```c
#include <stdlib.h>
#include <stdio.h>
#include <mpi.h>
int main (int argc, char * argv[])
{
  int rank, size;
  MPI_Init (&argc, &argv);
  MPI_Comm_rank (MPI_COMM_WORLD, &rank);
  MPI_Comm_size (MPI_COMM_WORLD, &size);
  printf ("I am process %d of %d\n", rank, size);
  MPI_Finalize();
  return EXIT_SUCCESS;
}
```



```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <mpi.h>

int main (int argc, char * argv[])
{
  int rank, size, src, dest, nc;
  int tag = 50;  // tag for messages
  MPI_Status status;
  char message[100];
  MPI_Init (&argc, &argv);
  MPI_Comm_rank (MPI_COMM_WORLD, &rank);
  MPI_Comm_size (MPI_COMM_WORLD, &size);

  if (rank != 0) {
    sprintf (message, "Hello from process %d.", rank);
    dest = 0;
    nc = strlen (message) + 1; // to include the trailing null
    MPI_Send (message, nc, MPI_CHAR, dest, tag, MPI_COMM_WORLD);
  } else {
    for (src = 1; src < size; src++) {
      MPI_Recv (message, 100, MPI_CHAR, src, tag, MPI_COMM_WORLD, &status);
      printf ("%s\n", message);
    }
  }

  MPI_Finalize();
  return EXIT_SUCCESS;
}
```

```c
// ------------------------------------------------------------------------
// Compute pi by integrating f(x) = 4 / (1 + x**2) -- serial version
// ------------------------------------------------------------------------
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define PI 3.14159265358979323846264338327950291

double f (double a)
{
  return (double)4.0 / ((double)1.0 + (a * a));
}

int main (int argc, char *argv[])
{
  unsigned long n, i;
  double h, pi, sum, x;
  n = atoi (argv[1]);
  h = (double)1.0 / (double)n;
  sum = 0.0;
  for (i = 1; i <= n; i++) {
    x = h * ((double)i - (double)0.5);
    sum += f(x);
  }
  pi = h * sum;
  printf ("pi is approximately: %.16f; error is: %.16f\n", pi, fabs(pi-PI));
  return EXIT_SUCCESS;
}
```



```c
// ------------------------------------------------------------------------
// Compute pi by integrating f(x) = 4 / (1 + x**2) -- MPI version
// ------------------------------------------------------------------------
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <mpi.h>
#define PI 3.14159265358979323846264338327950291
#define MTAG1 1
#define MTAG2 2

double f (double a)
{
  return (double)4.0 / ((double)1.0 + a * a);
}

int main(int argc, char *argv[])
{
  double mypi, pi, h, sum, x;
  MPI_Status status;
  int n, myid, numprocs, i, islave;

  MPI_Init (&argc, &argv);
  MPI_Comm_size (MPI_COMM_WORLD, &numprocs);
  MPI_Comm_rank (MPI_COMM_WORLD, &myid);
  if (myid == 0) {
    n = atoi (argv[1]);
    for (islave = 1; islave < numprocs; islave++) {
      MPI_Send (&n, 1, MPI_INT, islave, MTAG1, MPI_COMM_WORLD);
    }
  } else {
    MPI_Recv (&n, 1, MPI_INT, 0, MTAG1, MPI_COMM_WORLD, &status);
  }

  h = 1.0 / (double) n;
  sum = 0.0;
  for (i = myid + 1 ; i <= n; i += numprocs) {
    x = h * ((double)i - 0.5);
    sum += f (x);
  }
  mypi = h * sum;

  if (myid != 0) {
    MPI_Send (&mypi, 1, MPI_DOUBLE, 0, MTAG2, MPI_COMM_WORLD);
  } else {
    pi = mypi;
    for (islave = 1; islave < numprocs; islave++) {
      MPI_Recv (&mypi, 1, MPI_DOUBLE, islave, MTAG2, MPI_COMM_WORLD, &status);
      pi += mypi;
    }
    printf ("pi is approximately: %.16f; error is: %.16f\n", pi, fabs(pi-PI));
  }

  MPI_Finalize ();
  return EXIT_SUCCESS;
}
```

```c
// ------------------------------------------------------------------------
// Compute pi by integrating f(x) = 4 / (1 + x**2) -- MPI version using Bcast
// ------------------------------------------------------------------------
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <mpi.h>
#define PI 3.14159265358979323846264338327950 29L
#define MTAG1 1
#define MTAG2 2

double f (double a)
{
  return (double)4.0 / ((double)1.0 + a * a);
}

int main(int argc, char *argv[])
{
  double mypi, pi, h, sum, x;
  MPI_Status status;
  int n = 0, myid, numprocs, i, islave, err;

  MPI_Init (&argc, &argv);
  MPI_Comm_size (MPI_COMM_WORLD, &numprocs);
  MPI_Comm_rank (MPI_COMM_WORLD, &myid);
  if (myid == 0) n = atoi (argv[1]);
  err = MPI_Bcast (&n, 1, MPI_INT, 0, MPI_COMM_WORLD);

  h = 1.0 / (double) n;
  sum = 0.0;
  for (i = myid + 1 ; i <= n; i += numprocs) {
    x = h * ((double)i - 0.5);
    sum += f (x);
  }
  mypi = h * sum;

  if (myid != 0) {
    MPI_Send (&mypi, 1, MPI_DOUBLE, 0, MTAG2, MPI_COMM_WORLD);
  } else {
    pi = mypi;
    for (islave = 1; islave < numprocs; islave++) {
      MPI_Recv (&mypi, 1, MPI_DOUBLE, islave, MTAG2, MPI_COMM_WORLD, &status);
      pi += mypi;
    }
    printf ("pi is approximately: %.16f; error is: %.16f\n", pi, fabs(pi-PI));
  }

  MPI_Finalize ();
  return EXIT_SUCCESS;
}
```



```c
// ------------------------------------------------------------------------
// A two-dimensional torus of 12 processes in a 4 x 3 grid
// ------------------------------------------------------------------------
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
  int rank, size;
  MPI_Comm comm;
  int dim[2], period[2], reorder;
  int coord[2], id;

  MPI_Init (&argc, &argv);
  MPI_Comm_rank (MPI_COMM_WORLD, &rank);
  MPI_Comm_size (MPI_COMM_WORLD, &size);
  if (size != 12) {
    fprintf (stderr, "Please run with 12 processes.\n");
    fflush (stdout);
    MPI_Abort (MPI_COMM_WORLD, 1);
  }

  dim[0] = 4; dim[1] = 3;
  period[0] = 1; period[1] = 0;
  reorder = 1;
  MPI_Cart_create (MPI_COMM_WORLD, 2, dim, period, reorder, &comm);
  if (rank == 5) {
    MPI_Cart_coords (comm, rank, 2, coord);
    printf ("Rank %d coordinates are %d %d\n", rank, coord[0], coord[1]);
    fflush (stdout);
  }
  if (rank == 0) {
    coord[0] = 3; coord[1] = 1;
    MPI_Cart_rank (comm, coord, &id);
    printf ("The processor at position (%d, %d) has rank %d\n",
            coord[0], coord[1], id);
    fflush (stdout);
  }
  MPI_Finalize();
  return EXIT_SUCCESS;
}
```

```
// ------------------------------------------------------------------------------
// Using MPI_Scatter and MPI_Gather
// ------------------------------------------------------------------------------
// Adapted from https://rqchp.ca/modules/cms/checkFileAccess.php?file=local.rqchpweb_udes
/mpi/exemples_c/ex07_Scatter_EN.c
// ------------------------------------------------------------------------------
/* This example uses MPI_Scatter to distribute data to all available tasks.

                <----- sendbuff ------>

          ###################################
          #      #      #      #      #      #
       0  #  AA  #  BB  #  CC  #  DD  #  EE  #
          #      #      #      #      #      #
          ###################################
    T     #      #      #      #      #      #
       1  #      #      #      #      #      #
    a     #      #      #      #      #      #
          ###################################
    s     #      #      #      #      #      #
       2  #      #      #      #      #      #        BEFORE
    k     #      #      #      #      #      #
          ###################################
    s     #      #      #      #      #      #
       3  #      #      #      #      #      #
          #      #      #      #      #      #
          ###################################
          #      #      #      #      #      #
       4  #      #      #      #      #      #
          #      #      #      #      #      #
          ###################################

          recvbuff

          ########
          #      #
       0  #  AA  #
          #      #
          ########
    T     #      #
       1  #  BB  #
    a     #      #
          ########
    s     #      #
       2  #  CC  #                                  AFTER
    k     #      #
          ########
    s     #      #
       3  #  DD  #
          #      #
          ########
          #      #
       4  #  EE  #
          #      #
          ########

   Task 0 prepares one vector of real random numbers per task, and
   place them contiguously in memory in the vector sendbuff. Then
   each of the vectors are sent to their corresponding task using
   MPI_Scatter.

   Before and after communication the sum for each vector is
   printed out for verification.

   The size of the vector (buffsize) is given as an argument to
   the program at run time.

   This example does exactly the same work as example 4 but
```



```
   does it using collective communication. You can compare
   the performance! You'll see that it's much more efficient to
   use collective communication when ever possible.

 Author: Carol Gauthier, Centre de Calcul scientifique,
         Universite de Sherbrooke */
// ------------------------------------------------------------------------------
#include <malloc.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <math.h>
#include <mpi.h>

int main (int argc,char** argv)
{
   int          taskid, ntasks;
   MPI_Status   status;
   int          ierr, i, j, itask;
   int          buffsize;
   double       **sendbuff, *recvbuff, buffsum, buffsums[1024];
   double       inittime, totaltime, recvtime, recvtimes[1024];

   // MPI Initialisation. It's important to put this call at the
   // begining of the program, after variable declarations.
   MPI_Init (&argc, &argv);

   // Get the number of MPI tasks and the taskid of this task.
   MPI_Comm_rank (MPI_COMM_WORLD, &taskid);
   MPI_Comm_size (MPI_COMM_WORLD, &ntasks);

   // Get buffsize value from program arguments.
   buffsize = atoi (argv[1]);

   // Print out the description of the example.
   if (taskid == 0) {
     printf ("############################################################\n\n");
     printf (" Collective Communication : MPI_Scatter \n\n");
     printf (" Vector size: %d\n",buffsize);
     printf (" Number of tasks: %d\n\n",ntasks);
     printf ("############################################################\n\n");
     printf ("                --> BEFORE COMMUNICATION <--\n\n");
   }

   // Memory allocation.
   recvbuff = (double *)malloc (sizeof(double) * buffsize);
   if (taskid == 0) {
     sendbuff = (double **)malloc (sizeof(double *) * ntasks);
     sendbuff[0] = (double *)malloc (sizeof(double) * ntasks * buffsize);
     for (i = 1; i < ntasks; i++)
       sendbuff[i] = sendbuff[i-1] + buffsize;
   } else {
     sendbuff = (double **)malloc (sizeof(double *) * 1);
     sendbuff[0] = (double *)malloc (sizeof(double) * 1);
   }

   if (taskid == 0) {
     // Vectors and/or matrices initialisation.
     srand ((unsigned)time (NULL) + taskid);
     for (itask = 0; itask < ntasks; itask++) {
       for (i = 0; i < buffsize; i++) {
         sendbuff[itask][i] = (double)rand() / RAND_MAX;
       }
     }

     // Print out before communication.
     for (itask = 1; itask < ntasks; itask++) {
       buffsum = 0.0;
```

```
      for (i = 0; i < buffsize; i++) {
        buffsum = buffsum + sendbuff[itask][i];
      }
      printf ("Task %d : Sum of vector sent to %d -> %e \n",
              taskid,itask,buffsum);
    }
    printf ("\n");
  }

  // Communication.
  inittime = MPI_Wtime ();
  ierr = MPI_Scatter (sendbuff[0], buffsize, MPI_DOUBLE, recvbuff, buffsize,
                      MPI_DOUBLE, 0, MPI_COMM_WORLD);
  totaltime = MPI_Wtime() - inittime;

  // Print out after communication.
  buffsum = 0.0;
  for (i = 0; i < buffsize; i++)
    buffsum = buffsum + recvbuff[i];
  ierr = MPI_Gather (&buffsum, 1, MPI_DOUBLE, buffsums, 1, MPI_DOUBLE,
                  0, MPI_COMM_WORLD);

  if (taskid == 0) {
    printf ("############################################################\n\n");
    printf ("                --> AFTER COMMUNICATION <-- \n\n");
    for (itask = 1; itask < ntasks; itask++) {
      printf ("Task %d : Sum of received vector= %e\n",
              itask, buffsums[itask]);
    }
    printf ("\n");
    printf ("############################################################\n\n");
    printf (" Communication time : %f seconds\n\n",totaltime);
    printf ("############################################################\n\n");
  }

  // Free the allocated memory.
  if (taskid == 0) {
    free (sendbuff[0]);
    free (sendbuff);
  } else {
    free (sendbuff[0]);
    free (sendbuff);
    free (recvbuff);
  }
  MPI_Finalize();
}
```