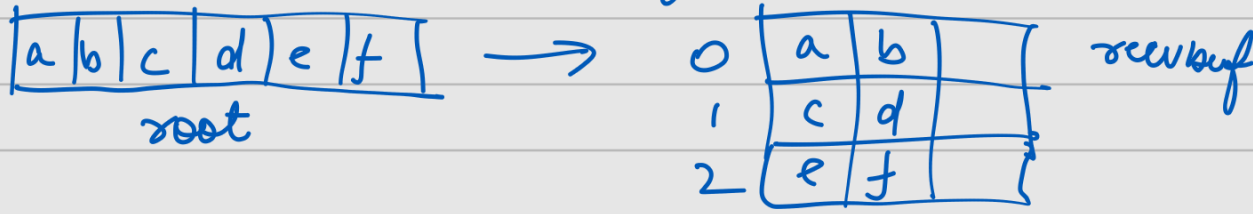
```
if (newr == news-1) {
    buf[0] = rank;
if (newr == 0){
  MPI_Send (          news-1 ,     , newcom)
if (newr = nw-1)
  MPI_Recv (&buf[1], ----0,    newcom)
MPI_Bcast (buf, 2, INT, news-1, newcom)
```

- MPI_Scatter

sendcount — # elements sent to each process

root ke sendbuf se sendcount elements order wise comm ke ranks ke recvbuf me recvcount elements daalega.

Sb



naive — One to many.

$T_{naive} = (p-1)L + \frac{n(p-1)}{p}/B$

$T_{bin} = \log_2 p \, L + \frac{(p-1)}{p}\left(\frac{n}{B}\right)$

$T_{opt} = (\log_2 P)L + \left(\frac{n}{P}\right)\left(\frac{P-1}{B}\right)$

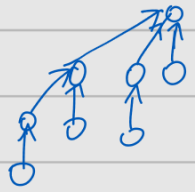
- **MPI_Reduce**

comm ke saare processes ke sbuf se count elements ke operator lgake root ke outbuf me rakh dega.

MIN, Max, Sum, Prod

* using Reduce instead of loop (manually Send_Recv) reduces time for same task.

Naive: many to one + computation



optimised: Recursive doubling → har baar dist doubling

$time = \log_2 P(L + n/B + n \times c)$

$t_{naive} = (P-1)\left(L + \frac{n}{B} + n \times c\right)$

- **MPI_Allgather**

sendbuf

A0		
A1		
A2		

→

A0	A1	A2
A0	A1	A2
A0	A1	A2

recvbuf

(gather + Broadcast)

comm ke saare processes ke sendbuf se sendcount elements pehle ek ke recvbuf me rakhlo, fir us recvbuf ko baat kaa sabke in comm

- **MPI_All reduce**

  (reduce + Bcast)

- **MPI_ All to all**

sb

A0	A1	A2
B0	B1	B2
C0	C1	C2

→

A0	B0	C0
A1	B1	C1
A2	B2	C2

(Transpose)

each rank calls scatter → &[recvbuf + rank]
each rank calls gather
↳ &[sendbuf + rank] se

- **Binomial Algo for Bcast**

P processes (0 → 15) $\log_2 P$ steps

K = 4

```
for (K=p; K>1 ; K=K/2){
    if (rank % K==0)
        send(          , rank+K/2 , rank )
    else if (rank % K==K/2)
        recv(          , rank-K/2, rank-K/2.)
```

→ Domain Decomp

N grid points, P processes, $\frac{N}{P}$ pts per process

	Comm	Comp	Ratio
1D domain 1D decompose	2	N/P	$\frac{2P}{N}$
2D domain 1D decompose	$2\sqrt{N}$	N/P	$\frac{2P}{\sqrt{N}}$
2D domain 2D decompose	$\frac{2\sqrt{N}}{\sqrt{P}}$	$\frac{N}{P}$	$2\frac{\sqrt{P}}{\sqrt{N}}$
5pt stencil	$\frac{4\sqrt{N}}{\sqrt{P}}$	$\frac{N}{P}$	$4\frac{\sqrt{P}}{\sqrt{N}}$

MPI_Pack (buf)
MPI_Send (buf)

MPI_Recv (buf)
MPI_Unpack (buf)

(just one) "Gather the count first"

integer)

for Gatherv variable size.