Bach Khoa University Of Technology
Faculty of Computer Science & Engineering

Course: Parallel Computing
Lab #4 - MPI Collective Communication

Thanh-Dang Diep

August 19, 2017

Goal: This lab helps students to get familiar with MPI collective communication which is another basic mechanism to write parallel programs running distributed memory systems.

Content: The first Section introduces all types of collective operations. The next Section explains collective communication routines. The last one is exercises to help practice the obtained knowledge.

Result: After finishing this lab, students are able to understand and write a basic parallel program running distributed memory systems by using MPI collective communication.

Contents

1 Introduction

There are three types of Collective Operations (as shown in Figure 1):

Synchronization: Processes wait until all members of the group have reached the synchronization point.

Data Movement: Broadcast, scatter/gather, all to all.

Collective Computation (reductions): One member of the group collects data from the other members and performs an operation (min, max, add, multiply, etc.) on that data.

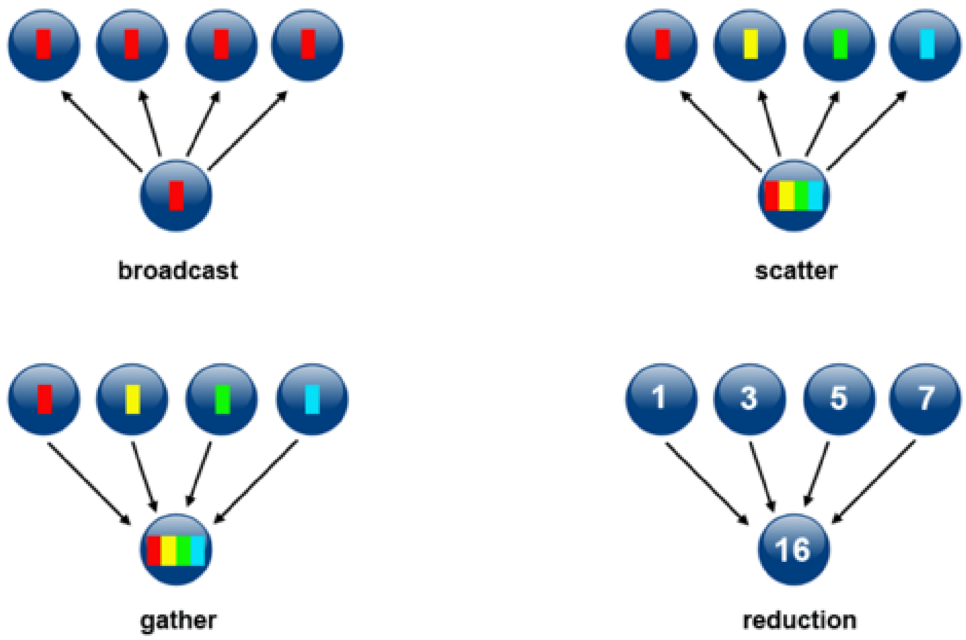

Figure 1: Basic collective operations

2 Collective Communication Routines

- **MPI_Barrier**

Synchronization operation. Creates a barrier synchronization in a group. Each process, when reaching the MPI_Barrier call, blocks until all processes in the group reach the same MPI_Barrier call. Then all processes are free to proceed.

```
MPI_Barrier(comm);
```

- **MPI_Bcast**

Data movement operation. Broadcasts (sends) a message from the process with rank "root" to all the other processes in the group (as shown in Figure 2).

```
MPI_Bcast(&buffer, count, datatype, root, comm);
```

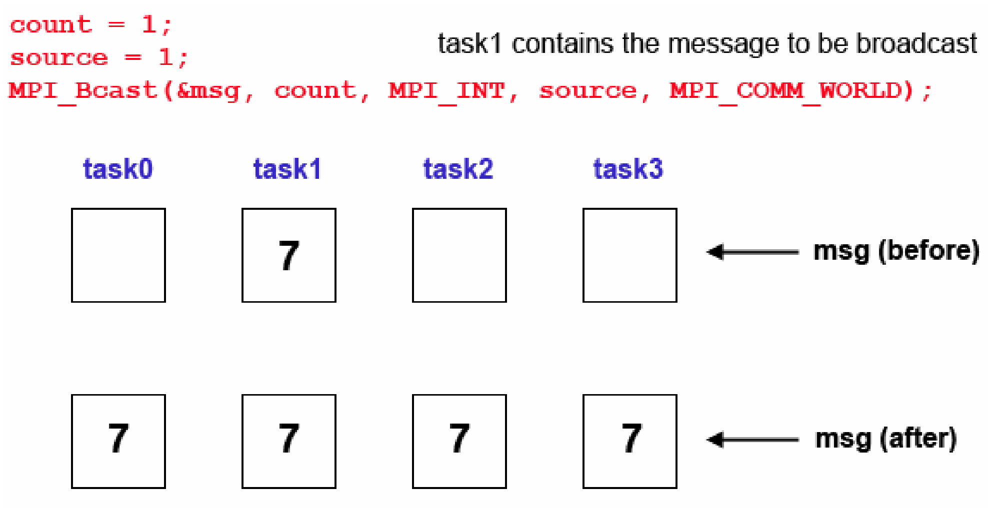

Figure 2: The diagram for MPI_Bcast

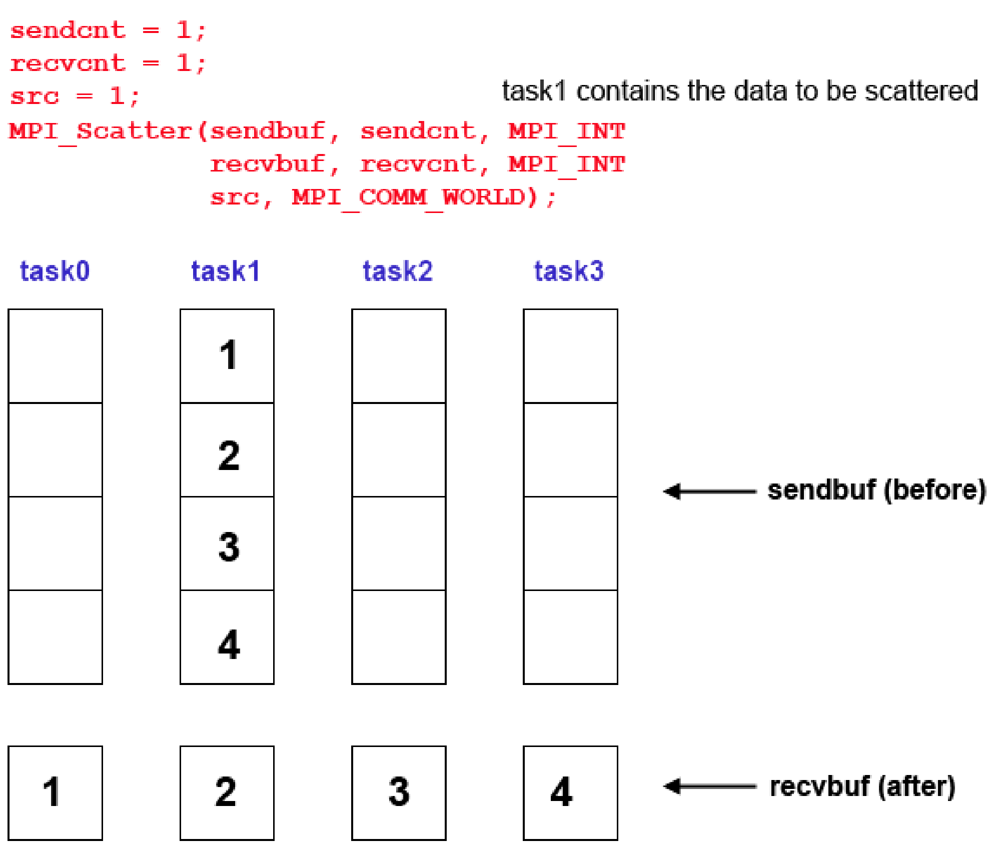

Figure 3: The diagram for MPI_Scatter

- **MPI_Scatter**

Data movement operation. Distributes distinct messages from a single source process to each process in the group (as shown in Figure 3).

```
MPI_Scatter(&sentbuf, sendcnt, sendtype, &recvbuf, recvcnt, recvtype, root, comm);
```

- **MPI_Gather**

Data movement operation. Gathers distinct messages from each process in the group to a single destination process. This routine is the reverse operation of MPI_Scatter (as shown in Figure 4).

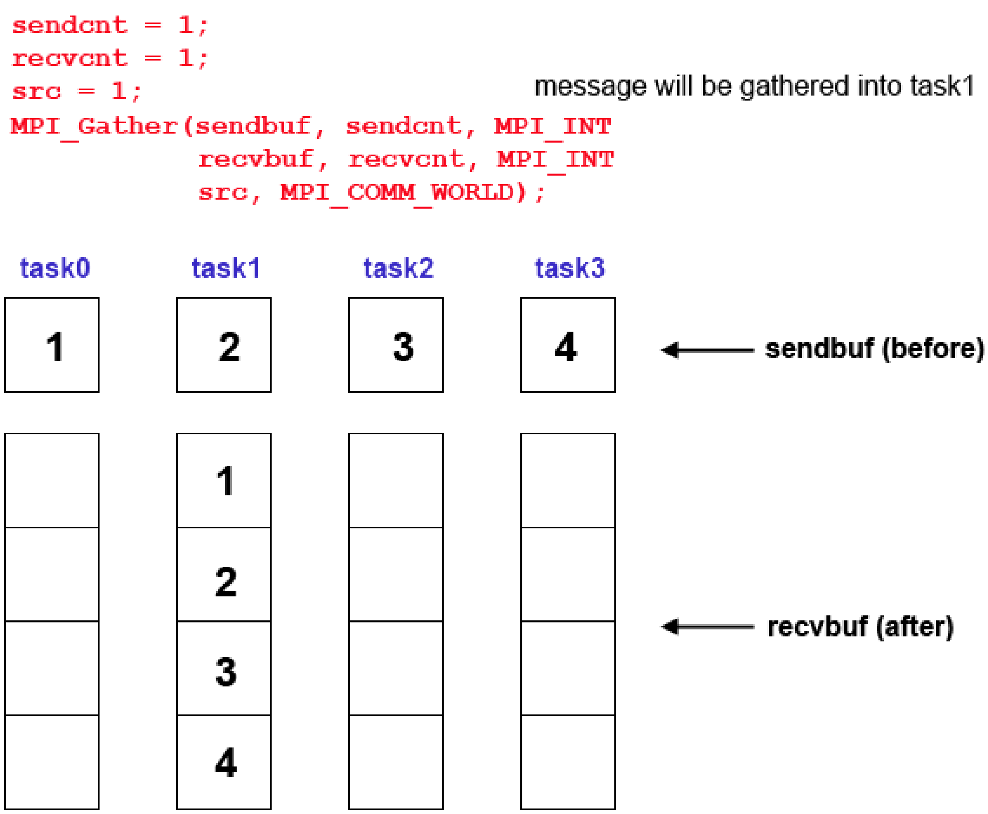

Figure 4: The diagram for MPI_Gather

```
MPI_Gather(&sendbuf, sendcnt, sendtype, &recvbuf, recvcnt, recvtype, root, comm);
```

- **MPI_Allgather**

Data movement operation. Concatenation of data to all processes in a group. Each process in the group, in effect, performs a one-to-all broadcasting operation within the group (as shown in Figure 5).

```
MPI_Allgather(&sendbuf, sendcnt, sendtype, &recvbuf, recvcnt, recvtype, comm);
```

- **MPI_Reduce**

Collective computation operation. Applies a reduction operation on all processes in the group and places the result in one process (as shown in Figure 6).

```
MPI_Reduce(&sendbuf, &recvbuf, count, datatype, op, root, comm);
```

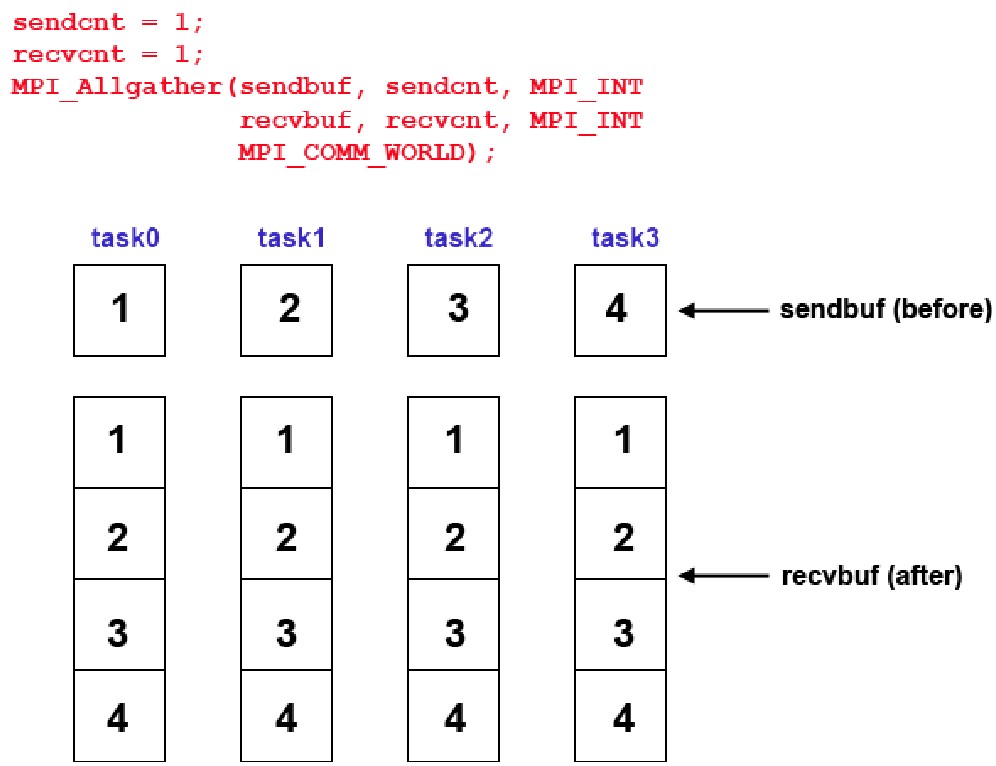

Figure 5: The diagram for MPI_Allgather

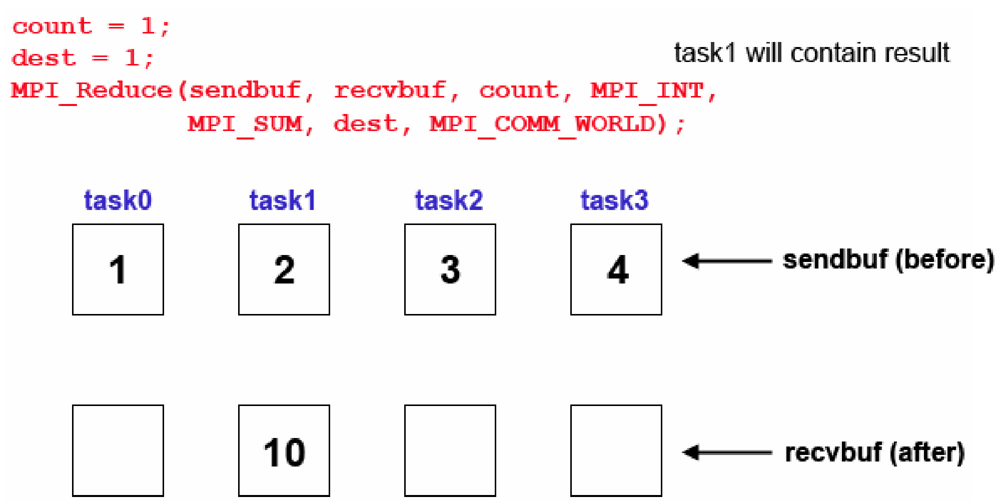

Figure 6: The diagram for MPI_Reduce

The predefined MPI reduction operations appear as shown in Figure 7. Users can also define their own reduction functions by using the MPI_Op_create routine.

- **MPI_Allreduce**

Collective computation operation + data movement. Applies a reduction operation and places the result in all tasks in the group. This is equivalent to an MPI_Reduce followed by an MPI_Bcast (as shown in Figure 8).

MPI Reduction Operation		C Data Types
MPI_MAX	maximum	integer, float
MPI_MIN	minimum	integer, float
MPI_SUM	sum	integer, float
MPI_PROD	product	integer, float
MPI_LAND	logical AND	integer
MPI_BAND	bit-wise AND	integer, MPI_BYTE
MPI_LOR	logical OR	integer
MPI_BOR	bit-wise OR	integer, MPI_BYTE
MPI_LXOR	logical XOR	integer
MPI_BXOR	bit-wise XOR	integer, MPI_BYTE
MPI_MAXLOC	max value and location	float, double and long double
MPI_MINLOC	min value and location	float, double and long double

Figure 7: The predefined MPI reduction operations

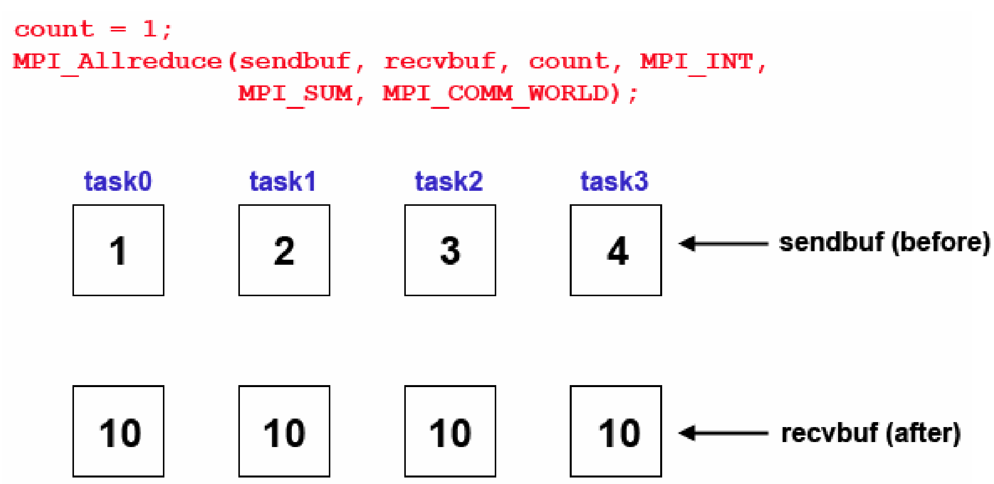

Figure 8: The diagram for MPI_Allreduce

```
MPI_Allreduce(&sendbuf, &recvbuf, count, datatype, op, comm);
```

Example: Collective Communications
Perform a scatter operation on the rows of an array

```
#include <mpi.h>
#include <stdio.h>
#define SIZE 4

int main(int argc, char **argv)
{
     int numtasks, rank, sendcount, recvcount, source;
     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, &numtasks);

    if (numtasks == SIZE)
    {
        /* define source task and elements to send/receive, then perform collective
            scatter */
        source = 1;
        sendcount = SIZE;
        recvcount = SIZE;
        MPI_Scatter(sendbuf, sendcount, MPI_FLOAT, recvbuf, recvcount, MPI_FLOAT,
            source, MPI_COMM_WORLD);

        printf("rank = %d  Results: %f %f %f %f\n", rank, recvbuf[0], recvbuf[1],
            recvbuf[2], recvbuf[3]);
    }
    else printf("Must specify %d processors. Terminating.\n", SIZE);

    MPI_Finalize();
    return 0;
}
```

3 Exercises

1. Write a program that computes the sum 1+2+...+p in the following manner: Each process i assigns the value i+1 to an integer and then the processes perform a sum reduction of these values. Process 0 should print the result of the reduction. As a way of double-checking the result, process 0 should also compute and print the value p(p+1)/2.

2. A small college wishes to assign unique identification numbers to all of its present and future students. The administration is thinking of using a six-digit identifier, but is not sure that there will be enough combinations, given various constraints that have been placed on what is considered to be an "acceptable" identifier. Write a parallel program to count the number of different six-digit combinations of the numerals 0-9, given these constraints:
   - The first digit may not be a 0.
   - Two consecutive digits may not be the same.
   - The sum of the digits may not be 7, 11, or 13.

3. Write a program that simulates the N-body problem.