BCAST

```
#include <stdio.h>
#include <mpi.h>
int main()
{
          int rank, value;
          value=0;
   MPI_Init(NULL, NULL);
   MPI_Comm_rank(MPI_COMM_WORLD, &rank); //what rank is the current processor

   if (rank == 3)
  {
       value=100;
      printf("100 is a number to be broadcast:\n");
   }
printf("process %d: Before MPI_Bcast, value is %d\n", rank, value);

      MPI_Bcast(&value, 1, MPI_INT, 0, MPI_COMM_WORLD);

printf("process %d: After MPI_Bcast, value is %d\n", rank, value);
MPI_Finalize();
return 0;
}
```

SCATTER

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

int main()
{
          int rank, size;
          int data=0;
          int *buff=NULL;

   MPI_Init( NULL, NULL);
   MPI_Comm_rank( MPI_COMM_WORLD, &rank );//0 1 2 3
   MPI_Comm_size( MPI_COMM_WORLD, &size );//4

   if(rank == 0)
          {
     int a[5]={10,20,30,40,50};
          buff=a;
   }
printf("Data before scatter in process %d is:%d\n",rank,data);
MPI_Scatter(buff,1,MPI_INT,&data,1,MPI_INT,0,MPI_COMM_WORLD);
printf("Data after scatter in process %d is:%d\n",rank,data);
```

```
MPI_Finalize();

}
```

GATHER

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

int main( int argc, char **argv )
{
   int isend;
   int rank, size;
 //int irecv[4];
int *buff;

   MPI_Init( NULL, NULL);
   MPI_Comm_rank( MPI_COMM_WORLD, &rank );//0 1 2 3
   MPI_Comm_size( MPI_COMM_WORLD, &size );//4

   isend = rank * 2;//0 2 4 6
 if(rank==1)
        buff=(int*)malloc(sizeof(int)*size);

   MPI_Gather(&isend, 1, MPI_INT, buff, 1, MPI_INT, 1, MPI_COMM_WORLD);

   if(rank == 1)
        for(int i=0;i<size;i++)
                printf("buffer has= %d \n", buff[i]);

   MPI_Finalize();

}
```

REDUCE

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

/* Define length of dot product vectors */
#define VECLEN 10

int main (int argc, char* argv[])
```

```
{
int i,myid, numprocs, len=VECLEN;
double *a, *b;
double mysum, allsum;

/* MPI Initialization */
MPI_Init (NULL, NULL);
MPI_Comm_size (MPI_COMM_WORLD, &numprocs);//5
MPI_Comm_rank (MPI_COMM_WORLD, &myid);//0 1 2 3 4

/*
  Each MPI task performs the dot product, obtains its partial sum, and then calls
  MPI_Reduce to obtain the global sum.
*/
if (myid == 0)
  printf("Starting dotprod_mpi. Using %d tasks...\n",numprocs);//5

/* Assign storage for dot product vectors */
a = (double*) malloc (len*sizeof(double));
b = (double*) malloc (len*sizeof(double));

/* Initialize dot product vectors */
for (i=0; i<len; i++) {
  a[i]=i;
  b[i]=i;
  }

/* Perform the dot product */
mysum = 0.0;
for (i=0; i<len; i++)
  {
    mysum += a[i] *b[i];//0 1 2 3 4
  }

printf("Task %d partial sum = %f\n",myid, mysum);

/* After the dot product, perform a summation of results on each node */
MPI_Reduce (&mysum, &allsum, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
if (myid == 0)
  printf ("Done. MPI version: global sum  =  %f \n", allsum);

free (a);
free (b);
MPI_Finalize();
}
```