Analysis and Design of Parallel Algorithms

Practica Nº 5

M. en C. Sandra Luz Morales Güitrón.

Instrucciones.

Copie y ejecute los siguientes programas, al final de la práctica deberá explicar cada uno de los programas de manera gráfica. Puede graficarlos a mano y luego escanearlos o bien puede usar alguna herramienta. Y anexarlo dentro de su reporte de práctica.

Practica.1.c

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

/*
! This program shows how to use MPI_Scatter and MPI_Reduce
! Each processor gets different data from the root processor
! by way of mpi_scatter.  The data is summed and then sent back
! to the root processor using MPI_Reduce.  The root processor
! then prints the global sum.
*/
/*  globals */
int numnodes,myid,mpi_err;
#define mpi_root 0
/* end globals  */

void init_it(int  *argc, char ***argv);

void init_it(int  *argc, char ***argv) {
       mpi_err = MPI_Init(argc,argv);
    mpi_err = MPI_Comm_size( MPI_COMM_WORLD, &numnodes );
    mpi_err = MPI_Comm_rank(MPI_COMM_WORLD, &myid);
}

int main(int argc,char *argv[]){
       int *myray,*send_ray,*back_ray;
       int count;
       int size,mysize,i,k,j,total,gtotal;

       init_it(&argc,&argv);
/* each processor will get count elements from the root */
       count=4;
       myray=(int*)malloc(count*sizeof(int));
/* create the data to be sent on the root */
       if(myid == mpi_root){
           size=count*numnodes;
              send_ray=(int*)malloc(size*sizeof(int));
              back_ray=(int*)malloc(numnodes*sizeof(int));
              for(i=0;i<size;i++)
```

```
                    send_ray[i]=i;
                }
/* send different data to each processor */
       mpi_err = MPI_Scatter(     send_ray, count,   MPI_INT,
                                                 myray,    count,   MPI_INT,
                                mpi_root,
                                MPI_COMM_WORLD);
/* each processor does a local sum */
       total=0;
       for(i=0;i<count;i++)
           total=total+myray[i];
       printf("myid= %d total= %d\n ",myid,total);
/* send the local sums back to the root */
    mpi_err = MPI_Reduce(&total,    &gtotal, 1,  MPI_INT,
                                        MPI_SUM,
                            mpi_root,
                            MPI_COMM_WORLD);
/* the root prints the global sum */
       if(myid == mpi_root){
         printf("results from all processors= %d \n ",gtotal);
       }
    mpi_err = MPI_Finalize();
}
```

Practica.2.c

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
/*
! This program shows how to use MPI_Gatherv.  Each processor sends a
! different amount of data to the root processor.  We use MPI_Gather
! first to tell the root how much data is going to be sent.
*/
/* globals */
int numnodes,myid,mpi_err;
#define mpi_root 0
/* end of globals */

void init_it(int  *argc, char ***argv);

void init_it(int  *argc, char ***argv) {
       mpi_err = MPI_Init(argc,argv);
    mpi_err = MPI_Comm_size( MPI_COMM_WORLD, &numnodes );
    mpi_err = MPI_Comm_rank(MPI_COMM_WORLD, &myid);
}

int main(int argc,char *argv[]){
/* poe a.out -procs 3 -rmpool 1 */
       int *will_use;
       int *myray,*displacements,*counts,*allray;
       int size,mysize,i;

       init_it(&argc,&argv);
       mysize=myid+1;
       myray=(int*)malloc(mysize*sizeof(int));
```

```
        for(i=0;i<mysize;i++)
                myray[i]=myid+1;
/* counts and displacement arrays are only required on the root */
        if(myid == mpi_root){
                counts=(int*)malloc(numnodes*sizeof(int));
                displacements=(int*)malloc(numnodes*sizeof(int));
        }
/* we gather the counts to the root */
        mpi_err = MPI_Gather((void*)myray,1,MPI_INT,
                                        (void*)counts,  1,MPI_INT,
                                        mpi_root,MPI_COMM_WORLD);
/* calculate displacements and the size of the recv array */
        if(myid == mpi_root){
                displacements[0]=0;
                for( i=1;i<numnodes;i++){
                        displacements[i]=counts[i-1]+displacements[i-1];
                }
                size=0;
                for(i=0;i< numnodes;i++)
                        size=size+counts[i];
                allray=(int*)malloc(size*sizeof(int));
        }
/* different amounts of data from each processor  */
/* is gathered to the root */
        mpi_err = MPI_Gatherv(myray, mysize,          MPI_INT,
                          allray,counts,displacements,MPI_INT,
                          mpi_root,
                          MPI_COMM_WORLD);

        if(myid == mpi_root){
                for(i=0;i<size;i++)
                        printf("%d ",allray[i]);
                printf("\n");
        }
     mpi_err = MPI_Finalize();
}
```

Practica.3.c

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

/*
! This program shows how to use MPI_Scatter and MPI_Gather
! Each processor gets different data from the root processor
! by way of mpi_scatter.  The data is summed and then sent back
! to the root processor using MPI_Gather.  The root processor
! then prints the global sum.
*/
/*  globals */
int numnodes,myid,mpi_err;
#define mpi_root 0
/* end globals  */
```

```
void init_it(int  *argc, char ***argv);

void init_it(int  *argc, char ***argv) {
       mpi_err = MPI_Init(argc,argv);
    mpi_err = MPI_Comm_size( MPI_COMM_WORLD, &numnodes );
    mpi_err = MPI_Comm_rank(MPI_COMM_WORLD, &myid);
}

int main(int argc,char *argv[]){
       int *myray,*send_ray,*back_ray;
       int count;
       int size,mysize,i,k,j,total;

       init_it(&argc,&argv);
/* each processor will get count elements from the root */
       count=4;
       myray=(int*)malloc(count*sizeof(int));
/* create the data to be sent on the root */
       if(myid == mpi_root){
           size=count*numnodes;
              send_ray=(int*)malloc(size*sizeof(int));
              back_ray=(int*)malloc(numnodes*sizeof(int));
              for(i=0;i<size;i++)
                     send_ray[i]=i;
              }
/* send different data to each processor */
       mpi_err = MPI_Scatter(     send_ray, count,   MPI_INT,
                                          myray,    count,   MPI_INT,
                                 mpi_root,
                                 MPI_COMM_WORLD);

/* each processor does a local sum */
       total=0;
       for(i=0;i<count;i++)
           total=total+myray[i];
       printf("myid= %d total= %d\n ",myid,total);
/* send the local sums back to the root */
    mpi_err = MPI_Gather(&total,    1,  MPI_INT,
                                        back_ray, 1,  MPI_INT,
                            mpi_root,
                            MPI_COMM_WORLD);
/* the root prints the global sum */
       if(myid == mpi_root){
         total=0;
         for(i=0;i<numnodes;i++)
           total=total+back_ray[i];
         printf("results from all processors= %d \n ",total);
       }
    mpi_err = MPI_Finalize();
}
```