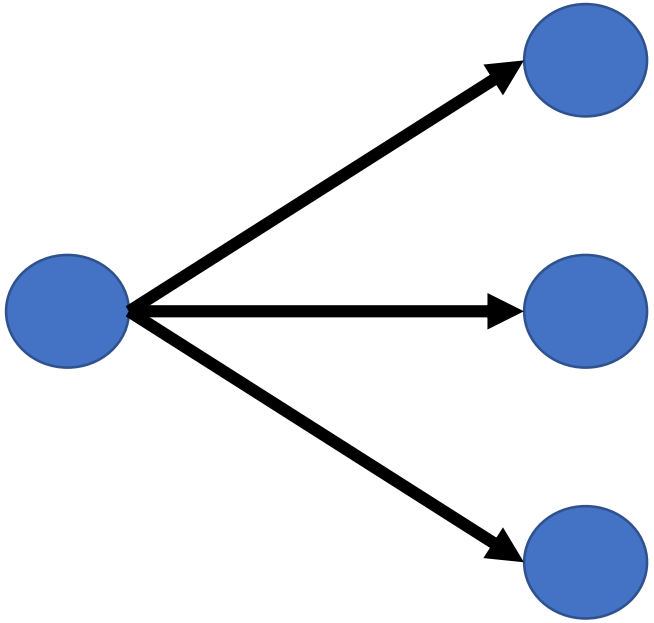
Collective Communication

Lecture 10

February 7, 2024

P2P and Collective



Collective Communication

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

Types

- Synchronization (MPI_Barrier)
- Global communication (MPI_Bcast, MPI_Gather, …)
- Global reduction (MPI_Reduce, ..)

Barrier

- MPI_Barrier (comm)
- Every rank needs to call this function (for true synchronization)
- Caller returns only after all processes have entered the call

```
printf (“Before barrier”);
MPI_Barrier (MPI_COMM_WORLD);
printf (“After barrier”);
```

Barrier

n=4

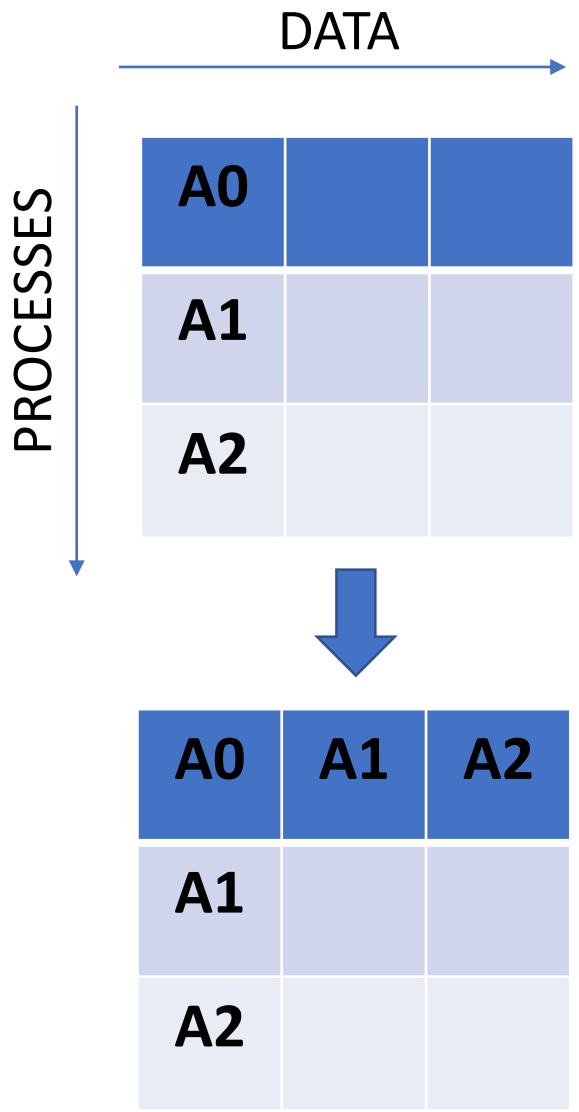
```
if (myrank != 0)
    MPI_Barrier (MPI_COMM_WORLD);
printf("%d\n", rank);
```

Gather

- Gathers values from all processes to a root process
- int MPI_Gather (sendbuf, sendcount, sendtype, *recvbuf*, recvcount, recvtype, root, comm)
- Arguments recv* not relevant on non-root processes
- recvcount → size of any (single) receive
- Distinct values (may be same) received from non-root processes at the root process
- Example: Reading multiple files (1 file per process)

Q: Equivalent point-to-point communications for the same?

- MPI_Recv at root
- MPI_Send at non-root



Example – Gather at 0

int MPI_Gather (sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, root, comm)

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31

Input: rows, cols

column-wise sub-comms

MPI_Comm_split (…, color=myrank%cols, …)

or

MPI_Comm_split (…, color=myrank/cols, …)

row-wise sub-comms

//newrank | newsize | newcomm

MPI_Gather (…., 0, … newcomm)

Next? (depends on comm_split)

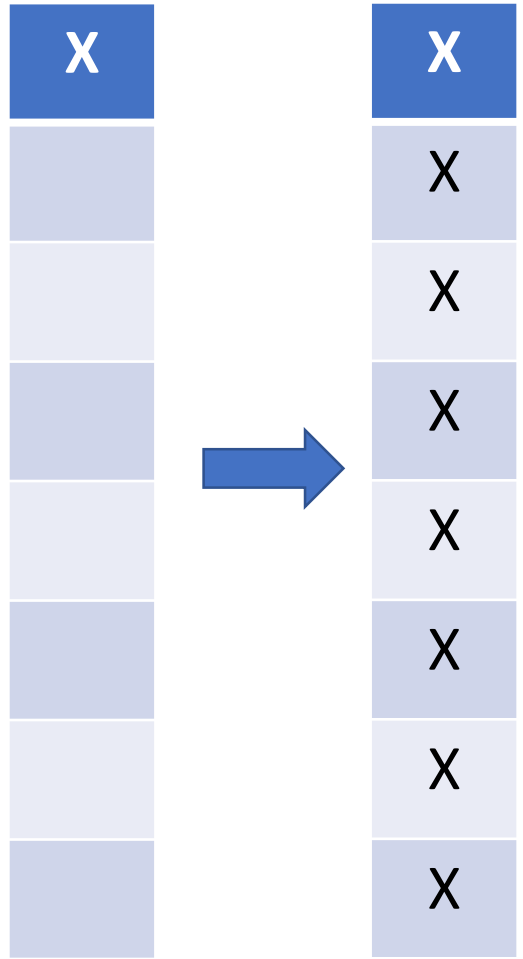
MPI_Group_incl (…)

MPI_Comm_create_group

(comm, group, tag, collcomm)

Broadcast

- Root process sends message to all processes
- int MPI_Bcast (buffer, count, datatype, root, comm)
- “count” is the number of elements in “buffer” – must match
  - “message sizes do not match across processes in the collective routine: Received 400 but expected 4000”
- “buffer” is input at root, output at non-root
- Any process can be a root process but has to be the same process when MPI_Bcast is called

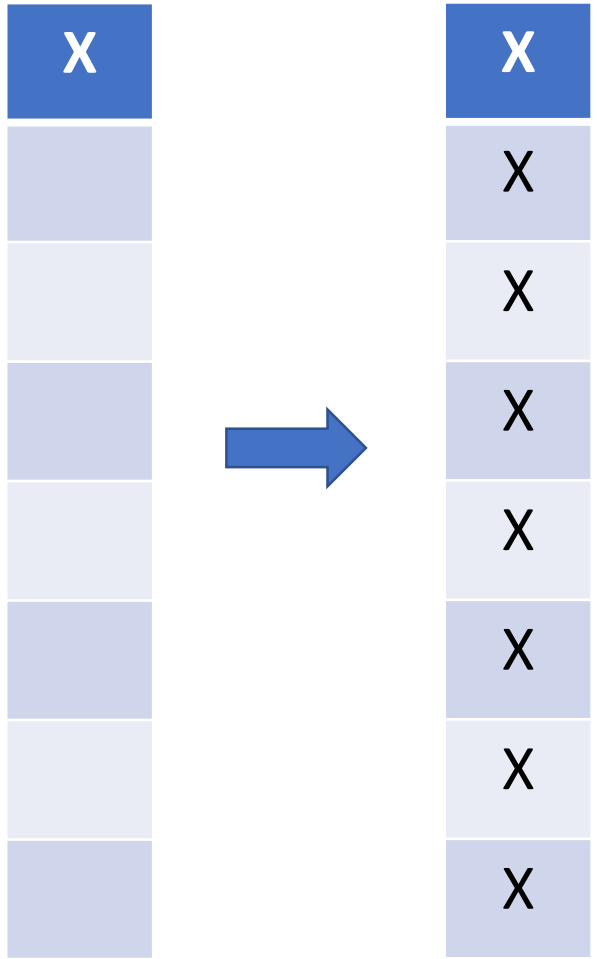


Broadcast

Q1: Point-to-point communication for the same?

Q2: Process 0 has read a file. The file size may vary across runs. Process 0 has to broadcast the file content to all processes. How do we achieve this?

Buffer size of "buffer" array is not known apriori at non-root processes, how should root broadcast buffer?



Homework

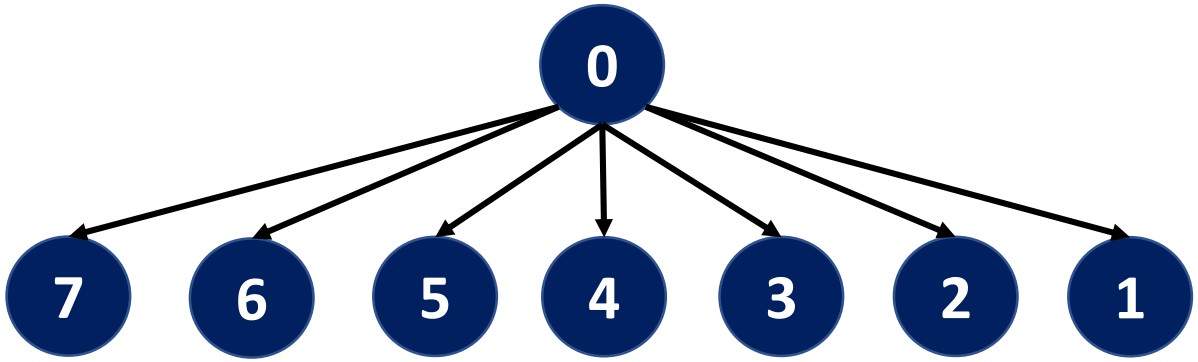
Write a program to split the total #processes (P) into G groups {G_0, G_1, …}, and the rank (P/G-1) of each group should broadcast its global rank and the global rank of the local rank 0 (as a pair of integers) of the group to all its group members. Assume G is divisible by P.

Broadcast – Naïve Algorithm

- Root process sends to every other process

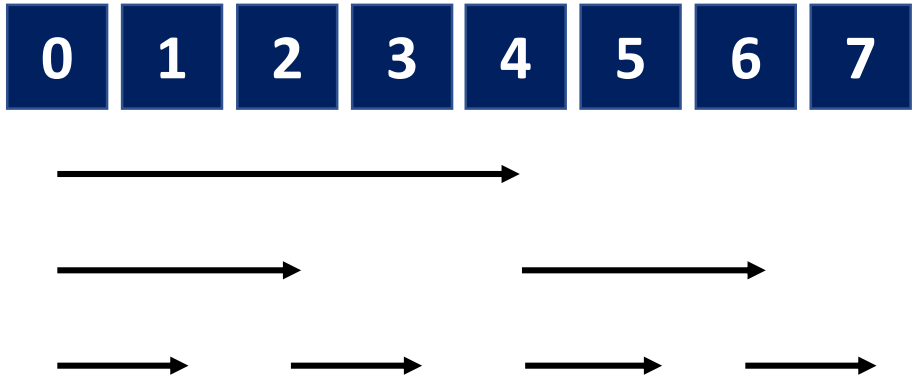
Cons

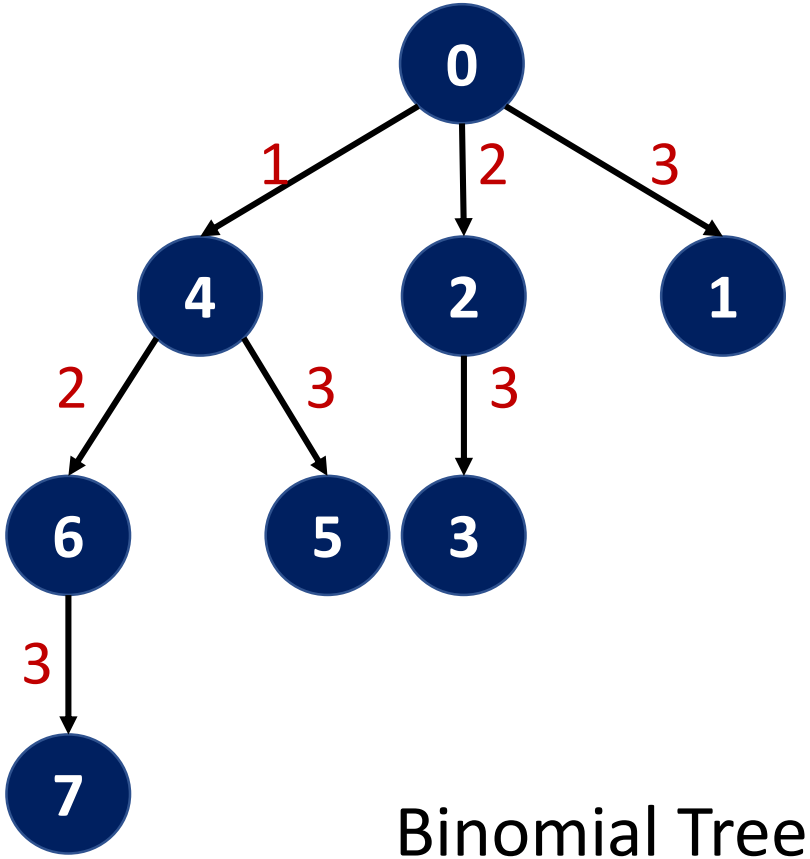
- Root is a bottleneck
- Poor scalability
- Idling processes
- Communication links are under-utilized



Broadcast – Optimized?

Write the P2P code





Binomial Tree

- #Steps for p (=2^d) processes?
  - log p
- Communication time for n bytes
  - T(p) = log p * (L + n/B)
  - $T(p^2)$ = 2 log p * (L + n/B)

MPI_Bcast Examples

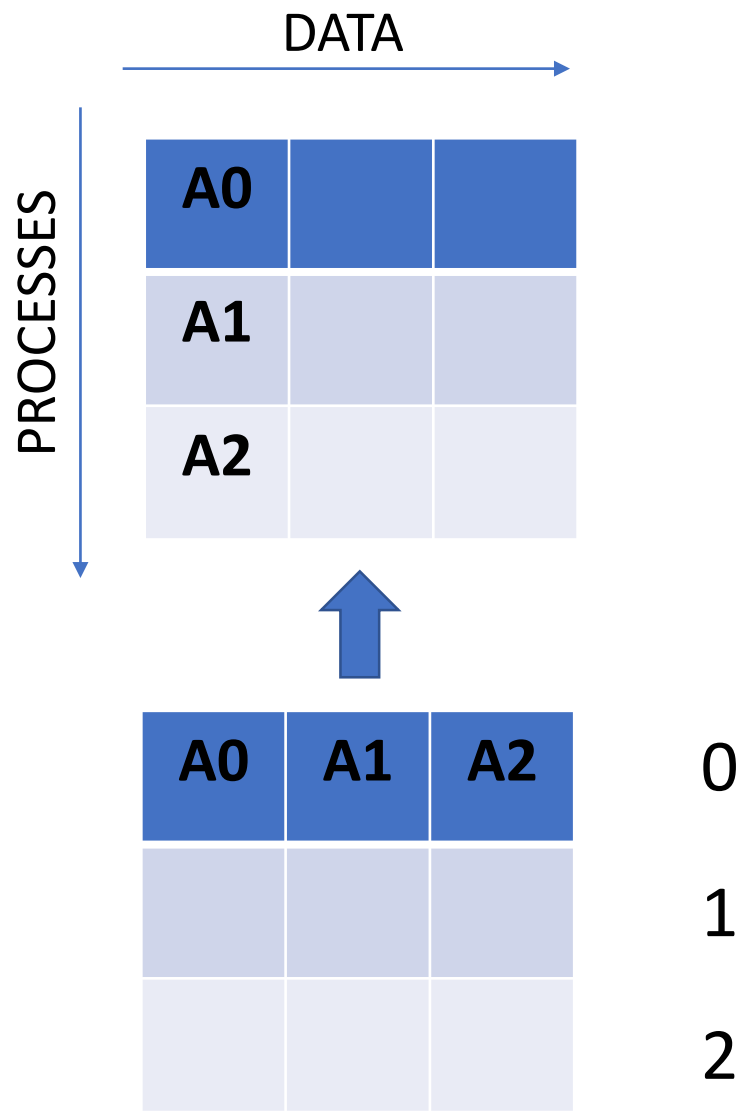
```
// … initialization tasks

int color=irand();
if (myrank != 3) MPI_Bcast (&color, 1, MPI_INT, 0, MPI_COMM_WORLD);
printf (“%d: %d\n”, myrank, color);
```

Output for n=4?

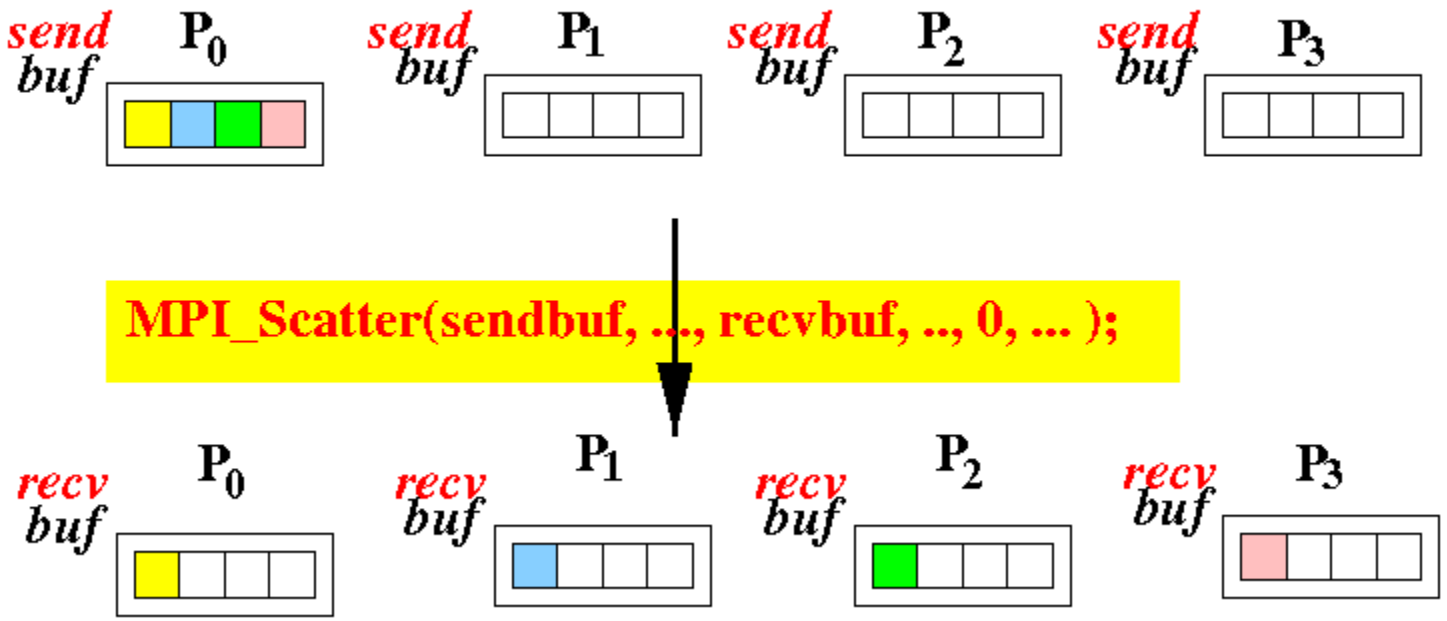
May succeed but not safe

Scatter

- Scatters values to all processes from a root process
- int MPI_Scatter (sendbuf, sendcount, sendtype, *recvbuf*, recvcount, recvtype, root, comm)
- Arguments send* irrelevant in non-root processes
- sendcount – Number of elements sent to each process
- Output parameter – recvbuf



MPI_Scatter Illustration



Credit: Shun Yan Cheung

Reduce

MPI_Reduce (inbuf, *outbuf*, count, datatype, op, root, comm)

- Combines element in inbuf of each process
- Combined value in outbuf of root
- op: MIN, MAX, SUM, PROD, …
- Example:

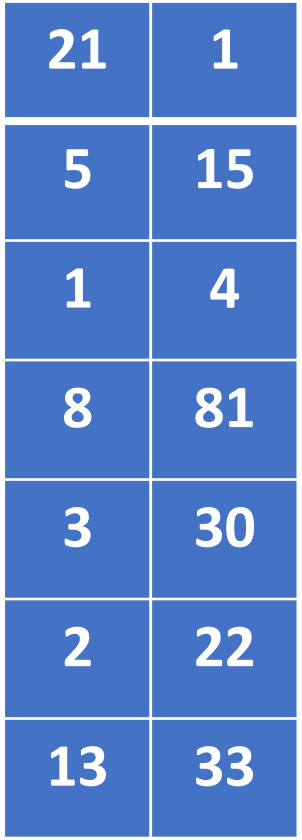
MPI_Reduce (…, 1, MPI_INT, MPI_MAX, …) Output: 21

MPI_Reduce (…, 1, MPI_INT, MPI_MIN, …) Output: 1

Reduce

MPI_Reduce (inbuf, *outbuf*, count, datatype, op, root, comm)

MPI_Reduce (…, 2, MPI_INT, MPI_MAX, …)
MPI_Reduce (…, 2, MPI_INT, MPI_SUM, …)



Reduce

MPI_Reduce (inbuf, *outbuf*, count, datatype, op, root, comm)

MPI_Reduce (…, 1, MPI_FLOAT, MPI_PROD, 0, …)
MPI_Reduce (…, 1, MPI_FLOAT, MPI_SUM, 0, …)

MPI_Reduce (…, 1, MPI_FLOAT, MPI_PROD, 5, …)
MPI_Reduce (…, 1, MPI_FLOAT, MPI_SUM, 5, …)

Reduce

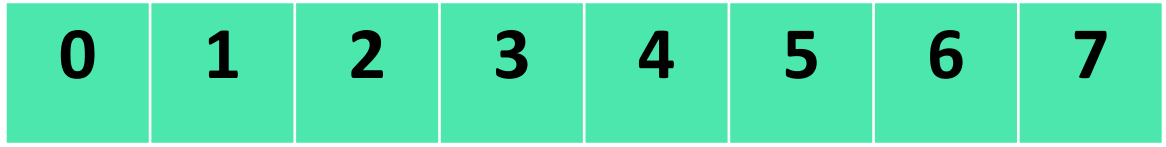
```
MPI_Reduce(&flnum, &val_p, 1, MPI_FLOAT, MPI_PROD, 0, MPI_COMM_WORLD);
MPI_Reduce(&flnum, &val_p, 1, MPI_FLOAT, MPI_PROD, 5, MPI_COMM_WORLD);

if (myrank== 0 || myrank == 5)
   printf ("%d: %f %f\n", myrank, val_s, val_p);
```

260855.593750

260855.609375

What Every Computer Scientist Should Know About Floating-Point Arithmetic, by David Goldberg, published in the March, 1991 issue of Computing Surveys.

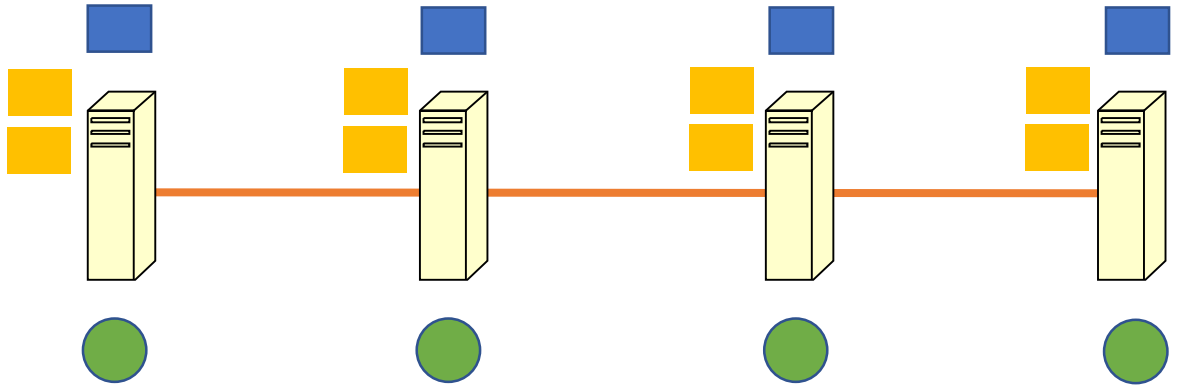
Data Input Example – File read by 0




```
If (rank == 0)
    read data

// bcast count per process

// scatter (use the above count)
```

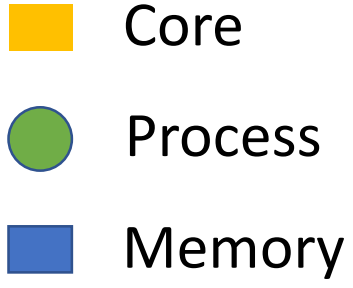
int MPI_Scatter (sendbuf, sendcount, sendtype, *recvbuf*, recvcount, recvtype, root, comm)

Parallel Sum



for i = 0 to N/P sum += a[i] * a[i]	for i = N/P to 2N/P sum += a[i] * a[i]	for i = 2N/P to 3N/P sum += a[i] * a[i]	for i = 3N/P to N sum += a[i] * a[i]

Rank = 0 Rank = 1 Rank = 2 Rank = 3



Alternately:
Process 0 can scatter the required part of the array

Using Send/Recv

```
int recvarr[numtasks];

// receive partial sums at rank 0
stime = MPI_Wtime();
if (rank)
{
  MPI_Send(&sum, 1, MPI_INT, 0, rank, MPI_COMM_WORLD);
}
else
{
  for (int r=1; r<numtasks; r++)
  {
    MPI_Recv(&recvarr[r], 1, MPI_INT, r, r, MPI_COMM_WORLD, &status);
  }
}
etime = MPI_Wtime();
cotime = etime - stime;

stime = MPI_Wtime();
// HOMEWORK
// Add the partial sums
for (int r=1; r<numtasks; r++)
  sum += recvarr[r];
etime = MPI_Wtime();
ctime += etime - stime;
```

Parallel Sum

```
// local computation at every process
for i = N/P * rank ; i < N/P * (rank+1) ; i++
    localsum += a[i] * a[i]

// collect localsum, add up at one of the ranks
MPI_Reduce (&localsum, … , MPI_SUM, …)
```

Using Reduce

```
// local sum computation
sum=0.0;
stime = MPI_Wtime();
for (i=sidx; i<sidx+N/numtasks ; i++)
    sum += a[i] * a[i];
etime = MPI_Wtime();
ctime = etime - stime;

int globalsum;

stime = MPI_Wtime();
MPI_Reduce (&sum, &globalsum, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
etime = MPI_Wtime();
cotime = etime - stime;


if (!rank)
printf ("%d %lf %lf\n", globalsum, ctime, cotime);
```

Timing

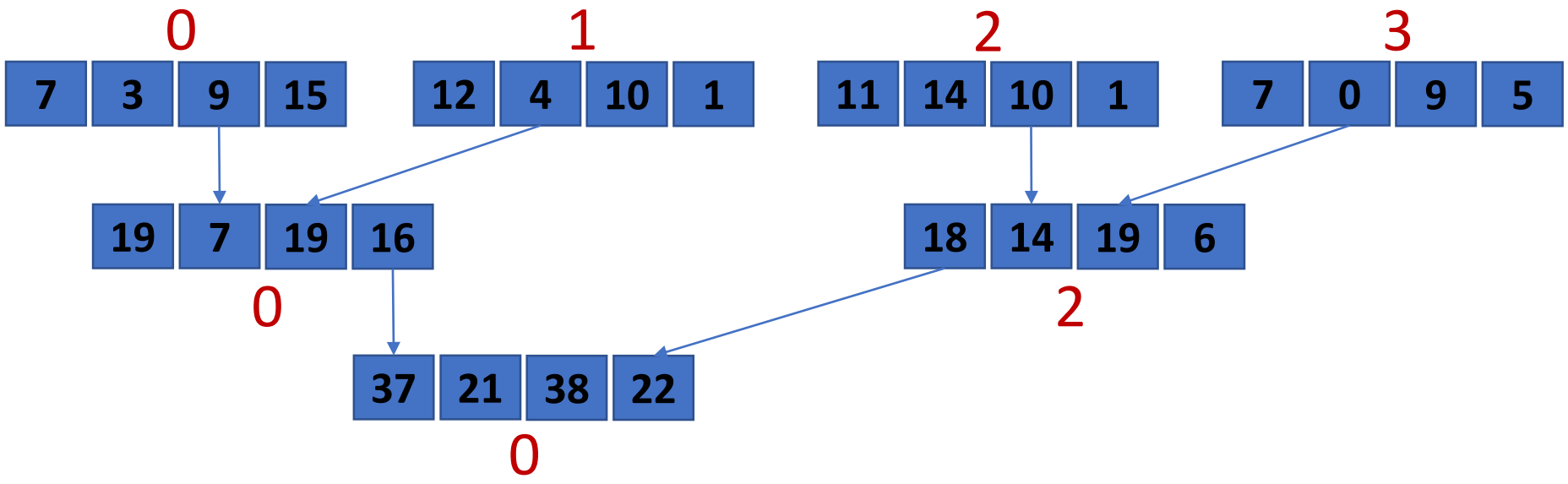
```
class $ for i in `seq 1 5`; do mpirun -np 3 ./parsum 6000 ; done
10000 0.000007 0.000301
10000 0.000006 0.000309
10000 0.000010 0.000015
10000 0.000006 0.000281
10000 0.000010 0.000041
class $ for i in `seq 1 5`; do mpirun -np 3 ./parsumreduce 6000 ; done
10000 0.000011 0.000172
10000 0.000010 0.000034
10000 0.000010 0.000033
10000 0.000010 0.000025
10000 0.000010 0.000028
```

Timing

```
class $ for i in `seq 1 5`; do mpirun -np 30 -hosts csews3:10,csews5:10,csews6:10 ./parsum 6000000 ; done
1711000000 0.001006 0.025789
1711000000 0.001034 0.010492
1711000000 0.001007 0.023931
1711000000 0.001029 0.038722
1711000000 0.001028 0.024971
class $ for i in `seq 1 5`; do mpirun -np 30 -hosts csews3:10,csews5:10,csews6:10 ./parsumreduce 6000000 ; done
1711000000 0.001003 0.011949
1711000000 0.002717 0.001540
1711000000 0.001092 0.009885
1711000000 0.003251 0.003160
1711000000 0.001091 0.012467
```

Reduce Algorithm

Recursive doubling

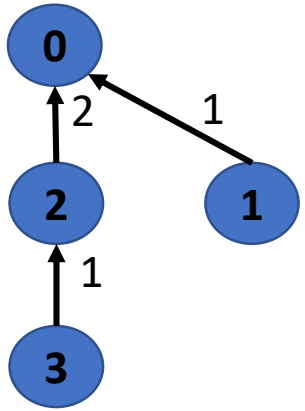


Time: log p (L + n*(1/B) + n*c)

L = latency, B = bandwidth

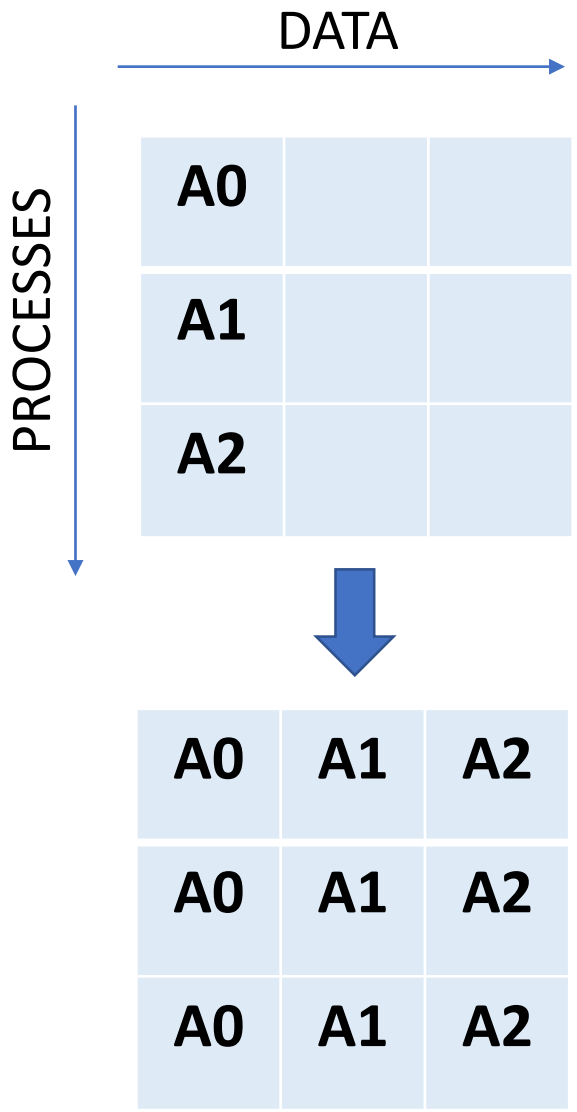
c = compute cost per byte

Used for short messages



Allgather

- All processes gather values from all processes
- int MPI_Allgather (sendbuf, sendcount, sendtype, *recvbuf*, recvcount, recvtype, comm)
- No root process




```
recvbuf = (int *) malloc
(size*100*sizeof(int));

MPI_Allgather (sendbuf, 100, MPI_INT,
recvbuf, 100, MPI_INT, comm);
```

Allreduce

- MPI_Allreduce (inbuf, *outbuf*, count, datatype, op, comm)
- op: MIN, MAX, SUM, PROD, …
- Combines element in inbuf of each process
- Combined value in outbuf of each process

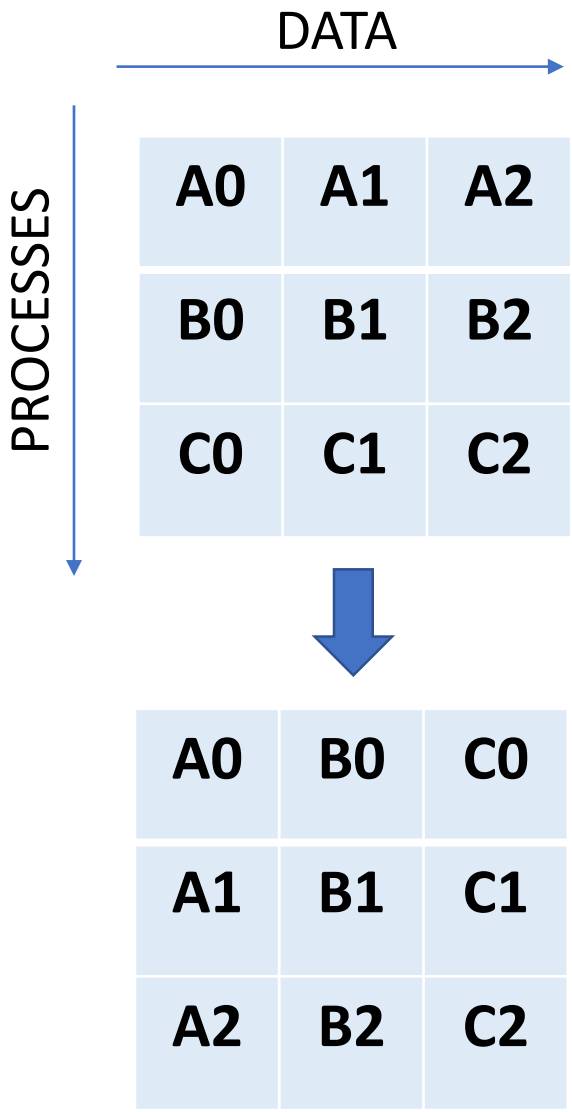
Equivalent collective?

- MPI_Reduce followed by MPI_Bcast

Alltoall

- Send data from all processes to all processes
- int MPI_Alltoall (sendbuf, sendcount, sendtype, *recvbuf*, recvcount, recvtype, comm)
- Output parameter – recvbuf
- sendcount/recvcount – sent/received from each process

Equivalent collective?



Homework

1. Broadcast N doubles from rank 0 to all ranks. You can run on any number of hosts (and processes). N = 10^3, 10^4, 10^5, 10^6, 10^7.

2. Let total number of processes be P. Compare performance of MPI_Gather with P2P on 1, 2, 4, 8 nodes (with ppn=4).

3. Let total number of processes be P. Compare performance of MPI_Scatter with P2P on 1, 2, 4, 8 nodes (with ppn=4).