→ Lec 2

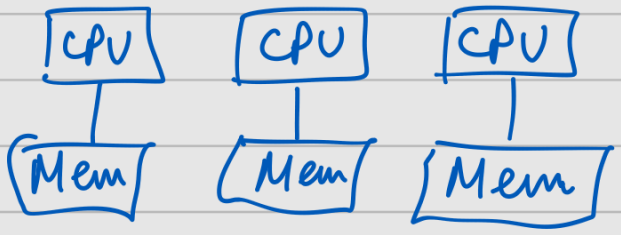
- Speedup = $\frac{\text{Time taken on 1 processor}}{\text{Time taken on P processors}}$

- Efficiency = $\frac{\text{Speedup}}{P}$

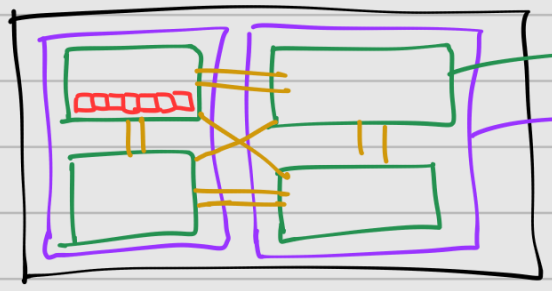
→ Amdahl's Law

Speedup $S = \frac{1}{(1-f) + f/P}$ (f = frac of code ||able)

→ NUMA



(Non uniform mem access)

→ 

→ Numa Nodes (cont cores)
→ Sockets

One Compute Node.

→ MPI

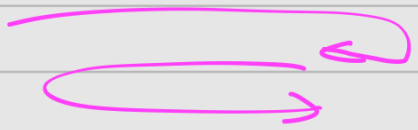
Standard for message passing.
Distinct / Distributed memory environment
Explicit communication

→ Process – instance of prog execution

→ Execution context of a running prog – resources associated with a prog's exec^n

→ PPN = process per node.

→ Round Robin



hostfile does not mention # processes on any node. 0 1 2 3
4 5 6 7

→ Sequential / Contiguous

hostfile mentions how many processes on any node. 0 2 4 6
1 3 5 7

- MPI_Comm_Size (MPI_COMM_WORLD, &size)
- MPI_Comm_Rank (MPI_COMM_WORLD, &rank)
  global communicator

→ give values wrt communicator.

→ Process
belongs to a group.
identified by rank within the group.

→ Communicator
defines scope, specifies comm. context

→ MPI_Get_processor_name (hostname, &len)
: host name (ceus 3)

→ core Id = sched_getcpu()
: which core on that host that process.

→ Comm < P-2-P

Collective

→ Data types: BYTE, CHAR, INT, FLOAT, DOUBLE

→ MPI Send:
(buf, comt, datatype, dstn, tag, comm)

→ MPI Send:
(buf, comt, datatype, src, tag, comm, status)

size (B) = count x sizeof (datatype)

→ Errors in Send – Recv

1) Recv → ctr Send → cts
ctr < cts : runtime error

2) tag mismatch: no compile-time or run-time error. (Blocked)

- strlen ( ) : size except \0.
Just string size.

3) MPI Send: blocking, standard mode.
- does not return till send buffer can be reused. message buffering affects this.

(can transfer it to MPI buffer & move on)

4) Eager v/s Rendezvous

Eager: buffering enabled.
- Completes without ack from dstn.
- small msgs, typically 128 KB.

Rendezvous:

- needs ack from matching receive.
- large msgs.

→ MPI States

MPI_SOURCE: source rank
MPI_TAG: message tag
count → MPI_Get_count: msg len

* we take max of send & recv times.
* intra faster than inter
* distant node ⇒ time inc

→ Sources of variation

- network congestion / background traffic
- OS noise
- other appl.

* MPI_Reduce
applies a func to all values in arg0 of all ranks & stores result in rank x variable arg1.

* Need for diff recv [numranks][datasize]

only needed when many-to-one. Not needed for one-to-many.

* MPI_Send: blocking send → does not return till the msg data and envelope have been safely stored so that sender is
MPI_Recv: blocking recv.

* MPI_ANY_SOURCE : recv specify wildcard for src
MPI_ANY_TAG : recv wildcard for tag.
→ only for recv

→ Many to One

recvarr [size][20]; buf [20];

MPI_Recv (buf, 20, ____ , MPI_ANY_SOURCE, MPI_ANY_TAG, ____)

(copy) recvarr [status.MPI_SOURCE] = buf;

(we get status obj after Recv is done).

→ Over decomposition :

processes ↑ but time rem same.

→ Timing factors

(Same # processes, data size ↑ ⇒ computation time ↑, comm time ↑)
Same data size, # processes ↑ ⇒ comp time ↓ comm time ↑

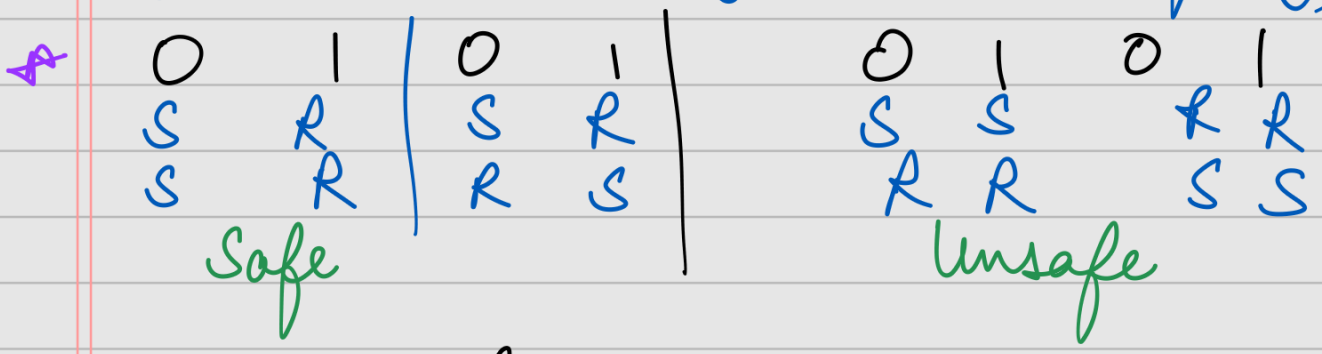
i) Data size ↑ ⇒ comm time inc
ii) Compute time scales almost.
iii) No. of processes ↑ ⇒ comm. time inc

* Send (buf, count, INT, ____)
Recv (buffer, count, CHAR, ____)

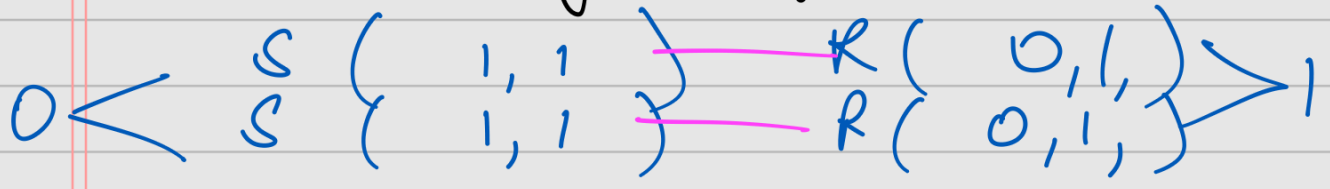
Runtime error : sent count x 4 bytes
recv space allowed = count bytes

if msg size is smaller than recv count, then only those locations corresponding to message are modified

★ MPI_Send: non blocking till one point, after that blocking. ✓ (no buffering)



→ Non-overtaking Messages



if src, tag match ⇒ first send matches first recv only.

→ Diff Sends

① Send: blocking. Standard

- started whether or not matching recv posted
- may complete before matching recv posted
- MPI may buffer messages.

② Bsend: buffered

- may complete before matching recv posted
- does not depend on occurence of matchg recv

③ **Ssend** Synchronous

- Completes only if a matching recv posted.

④ **Rsend** Ready

- Started only if a matching recv posted.

- MPI_BUFFER_ATTACH (buffer_addr, size)
  - buffer in user's memory
  - used by msgs sent in buffered mode
  - only one buffer can be attached to a process at a time.

MPI_BUFFER_DETACH (buffer_addr, size)

Size = sum of sizes of all outstanding Bsends + MPI_BSEND_OVERHEAD for each Bsend.

*

0	1	0	1	0	1
Ss	R	Ss	R	Ss	Ss
Ss	R	R	Ss	R	R
safe				Unsafe/Deadlock	

→ **2D NN with odd even**

```
if (r%2==0 and r<P-1)
    S(____, r+1, r+1, ____)
    R(____, r+1, r, ____)
else if (r%2==1 and r>0)
    R(____, r-1, r, ____)
```

S (r-1, r+1,)

if (r%2==0 and r>0)
S(r-1, r-1,)
R(r-1, r,)

else if (r%2==1 and r<P-1)
R(r+1, r,)
S(r+1, r+1,)

- NN-2 takes almost half time than NN-1 (both 2D). if scaled. when internode. Intranode → both same time.
- NN time more determined by no. of internode comm happening.

→ Isend / Irecv : Non Blocking P-to-P

MPI_Isend (buf, count, datatype, dest, tag, comm, request)

MPI_Irecv (buf, count, datatype, src, tag, comm, request)

MPI_Wait (request, status)

MPI_Waitall (count, request, status)

# of outstanding Irecv + Isend called by that process

* Intra node V/s Inter node — no effect on computation time. Only comm time ↑

* Small messages are latency bound.
  large messages bandwidth ~ bound.

* Ring algo - data size remains same for each successive communication
  Recursive Doubling — data size comm doubles.

  also # hops increases in rec doubling while in ring we always comm. with neighbour