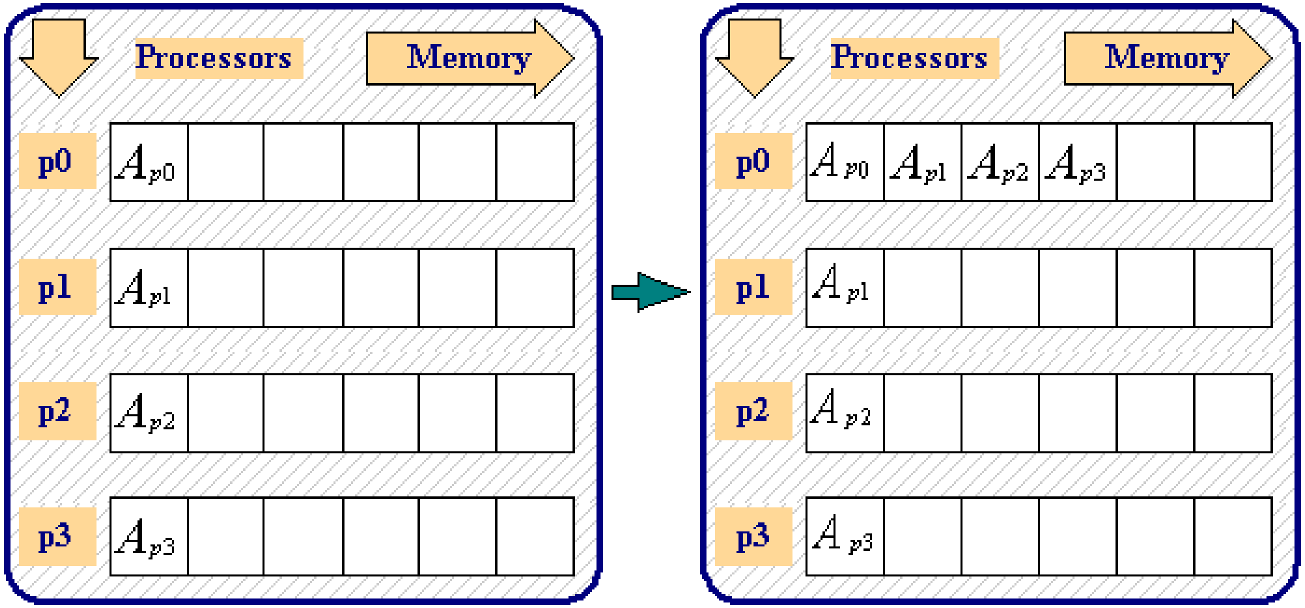
Parallel programming

MPI Interface

Gather

- The *MPI_GATHER* routine is an *all-to-one* communication
- When MPI_GATHER is called, each process (including the root process) sends the contents of its send buffer to the root process
- The root process receives the messages and stores them in rank order



Gather

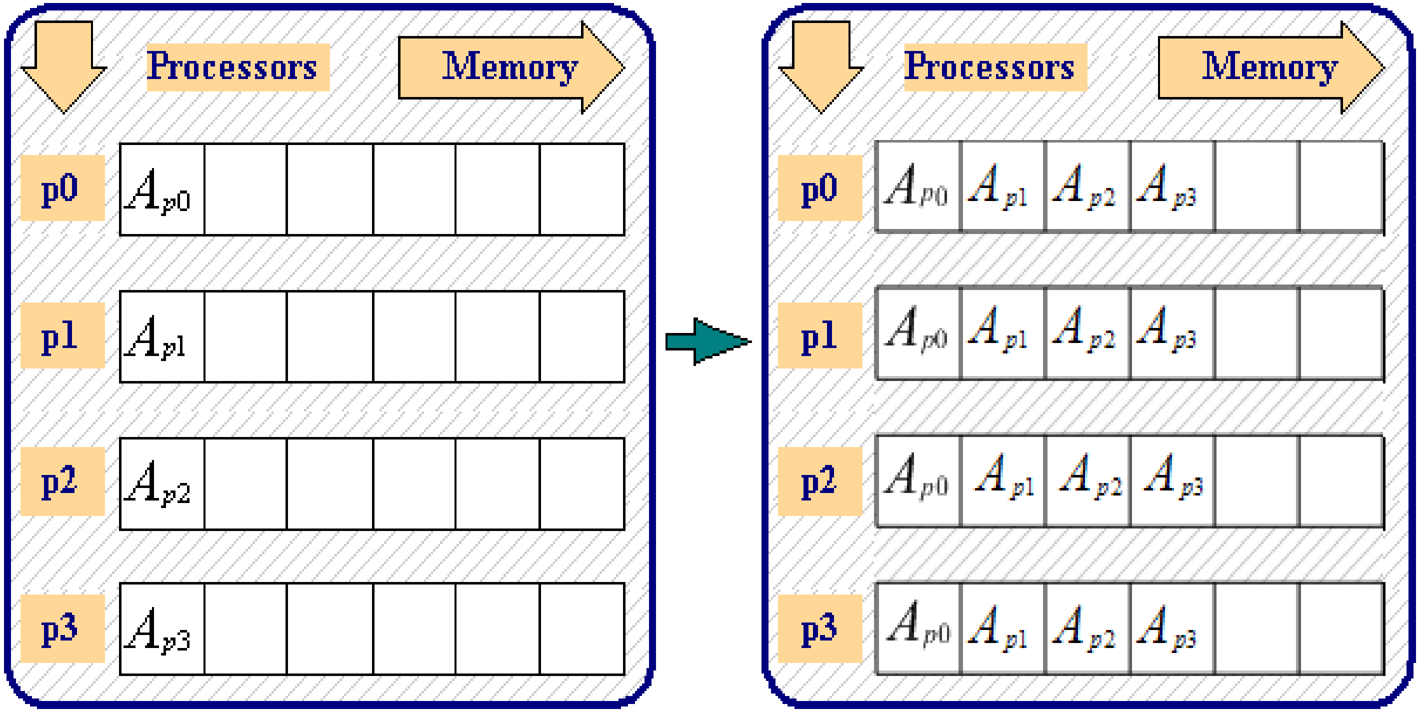
-
  ```
  int MPI_Gather ( void* send_buffer, int send_count,
  MPI_datatype send_type, void* recv_buffer, int
  recv_count, MPI_Datatype recv_type, int rank, MPI_Comm
  comm )
  ```

- send_buffer   in   starting address of send buffer
- send_count   in   number of elements in send buffer
- send_type   in   data type of send buffer elements
- recv_buffer   out   starting address of receive buffer
- recv_count   in   number of elements in receive buffer for a single receive
- recv_type   in   data type of elements in receive buffer
- recv_rank   in   rank of receiving process
- comm   in   mpi communicator

Example 05 - Gather

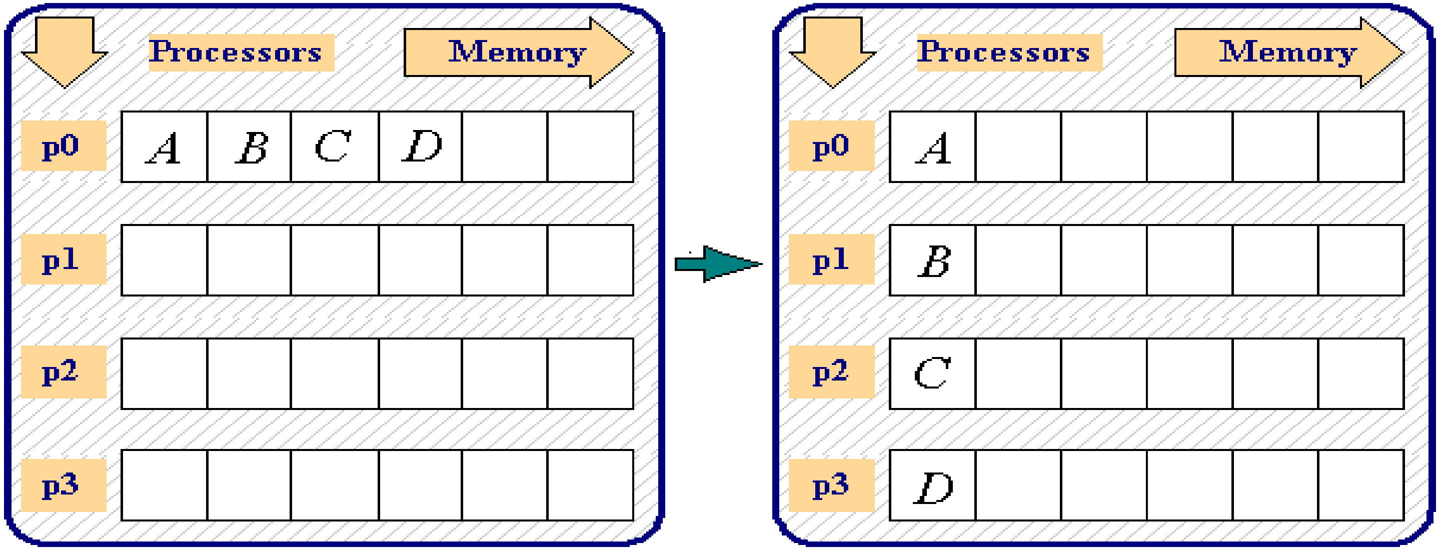
MPI_ALLGATHER

- MPI_GATHER + MPI_BCAST = ***MPI_ALLGATHER***


Processors
Memory
p0
p1
p2
p3
A_{p0}
A_{p1}
A_{p2}
A_{p3}
Processors
Memory
p0
p1
p2
p3
A_{p0} A_{p1} A_{p2} A_{p3}
A_{p0} A_{p1} A_{p2} A_{p3}
A_{p0} A_{p1} A_{p2} A_{p3}
A_{p0} A_{p1} A_{p2} A_{p3}

Scatter

- The *MPI_SCATTER* routine is a *one-to-all* communication
- Different data are sent from the root process to each process **(in rank order)**



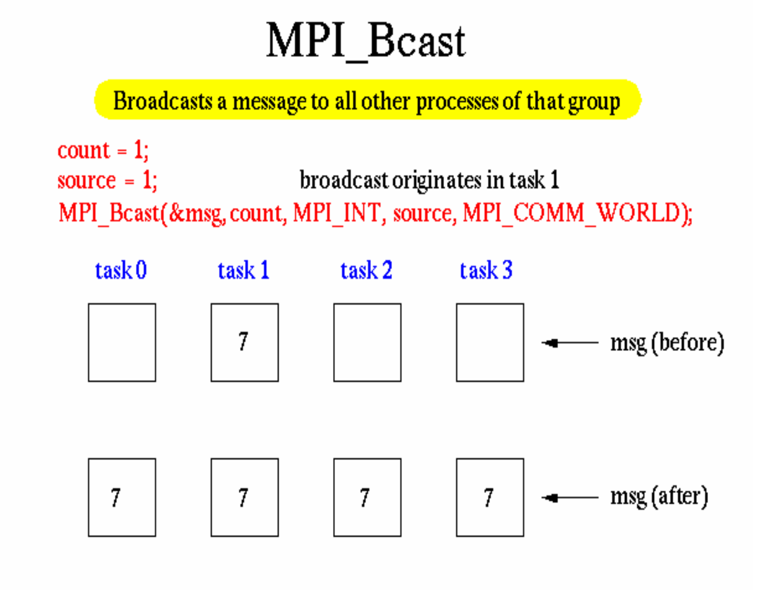
Scatter

- `MPI_Scatter ( send_buffer, send_count, send_type, recv_buffer, recv_count, recv_type, rank, comm )`

- send_buffer - starting address of send buffer
- send_count - number of elements in send buffer to send to each process (not the total number sent)
- send_type - data type of send buffer elements
- recv_buffer - starting address of receive buffer
- recv_count - number of elements in receive buffer
- recv_type - data type of elements in receive buffer
- rank - rank of sending process
- comm - mpi communicator

Example 06 - Scatter

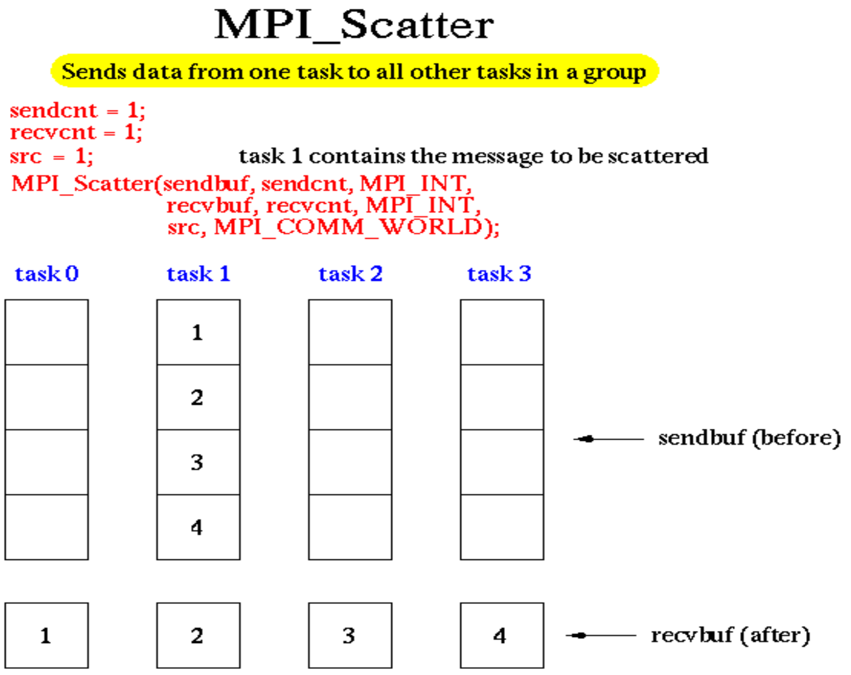
Summary of collective communication



MPI_Bcast

Broadcasts a message to all other processes of that group

```
count = 1;
source = 1;                broadcast originates in task 1
MPI_Bcast(&msg, count, MPI_INT, source, MPI_COMM_WORLD);
```

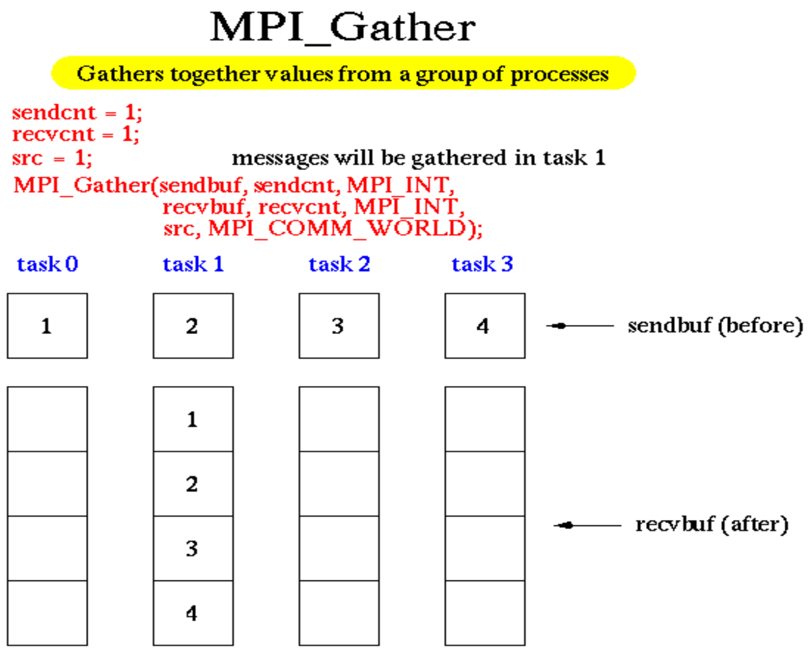


MPI_Scatter

Sends data from one task to all other tasks in a group

```
sendcnt = 1;
recvcnt = 1;
src  = 1;                 task 1 contains the message to be scattered
MPI_Scatter(sendbuf, sendcnt, MPI_INT,
            recvbuf, recvcnt, MPI_INT,
            src, MPI_COMM_WORLD);
```

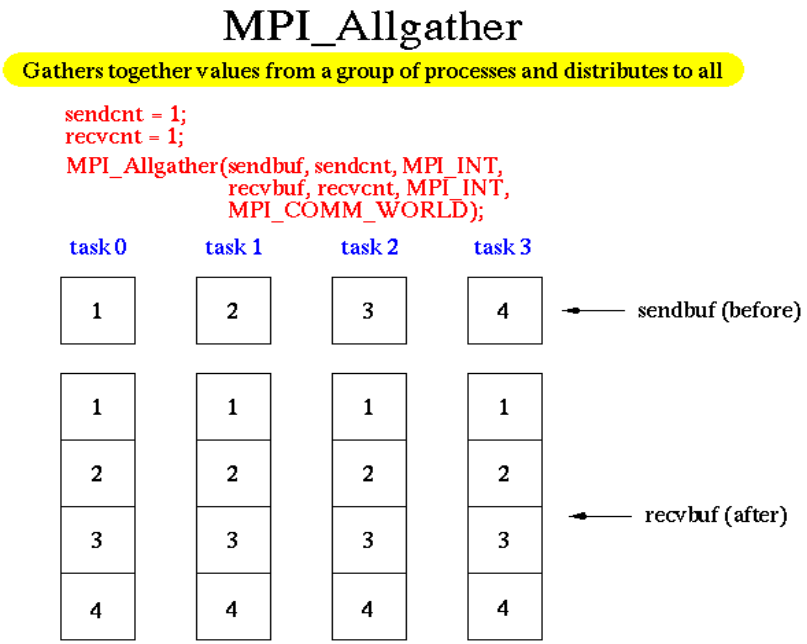
Summary of collective communication



MPI_Gather

Gathers together values from a group of processes

```
sendcnt = 1;
recvcnt = 1;
src  = 1;                  messages will be gathered in task 1
MPI_Gather(sendbuf, sendcnt, MPI_INT,
           recvbuf, recvcnt, MPI_INT,
           src, MPI_COMM_WORLD);
```

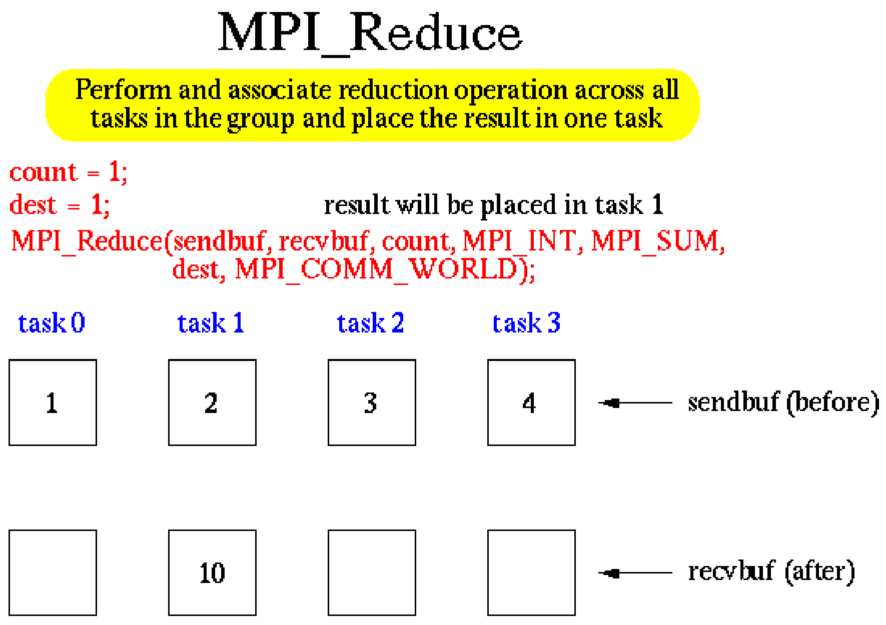


MPI_Allgather

Gathers together values from a group of processes and distributes to all

```
sendcnt = 1;
recvcnt = 1;
MPI_Allgather(sendbuf, sendcnt, MPI_INT,
              recvbuf, recvcnt, MPI_INT,
              MPI_COMM_WORLD);
```

Summary of collective communication

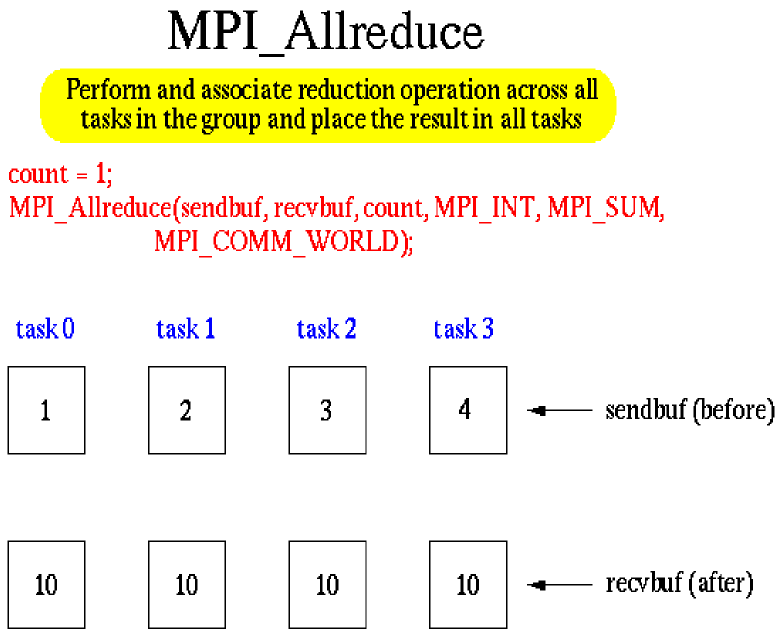


MPI_Reduce

Perform and associate reduction operation across all tasks in the group and place the result in one task

```
count = 1;
dest = 1;                    result will be placed in task 1
MPI_Reduce(sendbuf, recvbuf, count, MPI_INT, MPI_SUM,
               dest, MPI_COMM_WORLD);
```

task 0	task 1	task 2	task 3	
1	2	3	4	← sendbuf (before)
	10			← recvbuf (after)



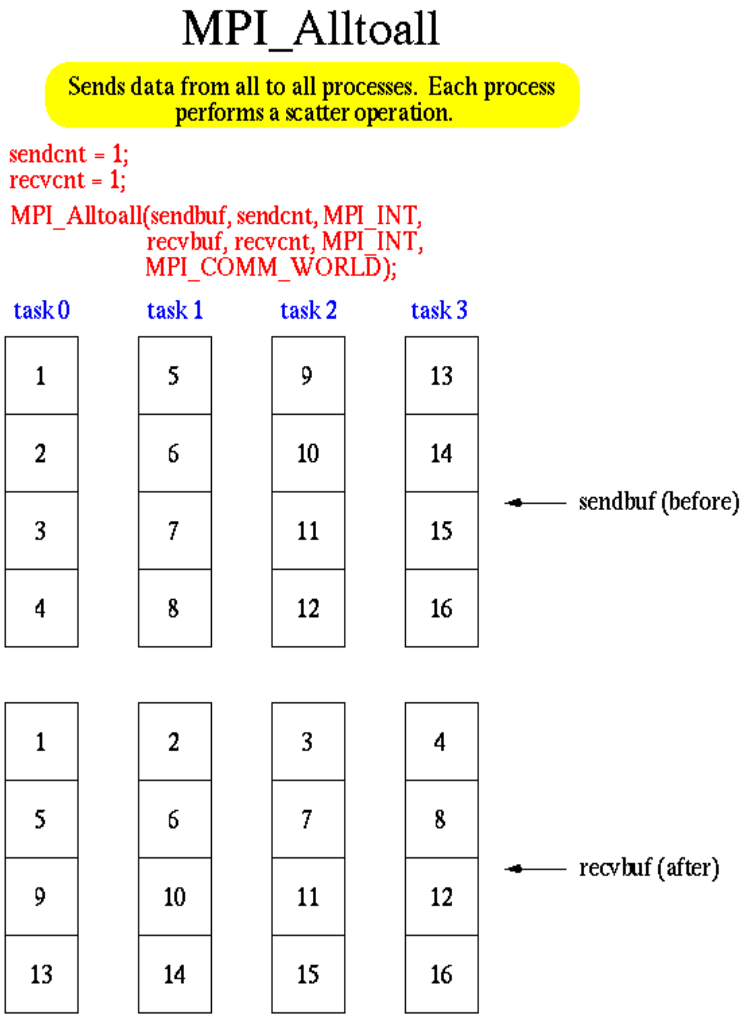
MPI_Allreduce

Perform and associate reduction operation across all tasks in the group and place the result in all tasks

```
count = 1;
MPI_Allreduce(sendbuf, recvbuf, count, MPI_INT, MPI_SUM,
              MPI_COMM_WORLD);
```

task 0	task 1	task 2	task 3	
1	2	3	4	← sendbuf (before)
10	10	10	10	← recvbuf (after)

Summary of collective communication



MPI_Alltoall

Sends data from all to all processes. Each process performs a scatter operation.

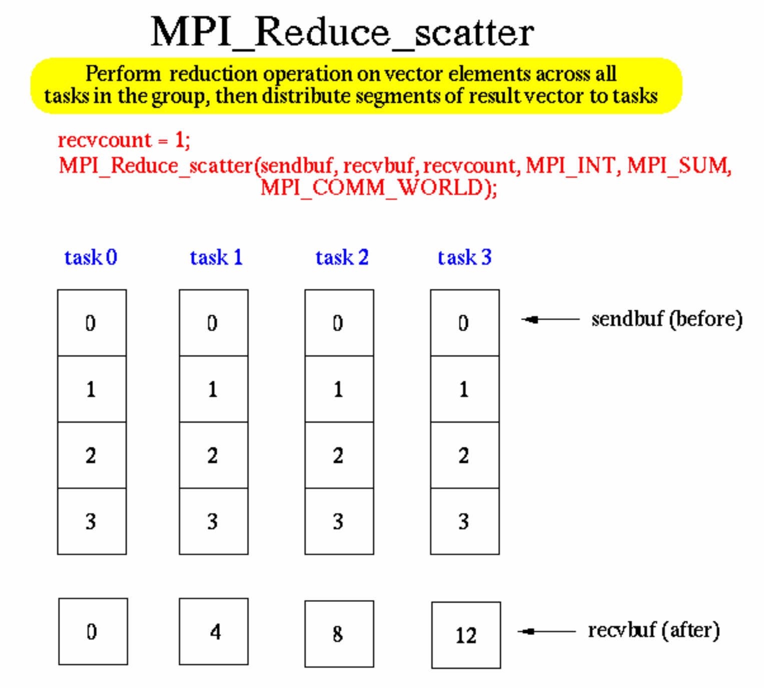
```
sendcnt = 1;
recvcnt = 1;
MPI_Alltoall(sendbuf, sendcnt, MPI_INT,
                     recvbuf, recvcnt, MPI_INT,
                     MPI_COMM_WORLD);
```

task 0	task 1	task 2	task 3
1	5	9	13
2	6	10	14
3	7	11	15
4	8	12	16

sendbuf (before)

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

recvbuf (after)



MPI_Reduce_scatter

Perform reduction operation on vector elements across all tasks in the group, then distribute segments of result vector to tasks

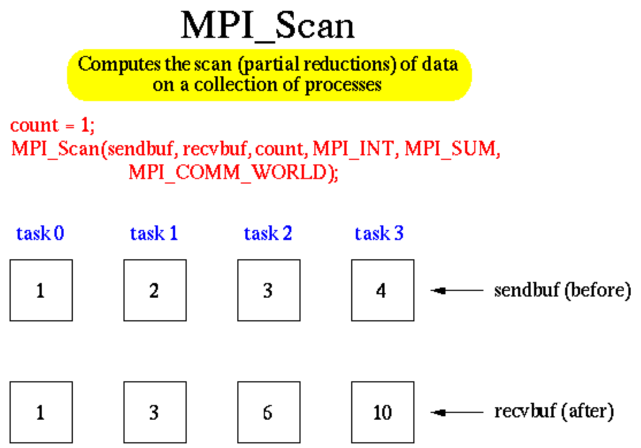
```
recvcount = 1;
MPI_Reduce_scatter(sendbuf, recvbuf, recvcount, MPI_INT, MPI_SUM,
                              MPI_COMM_WORLD);
```

task 0	task 1	task 2	task 3
0	0	0	0
1	1	1	1
2	2	2	2
3	3	3	3

sendbuf (before)

0	4	8	12

recvbuf (after)



MPI_Scan

Computes the scan (partial reductions) of data on a collection of processes

```
count = 1;
MPI_Scan(sendbuf, recvbuf, count, MPI_INT, MPI_SUM,
                 MPI_COMM_WORLD);
```

task 0	task 1	task 2	task 3
1	2	3	4

sendbuf (before)

1	3	6	10

recvbuf (after)

05 - Consecutive odd numbers which are prime

A prime number is a positive integer evenly divisible by exactly two positive integers: itself and 1. The firs five prime numbers are 2, 3, 5, 7, 11. Sometimes two consecutive odd numbers are both prime. For example, the odd integer following 3, 5, 11 are all prime numbers. However, the odd integer following 7 is not prime number. Write a parallel program to determine, for all integers less than 1,000,000 the number of times that two consecutive odd integers are both prime.

06 - *Goldbach's conjecture*

Goldbach's conjecture je jedan od najstarijih nerešenih problema u teoriji brojeva i u celoj matematici. 7. juna 1742. Nemački matematičar **Christian Goldbach** je postavio problem koji glasi ovako:

Svaki paran broj veći od 2 se može predstaviti kao zbir dva prosta broja.

Pokazati da je ovo tvrđenje tačno do broja 1 000 000 zaključno sa njim. Za to koristiti MPI program koji će se izvršavati na proizvoljnom broju procesa.