MPI - Collective Communication

Overview of Collective Communication

- It allows exchanging data among a group of processes.
- It must involve all processes in the scope of a communicator.
- The communicator argument in a collective communication routine should specify which processes are involved in the communication.
- It is the programmer's responsibility to ensure that all processes within a communicator participate in any collective operation.

Collective Communication Patterns

Operations:

1. Broadcast
2. Scatter
3. Gather
4. All-gather
5. All-to-All
6. Reduce
7. All-Reduce
8. Scan
9. Reduce-scatter
10. Barrier

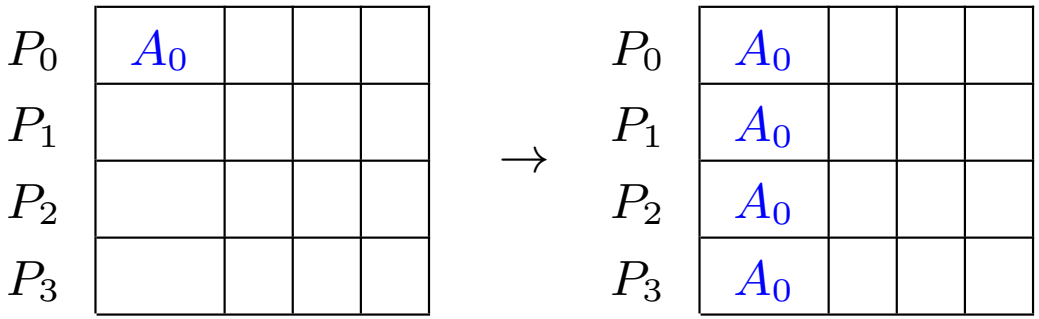
Classification by operation mode:

1. One-To-All Mode: One process contributes to the results. All processes receive the result.
   - `MPI_Bcast()`, `MPI_Scatter()`
2. All-to-One Mode: All processes contribute to the result. One process receives the result.
   - `MPI_Gather()`, `MPI_Reduce()`
3. All-to-All Mode: All processes contribute to the result. All processes receive the result.
   - `MPI_Alltoall()`, `MPI_Allgather()`
   - `MPI_Allreduce()`, `MPI_Reduce_scatter()`
4. Other: operations that do not fit into the above categories
   - `MPI_Scan()`, `MPI_Barrier()`

Broadcast

- `root` process broadcasts a message to all other processes in the group.
- On return, the content of root's buffer has been copied to all processes.

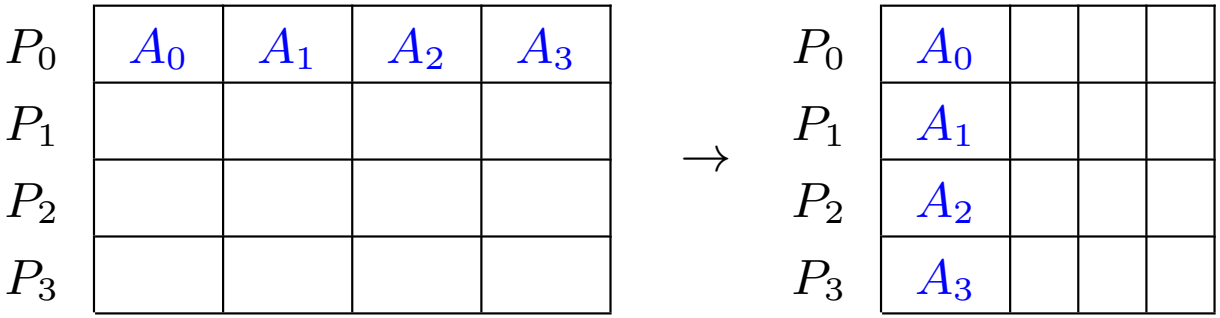
```
int MPI_Bcast (void *buffer, int count, MPI_Datatype datatype,
               int root, MPI_Comm comm)
```



Scatter

Splits the data in `sendbuf` into p segments, each of which has size `sendcnt` of type `sendtype`. The first segment is sent to process 0, the second to process 1, etc.

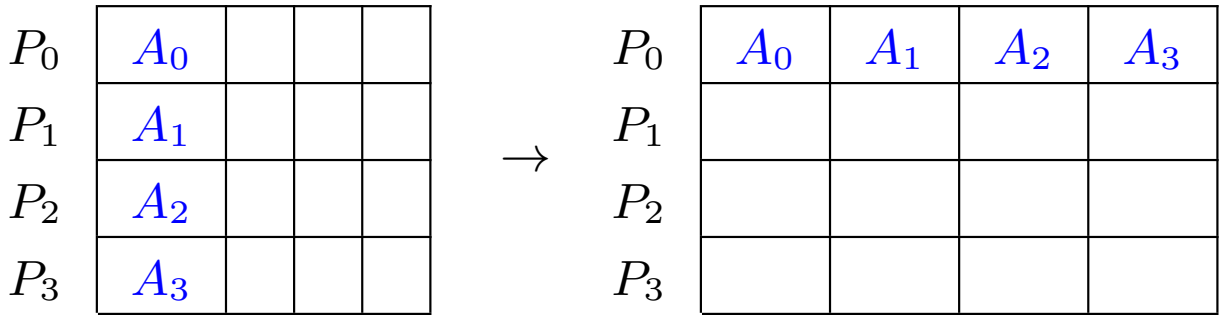
```
int MPI_Scatter(void *sendbuf, int sendcnt, MPI_Datatype sendtype,
                void *recvbuf /*out*/, int recvcnt,
                MPI_Datatype recvtype,  int root, MPI_Comm comm);
```



Gather (All-To-One)

- Each process in the same communicator sends contents in `sendbuf` to `root`
- root stores received contents in rank order
- `recvbuf` is the address of receive buffer, which is significant only at `root`
- `revcnt` is the size for any single receive, which is significant only at `root`.

```
int MPI_Gather(void *sendbuf, int sendcnt, MPI_Datatype sendtype,
               void *recvbuf /* out */, int recvcnt,
               MPI_Datatype recvtype, int root, MPI_Comm comm)
```

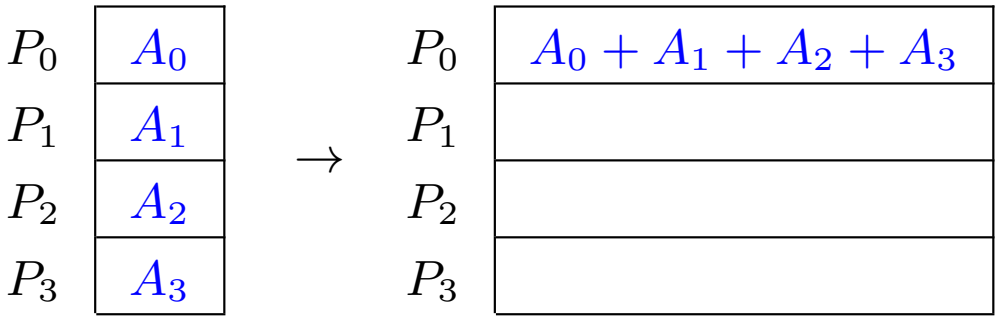


Reduce (All-To-One)

- This routine combines values in `sendbuf` on all processes to a single value using the specified operation `op`.
- The combined value is put in `recvbuf` of the process with rank `root`.

```
int MPI_Reduce( void *sendbuf, void *recvbuf /* out */,
                int count, MPI_Datatype datatype, MPI_Op op,
                int root, MPI_Comm comm)
```

E.g.

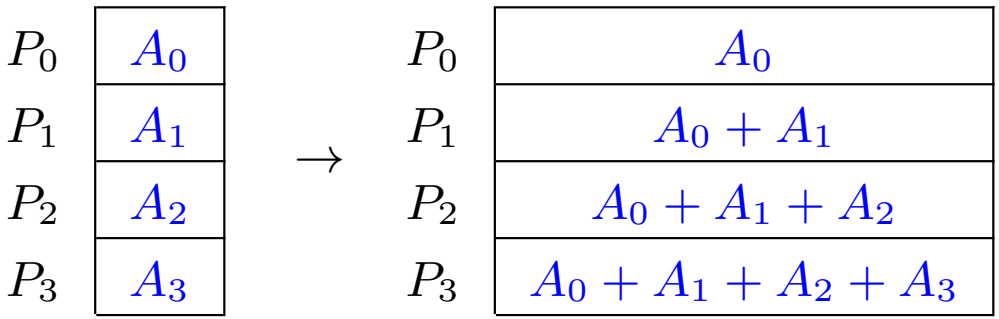


Scan

Scan computes partial reductions of data on a collection of processes.

```
int MPI_Scan( void *sendbuf, void *recvbuf /* out */,  int count,
              MPI_Datatype datatype, MPI_Op op, MPI_Comm comm)
```

E.g.

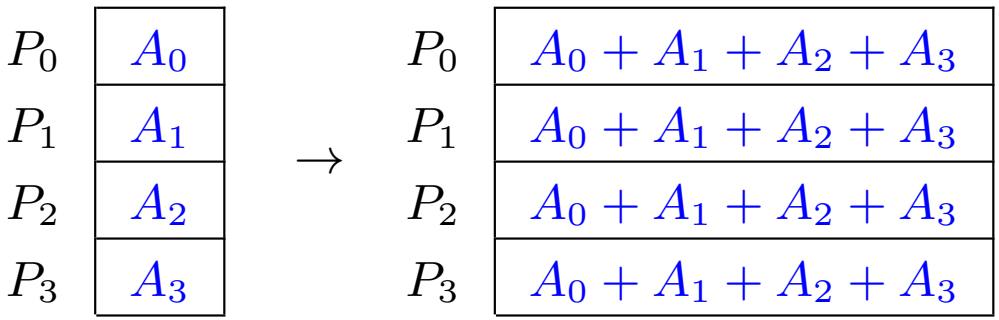


All Reduce (All-To-All)

- Allreduce has the same interface as reduce, but places the result in all processes.
- It is equivalent to `MPI_Reduce` followed by an `MPI_Bcast`.

```
int MPI_Allreduce( void *sendbuf, void *recvbuf /* out */,
                   int count, MPI_Datatype datatype, MPI_Op op,
                   int root, MPI_Comm comm)
```

E.g.



Predefined Reduction Operations

```
MPI_MAX
MPI_MIN
MPI_SUM
MPI_PROD
MPI_LAND      // logical and
MPI_LOR
MPI_BAND      // bit-wise and
MPI_BOR
MPI_LXOR
MPI_BXOR
MPI_MINLOC      // min value and location
MPI_MAXLOC
```

Reduce Scatter

Reduce Scatter combines values and scatters the results

```
int MPI_Reduce_scatter( void *sendbuf, void *recvbuf /* out */,  int *recv
                MPI_Datatype datatype, MPI_Op op, MPI_Comm comm)
```

E.g.

P_0	A_0	A_1	A_2	A_3
P_1	B_0	B_1	B_2	B_3
P_2	C_0	C_1	C_2	C_3
P_3	D_0	D_1	D_2	D_3

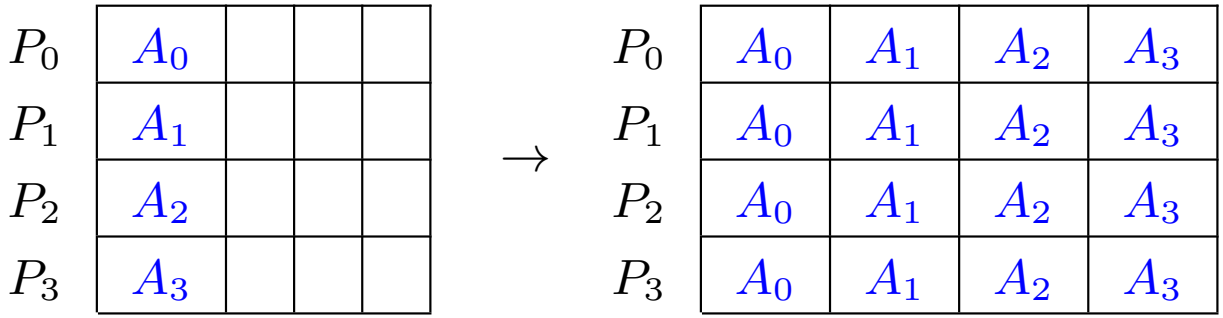
$\rightarrow$

P_0	$A_0 + B_0 + C_0 + D_0$
P_1	$A_1 + B_1 + C_1 + D_1$
P_2	$A_2 + B_2 + C_2 + D_2$
P_3	$A_3 + B_3 + C_3 + D_3$

AllGather (All-To-All)

- Gather data from all processes and distribute the combined data to all processes
- `revcnt` is the size received from any process
- Similar to Gather + Bcast

```
int MPI_Allgather( void *sendbuf, int sendcnt, MPI_Datatype sendtype,
                   void *recvbuf /* out */, int recvcount,
                   MPI_Datatype recvtype, MPI_Comm comm)
```



AlltoAll (All-To-All)

- An extension of `MPI_Allgather` to case where each process sends distinct data to each of the receivers
- The type signature associated with `sendcount, sendtype` at a process must be equal to the type structure associated with `recvcount, recvtype` at any other process

```
int MPI_Alltoall( void *sendbuf, int sendcnt, MPI_Datatype sendtype,
                  void *recvbuf /* out */, int recvcount,
                  MPI_Datatype recvtype, MPI_Comm comm)
```

P_0	A_0	B_0	C_0	D_0
P_1	A_1	B_1	C_1	D_1
P_2	A_2	B_2	C_2	D_2
P_3	A_3	B_3	C_3	D_3

$\rightarrow$

P_0	A_0	A_1	A_2	A_3
P_1	B_0	B_1	B_2	B_3
P_2	C_0	C_1	C_2	C_3
P_3	D_0	D_1	D_2	D_3

MPI_Scatter - Code Example

```
#include "mpi.h"
#include <stdio.h>
#define SIZE 4
int main(int argc, char *argv[])  {
int procs, rank, sendcount, recvcount, src;
float sendbuf[SIZE][SIZE] = {
    {1.0, 2.0, 3.0, 4.0},
    {5.0, 6.0, 7.0, 8.0},
    {9.0, 10.0, 11.0, 12.0},
    {13.0, 14.0, 15.0, 16.0}  };
float recvbuf[SIZE];

MPI_Init(&argc,&argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &procs);
if (procs == SIZE) {
    src = 1;
    sendcount = SIZE;
    recvcount = SIZE;
    MPI_Scatter(sendbuf,sendcount,MPI_FLOAT,recvbuf,recvcount,
                MPI_FLOAT,src,MPI_COMM_WORLD);
    printf("rank= %d  Results: %.4f %.4f %.4f %.4f\n",rank,recvbuf[0],
        recvbuf[1],recvbuf[2],recvbuf[3]);
    }
  else
    printf("Must specify %d processors. Terminating.\n",SIZE);

 MPI_Finalize();
 return 0;
}
```

Barrier Synchronization

`MPI_Barrier(MPI_Comm comm)`

- block until all processes in the communicator have reached this routine.

```
#include <mpi.h>
#include <stdio.h>

int main(int argc, char *argv[]) {
  int rank, nprocs;
  MPI_Init(&argc,&argv);
  MPI_Comm_size(MPI_COMM_WORLD,&nprocs);
  MPI_Comm_rank(MPI_COMM_WORLD,&rank);
  MPI_Barrier(MPI_COMM_WORLD);
  printf("Hello, world. I am %d of %d\n", rank, procs);
  fflush(stdout);
  MPI_Finalize();
  return 0;
}
```