MPI_Scatterv

Definition

MPI_Scatterv is a version of MPI_Scatter in which the data dispatched from the root process can vary in the number of elements, and the location from which load these elements in the root process buffer. Also, MPI_Scatterv is a collective operation; all processes in the communicator must invoke this routine. Other variants of MPI_Scatterv are MPI_Scatter, MPI_Iscatter and MPI_Iscatterv. Refer to MPI_Iscatterv to see the blocking counterpart of MPI_Scatterv.

Copy Feedback

```
001. int MPI_Scatterv(const void* buffer_send,
002.                  const int counts_send[],
003.                  const int displacements[],
004.                  MPI_Datatype datatype_send,
005.                  void* buffer_recv,
006.                  int count_recv,
007.                  MPI_Datatype datatype_recv,
008.                  int root,
009.                  MPI_Comm communicator);
```

Parameters

buffer_send

The buffer containing the data to disptach from the root process. For non-root processes, the send parameters like this one are ignored.

counts_send

An array that contains the number of elements to send to each process, not the total number of elements in the send buffer. For non-root processes, the send parameters like this one are ignored.

displacements

An array containing the displacement to apply to the message sent to each process. Displacements are expressed in number of elements, not bytes. For non-root processes, the sending parameters like this one are ignored.

datatype_send

The type of one send buffer element. For non-root processes, the send parameters like this one are ignored.

buffer_recv

The buffer in which store the data dispatched.

count_recv

The number of elements in the receive buffer.

datatype_recv

The type of one receive buffer element.

root

The rank of the root process, which will dispatch the data to scatter.

communicator

The communicator in which the scatter takes place.

Return value

The error code returned from the variable scatter.

- MPI_SUCCESS: the routine successfully completed.

Example

Copy Feedback

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

/**
 * @brief Illustrates how to use the variable versi
 * @details A process is designed as root and begin
 * values, and prints them. It then dispatches thes
 * in the same communicator. Other process just rec
 * meant for them. Finally, everybody prints the va
 * application is designed to cover all cases:
 * - Different send counts
 * - Different displacements
 * This application is meant to be run with 3 proce
 *
 *       +-------------------------------------
 *       |               Process 0
 *       +-----+-----+-----+-----+-----+-----+-----
 *       | 100 |  0  | 101 | 102 |  0  |  0  | 103
 *       +-----+-----+-----+-----+-----+-----+-----
 *         |           |     |               |
 *         |           |     |               |
 *         |           |     |               |
 *         |           |     |               |
 *         |           |     |               |
 *         |           |     |               |
 * +-----------+ +------------------+ +-----------
 * | Process 0 | |    Process 1     | | Process 2
 * +-+-------+-+ +-+-------+-------+-+ +-+-------+-
 *   | Value |     | Value | Value |     | Value |
```

```
 *   | value |     | value | value |     | value |
 *   |  100  |     |  101  |  102  |     |  103  |
 *   +-------+     +-------+-------+     +-------+
 *
 **/
int main(int argc, char* argv[])
{
    MPI_Init(&argc, &argv);

    // Get number of processes and check that 3 pro
    int size;
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    if(size != 3)
    {
        printf("This application is meant to be run
        MPI_Abort(MPI_COMM_WORLD, EXIT_FAILURE);
    }

    // Determine root's rank
    int root_rank = 0;

    // Get my rank
    int my_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

    switch(my_rank)
    {
        case 0:
        {
            // Define my value
            int my_value;

            // Declare the buffer
            int buffer[7] = {100, 0, 101, 102, 0, 0

            // Declare the counts
            int counts[3] = {1, 2, 1};

            // Declare the displacements
            int displacements[3] = {0, 2, 6};

            printf("Values in the buffer of root pr
            for(int i = 0; i < 7; i++)
            {
                printf(" %d", buffer[i]);
            }
            printf("\n");
            MPI_Scatterv(buffer, counts, displaceme
            printf("Process %d received value %d.\n
            break;
```

```
080.         }
081.         case 1:
082.         {
083.             // Declare my values
084.             int my_values[2];
085.
086.             MPI_Scatterv(NULL, NULL, NULL, MPI_INT,
087.             printf("Process %d received values %d a
088.             break;
089.         }
090.         case 2:
091.         {
092.             // Declare my values
093.             int my_value;
094.
095.             MPI_Scatterv(NULL, NULL, NULL, MPI_INT,
096.             printf("Process %d received value %d.\n
097.             break;
098.         }
099.     }
100.
101.     MPI_Finalize();
102.
103.     return EXIT_SUCCESS;
104. }
```