

[Homepage](#) ▶ [MPI](#) ▶ [Docs](#) ▶ [MPI_Scatterv](#)[About](#)[Docs](#)[Tools](#)[Exercises](#)[Collectives](#)[✓ C](#)[Fortran-2008](#)[Fortran-90](#)

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 dispatch 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

```

001. #include <stdio.h>
002. #include <stdlib.h>
003. #include <mpi.h>
004.
005. /**
006.  * @brief Illustrates how to use the variable version of MPI_Scatterv.
007.  * @details A process is designed as root and begins by sending values, and prints them. It then dispatches these values in the same communicator. Other process just receive the values meant for them. Finally, everybody prints the values. This application is designed to cover all cases:
008.  * - Different send counts
009.  * - Different displacements
010.  * This application is meant to be run with 3 processes.
011.  */
012.
013. #define N 100
014. #define N2 101
015. #define N3 102
016. #define N4 103
017.
018. int main(int argc, char** argv)
019. {
020.     MPI_Init(&argc, &argv);
021.     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
022.     MPI_Comm_size(MPI_COMM_WORLD, &size);
023.
024.     if (rank == 0)
025.     {
026.         int* data = (int*)malloc(N * sizeof(int));
027.         for (int i = 0; i < N; i++)
028.             data[i] = i;
029.
030.         MPI_Scatterv(data, [N, 0, 0, 0], MPI_INT, MPI_COMM_WORLD, data, [N, 0, 0, 0], MPI_INT);
031.         printf("Process 0: %d\n", data[0]);
032.     }
033.     else if (rank == 1)
034.     {
035.         int* data = (int*)malloc(N2 * sizeof(int));
036.         for (int i = 0; i < N2; i++)
037.             data[i] = i + N;
038.         MPI_Scatterv(data, [N2, 0, 0, 0], MPI_INT, MPI_COMM_WORLD, data, [N2, 0, 0, 0], MPI_INT);
039.         printf("Process 1: %d\n", data[0]);
040.     }
041.     else if (rank == 2)
042.     {
043.         int* data = (int*)malloc(N3 * sizeof(int));
044.         for (int i = 0; i < N3; i++)
045.             data[i] = i + N + N2;
046.         MPI_Scatterv(data, [N3, 0, 0, 0], MPI_INT, MPI_COMM_WORLD, data, [N3, 0, 0, 0], MPI_INT);
047.         printf("Process 2: %d\n", data[0]);
048.     }
049.     else if (rank == 3)
050.     {
051.         int* data = (int*)malloc(N4 * sizeof(int));
052.         for (int i = 0; i < N4; i++)
053.             data[i] = i + N + N2 + N3;
054.         MPI_Scatterv(data, [N4, 0, 0, 0], MPI_INT, MPI_COMM_WORLD, data, [N4, 0, 0, 0], MPI_INT);
055.         printf("Process 3: %d\n", data[0]);
056.     }
057.
058.     MPI_Finalize();
059.     return 0;
060. }
  
```

```

030.      / value /      / value / value /      / value /
031.  * | 100 |      | 101 | 102 |      | 103 |
032.  * +-----+      +-----+-----+      +-----+
033.  *
034.  **/
035. int main(int argc, char* argv[])
036. {
037.     MPI_Init(&argc, &argv);
038.
039.     // Get number of processes and check that 3 pro
040.     int size;
041.     MPI_Comm_size(MPI_COMM_WORLD, &size);
042.     if(size != 3)
043.     {
044.         printf("This application is meant to be run
045.         MPI_Abort(MPI_COMM_WORLD, EXIT_FAILURE);
046.     }
047.
048.     // Determine root's rank
049.     int root_rank = 0;
050.
051.     // Get my rank
052.     int my_rank;
053.     MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
054.
055.     switch(my_rank)
056.     {
057.         case 0:
058.         {
059.             // Define my value
060.             int my_value;
061.
062.             // Declare the buffer
063.             int buffer[7] = {100, 0, 101, 102, 0, 0
064.
065.             // Declare the counts
066.             int counts[3] = {1, 2, 1};
067.
068.             // Declare the displacements
069.             int displacements[3] = {0, 2, 6};
070.
071.             printf("Values in the buffer of root pr
072.             for(int i = 0; i < 7; i++)
073.             {
074.                 printf(" %d", buffer[i]);
075.             }
076.             printf("\n");
077.             MPI_Scatterv(buffer, counts, displaceme
078.             printf("Process %d received value %d.\n
079.             break:

```

```
077.         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. }
```

[Site Map](#)[Privacy Policy](#)[Contact](#)

©2019-2021 RookieHPC.