Introduction to Parallel Programming with MPI

Dr. Alireza Ghasemi, Dr. Georg Hager

Erlangen National High Performance Computing Center

Blocking Collective Communication

Collectives in MPI

Collectives: operations including all ranks of a communicator

All ranks must call the function!

- Blocking variants: buffer can be reused after return
- Nonblocking variants (since MPI 3.0):
  buffer can be used after completion (**`MPI_Wait*`**/**`MPI_Test*`**)

- May or may not synchronize the processes
- Cannot interfere with point-to-point communication
  - Completely separate modes of operation!

Collectives in MPI

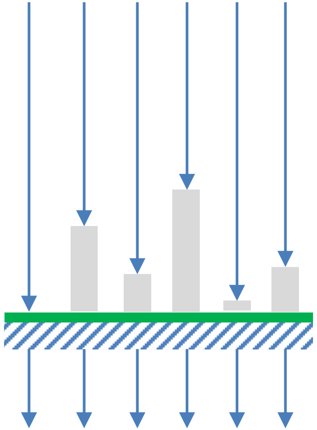
- Rules for all collectives
  - Data type matching
  - No tags
  - Count must be exact, i.e., there is only one message length, buffer must be large enough
- Types:
  - Synchronization (barrier)
  - Data movement (broadcast, scatter, gather, all to all)
  - Collective computation (reduction, scan)
  - Combinations of data movement and computation (reduction + broadcast)
- General assumption: MPI does a better job at collectives than you trying to emulate them with a collection of point-to-point calls

Barrier

- Explicit synchronization of all ranks from specified communicator

  ```
  MPI_Barrier(comm);
  ```

- Ranks only return from call after every rank has called the function



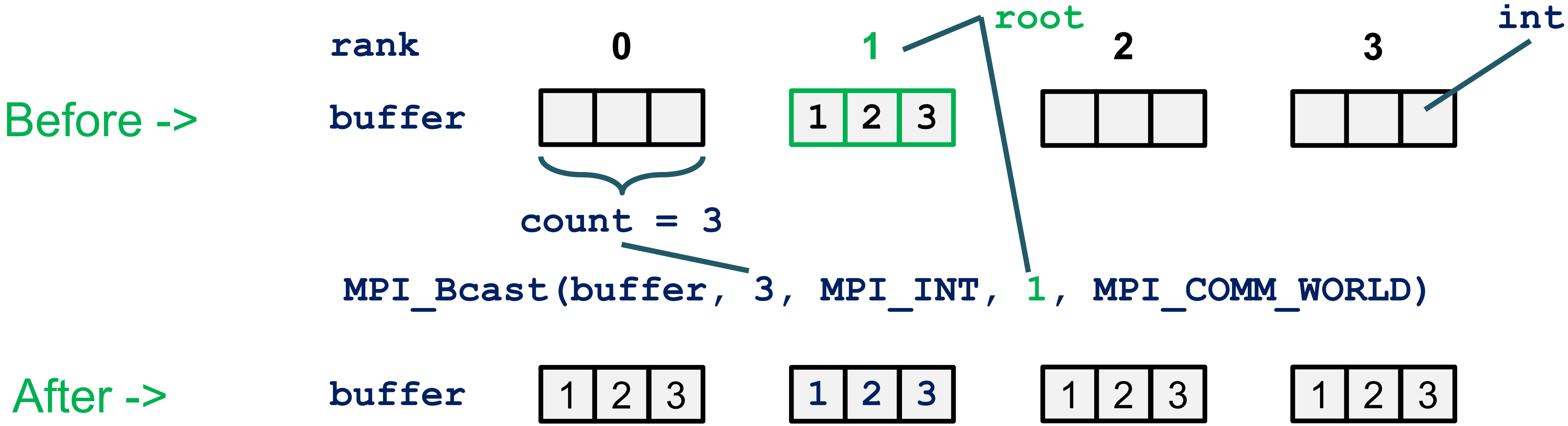
- `MPI_Barrier`: rarely needed
  - Debugging

Broadcast

- Send buffer contents from one rank (“root”) to all ranks

```
MPI_Bcast(buf, count, datatype, int root, comm);
```

- no restrictions on which rank is root – often rank 0



```
MPI_Bcast(buffer, 3, MPI_INT, 1, MPI_COMM_WORLD)
```

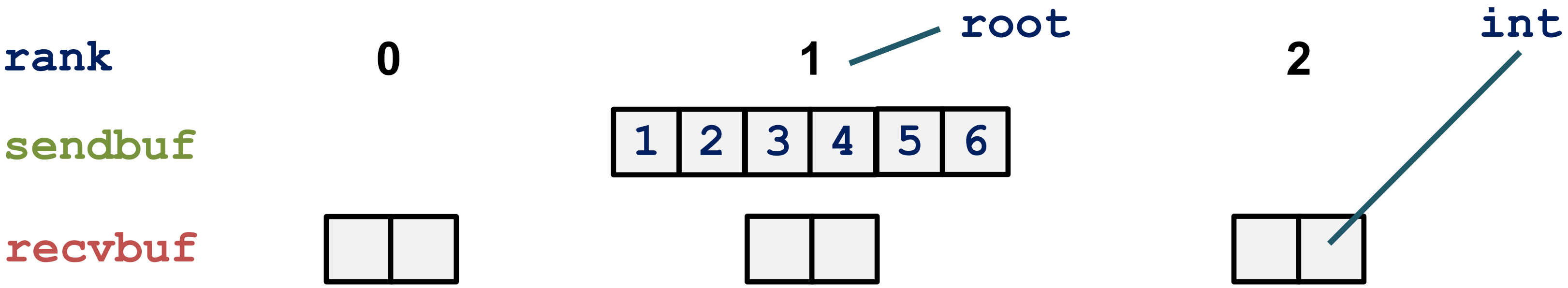
Scatter

- Send every i-th chunk of an array to the i-th rank

```
MPI_Scatter(sendbuf, sendcount, sendtype,
            recvbuf, recvcount, recvtype,
            root, comm);
```

- Root and comm must be the same on all processes
- Type signature of send and receive variables must match
- Usually, **`sendcount`** = **`recvcount`** because **`sendtype = recvtype`**
  - This is the length of the chunk
- **`sendbuf`** is ignored on non-root ranks because there is nothing to send

Scatter



rank 0 1 2

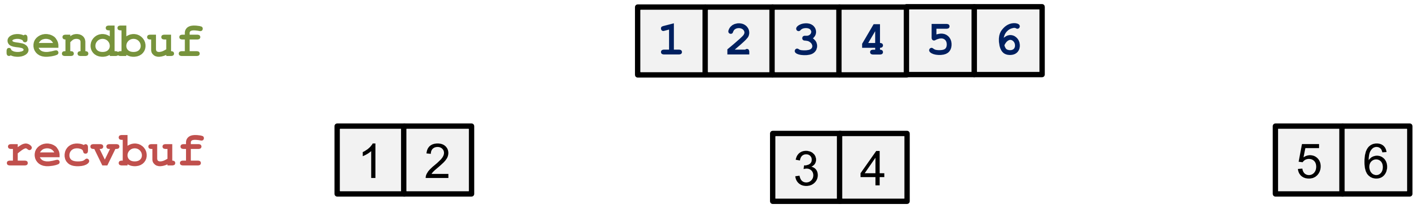
root

int

sendbuf 1 2 3 4 5 6

recvbuf

```
MPI_Scatter(sendbuf, 2, MPI_INT, recvbuf, 2, MPI_INT,
            root, MPI_COMM_WORLD)
```



sendbuf 1 2 3 4 5 6

recvbuf 1 2 3 4 5 6

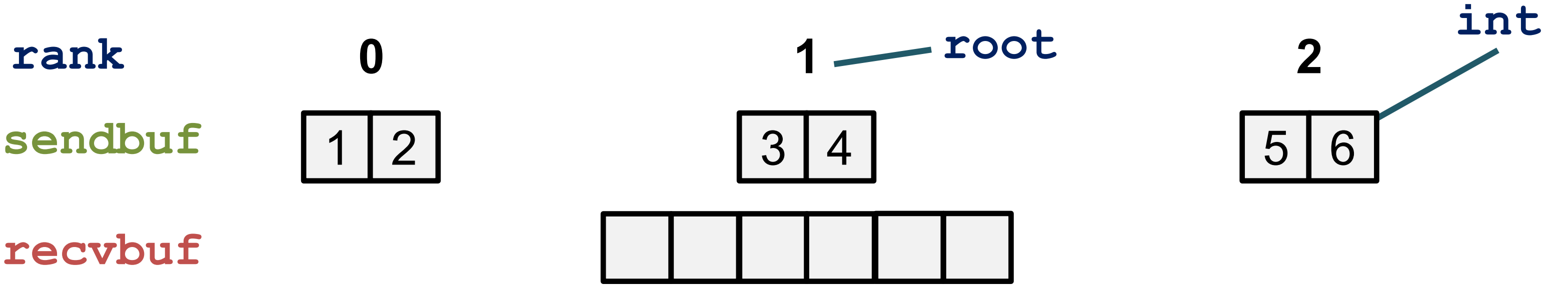
Gather

- Receive a message from each rank and place i-th rank’s message at i-th position in receive buffer

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

- Root and comm must be the same on all processes
- Type signature of send and receive variables must match
- Usually, **`sendcount`** = **`recvcount`** because **`sendtype = recvtype`**
- **`recvbuf`** is ignored on non-root ranks because there is nothing to receive

Gather

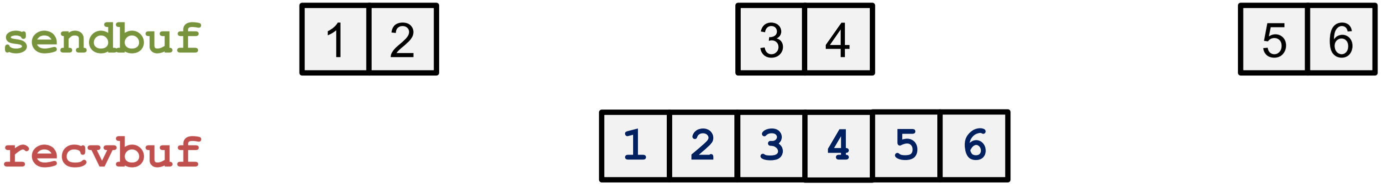


root

int

rank	0	1	2
sendbuf	1 2	3 4	5 6
recvbuf			

```
MPI_Gather(sendbuf, 2, MPI_INT, recvbuf, 2, MPI_INT,
           root, MPI_COMM_WORLD)
```



sendbuf	1 2	3 4	5 6
recvbuf		1 2 3 4 5 6	

Scatterv: more flexible scatter

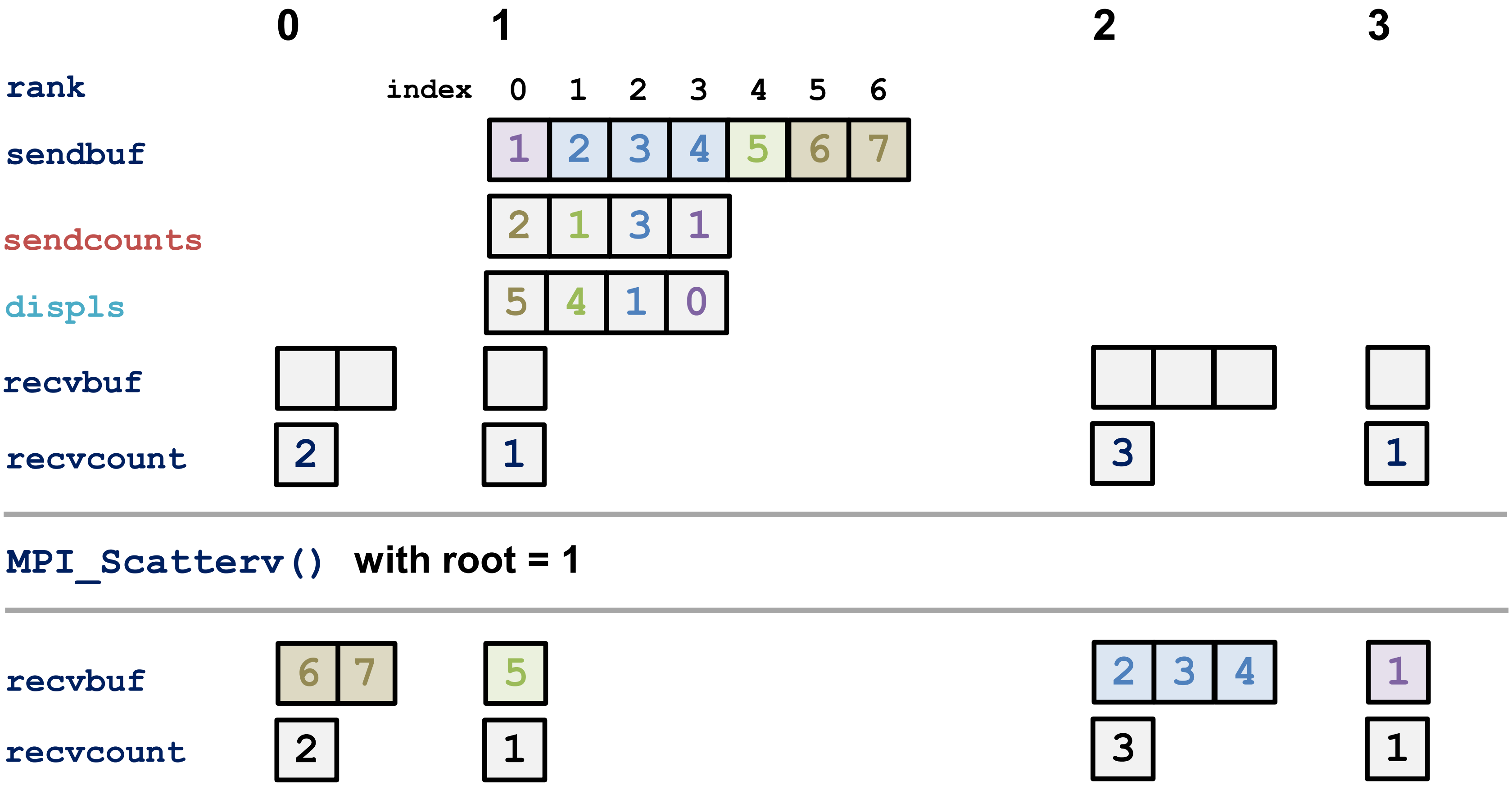
- Send chunks of different sizes to different ranks

```
MPI_Scatterv(
    sendbuf, int sendcounts[], int displs[], sendtype,
    recvbuf, recvcount, recvtype,
    root, comm);
```

`sendcounts[]`: array specifying the number of elements to send to each rank: send **`sendcounts[i]`** elements to rank **`i`**

`displs[]`: integer array specifying the displacements in **`sendbuf`** from which to take the outgoing data to each rank, specified in number of elements

Scatterv



rank	0	1	2	3

index	0	1	2	3	4	5	6
sendbuf	1	2	3	4	5	6	7
sendcounts	2	1	3	1			
displs	5	4	1	0			

rank	0	1	2	3
recvbuf	□□	□	□□□	□
recvcount	2	1	3	1

MPI_Scatterv() with root = 1

rank	0	1	2	3
recvbuf	6 7	5	2 3 4	1
recvcount	2	1	3	1

Gatherv: more flexible gather

- Receive segments of different sizes from different ranks

```
MPI_Gatherv(
    sendbuf, sendcount, sendtype,
    recvbuf, int recvcounts[], int displs[], recvtype,
    root, comm)
```

`recvcounts[]`: array specifying the number of elements to receive from each rank: receive **`recvcounts[i]`** elements from rank **`i`**

`displs[]`: integer array specifying the displacements where received data from specific rank is put in **`recvbuf`**, in units of elements:

Allgather

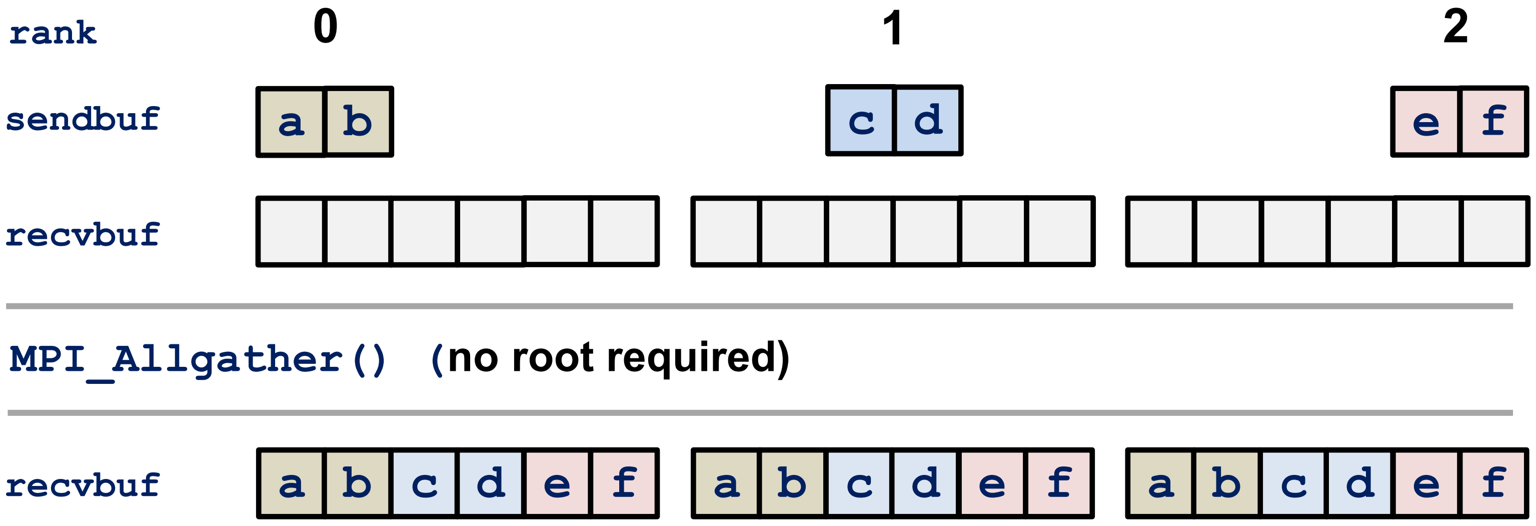
- Combination of gather and broadcast

```
MPI_Allgather(sendbuf, sendcount, sendtype,
              recvbuf, recvcount, recvtype,
              comm);
```

- Also available: `MPI_Allgatherv()` (cf. `MPI_Gatherv()`)

- Why not just use gather followed by a broadcast instead?
  - MPI library has more options for optimization
  - General assumption: Combined collectives are faster than using separate ones

Allgather



rank 0 1 2

sendbuf a b c d e f

recvbuf

`MPI_Allgather()` (no root required)

recvbuf a b c d e f a b c d e f a b c d e f

In this example: `sendcount=recvcount=2`

Alltoall

- `MPI_Alltoall`: For all ranks, send i-th chunk to i-th rank

  ```
  MPI_Alltoall(sendbuf, sendcount, sendtype,
               recvbuf, recvcount, recvtype,
               comm);
  ```

- `MPI_Alltoallv`: Allows different number of elements to be send/received by each rank

- `MPI_Alltoallw`: Allows also different data types and displacements in bytes

Alltoall

`rank`	0				1				2				3			
`sendbuf`	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
`sendcount`	1				1				1				1			
`recvbuf`																
`recvcount`	1				1				1				1			

`MPI_Alltoall()` (no root required)

`recvbuf`	0	4	8	12	1	5	9	13	2	6	10	14	3	7	11	15

Summary of MPI Collective Communications

- **MPI (blocking) collectives**
  - All ranks in communicator must call the function

- **Communication and synchronization**
  - Barrier, broadcast, scatter, gather, and combinations thereof

- **In-place buffer specification `MPI_IN_PLACE`**
  - Save some space if need be

Quiz:

1) Why should one use collective communication rather than emulating by a set of point-to-point calls?

2) Can MPI collective communications interfere with point-to-point calls?

a) Yes b) No

3) For a collective communication, it is not necessary every process of a communicator to call it?

a) Correct b) Incorrect

Quiz:

4) To send an identical piece of data to all other processes in a communicator, which collective call should be used?

   a) `MPI_Gather`

   b) `MPI_Bcast`

   c) `MPI_Scatter`

   d) `MPI_Alltoall`

5) Which of the following collective calls is similar to the process of transposing a matrix in mathematics?

   a) `MPI_Gather`

   b) `MPI_Bcast`

   c) `MPI_Scatter`

   d) `MPI_Alltoall`