MPI Programs

P2p.c

```
#include <stdio.h>
#include <mpi.h>
#include <string.h>
#define BUFFER_SIZE 32

int main(int argc,char *argv[])
{
	int  MyRank,Numprocs, Destination, iproc;
	int  tag = 0;
	int  Root = 0, temp = 1;
	char Message[BUFFER_SIZE];
	MPI_Init(&argc,&argv);
	MPI_Status status;

	MPI_Comm_rank(MPI_COMM_WORLD,&MyRank);
	MPI_Comm_size(MPI_COMM_WORLD,&Numprocs);

	/* print host name, and send message from process with rank 0 to all other
processes */
	if(MyRank == 0)		{
		system("hostname");
		strcpy(Message, "Hello India");
		for (temp=1; temp<Numprocs;temp++)
		{
			MPI_Send(Message, BUFFER_SIZE, MPI_CHAR, temp, tag,MPI_COMM_WORLD);
		}
	}
	else {
		system("hostname");
		MPI_Recv(Message, BUFFER_SIZE, MPI_CHAR, Root, tag,MPI_COMM_WORLD,
&status);
		printf("\n%s in process with rank %d from Process with rank %d\n",
Message,MyRank,Root);
	}

	MPI_Finalize();
}
```

P2p sum.c

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

int main(int argc,char *argv[])
{
    int iproc;
    int MyRank, Numprocs, Root = 0;
    int value, sum = 0;
```

```
    int       Source, Source_tag;
    int Destination, Destination_tag;
    MPI_Status status;


    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&Numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&MyRank);

    if(MyRank == Root){

       for(iproc = 1 ; iproc < Numprocs ; iproc++){
           Source     = iproc;
          Source_tag = 0;

          MPI_Recv(&value, 1, MPI_INT, Source, Source_tag,
                   MPI_COMM_WORLD, &status);
          sum = sum + value;
       }
       printf("MyRank = %d, SUM = %d\n", MyRank, sum);
    }
    else{
       Destination     = 0;
       Destination_tag = 0;

       MPI_Send(&MyRank, 1, MPI_INT, Destination, Destination_tag,
                MPI_COMM_WORLD);
    }

    MPI_Finalize();

}
```

Broadcast

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

int main (int argc, char *argv[])
{
        int rank, i;

         MPI_Init (&argc, &argv);

   MPI_Comm_rank (MPI_COMM_WORLD, &rank);


   if (rank == 0) i = 27;

   MPI_Bcast ((void *)&i, 1, MPI_INT, 0, MPI_COMM_WORLD);

   printf ("[%d] i = %d\n", rank, i);
```

```
   // Wait for every process to reach this code


   MPI_Barrier (MPI_COMM_WORLD);

   MPI_Finalize();


   return 0;


}
```

Gather.c

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

  void main(int argc, char *argv[])
  {
       int rank,size;
       double param[6],mine;
       int sndcnt,rcvcnt;
       int i;

       MPI_Init(&argc, &argv);
       MPI_Comm_rank(MPI_COMM_WORLD,&rank);
       MPI_Comm_size(MPI_COMM_WORLD,&size);

       sndcnt=1;
       mine=23.0+rank;
       if(rank==3) rcvcnt=1;

       MPI_Gather(&mine,sndcnt,MPI_DOUBLE,param,rcvcnt,MPI_DOUBLE,3,MPI_COMM_WORLD);

       if(rank==3)
              for(i=0;i<size;++i)
                //printf("PE:%d param[%d] is %f \n",rank,i,param[i]]);
                printf(" %d %d  \n",rank,i);

       MPI_Finalize();
}
```

Pie collective.c

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

double func(double x)
{
    return (4.0 / (1.0 + x*x));
```

```
}

int main(int argc,char *argv[])
{
    int    NoInterval, interval;
    int    MyRank, Numprocs, Root = 0;
    double mypi, pi, h, sum, x;
    double PI25DT = 3.14159265358979238462643;

    /*....MPI initialisation....*/
    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&Numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&MyRank);

    if(MyRank == Root){
       printf("\nEnter the number of intervals : ");
       scanf("%d",&NoInterval);
    }

   /*....Broadcast the number of subintervals to each processor....*/
   MPI_Bcast(&NoInterval, 1, MPI_INT, 0, MPI_COMM_WORLD);

   if(NoInterval <= 0){
      if(MyRank == Root)
        printf("Invalid Value for Number of Intervals .....\n");
      MPI_Finalize();
      exit(-1);
   }

   h   = 1.0 / (double)NoInterval;
   sum = 0.0;
   for(interval = MyRank + 1; interval <= NoInterval; interval += Numprocs){
       x = h * ((double)interval - 0.5);
       sum += func(x);
   }
   mypi = h * sum;

   /*....Collect the areas calculated in P0....*/
   MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, Root, MPI_COMM_WORLD);

   if(MyRank == Root){
      printf("pi is approximately %.16f, Error is %.16f\n",
                                             pi, fabs(pi - PI25DT));
   }

   MPI_Finalize();

}
```