Sistemas Distribuidos

Practica 5: MPI

Arturo Fuentes Velasco

14 de marzo de 2020

1. Resumen

En esta practica se trabajara el paso de mensaje entre procesos usando MPI para crear una comunicación del tipo broadcast entre un nodo y sus vecinos y así crear un árbol abarcador de la topología de red de manera distribuida.

2. Introducción

MPI (Message Passing Interface) es un protocolo de comunicación usado para el paso de mensaje entre varias arquitecturas de computo paralelo. La implementación de MPI que se usa para esta practica es la de Open MPI y se usaran

Se usaran dos tipos de comunicación punto a punto, comunicación síncrona y asíncrona. Se usara MPI_Send y MPI_Recv, que son bloqueantes, para el enviar y recibir mensajes de forma síncrona. Para enviar y recibir mensajes de forma asíncrona se usa MPI_Isend y MPI_iRecv, que no son bloqueantes.

Para poder empaquetar y mandar los datos deseados, MPI usa los siguientes tipos de datos al enviar y recibir mensajes: MPI_INT, MPI_FLOAT, MPI_CHAR, MPI_LONG, MPI_DOUBLE Y MPI_LONG_DOUBLE. Cada tipo de dato usado por MPI corresponde a un tipo de dato primitivo dentro del lenguaje de programación, en este caso C.

Para esta práctica se necesita saber que es un broadcast, o mensaje tipo broadcast. Este tipo de comunicación es del tipo uno a todos, el nodo que lanza el broadcast manda un mensaje a cada uno de sus vecinos. Este tipo de mensaje hace posible la inundación de todas los vecinos de un nodo, por lo que se usara.

3. Problema

Usando la topología de red vista en la Figura 1 se deberá de enviar un mensaje a todos los nodos por medio de inundación. Se deberá escoger un nodo tal que él mande un mensaje a sus vecinos notificándolos de quien es su padre.Los vecinos de este nodo deberán repetir el proceso hasta que los nodos hayan recibido un mensaje de retroalimentación de sus hijos.

La red sera divida en N grupos, donde N sera el número de nodos en la red. Cada grupo sera compuesto solamente de los vecinos del nodo. De

esta forma los nodos individuales no conocen a la topología de la red y se ve innecesario saber el número de nodos totales y los vecinos de cada nodo.

Al mandar el nodo los mensajes, lo hará con broadcast. Si el nodo no ha sido visitado por otro nodo, volverá a hacer el broadcast y si ya ha sido visitado no hará nada.

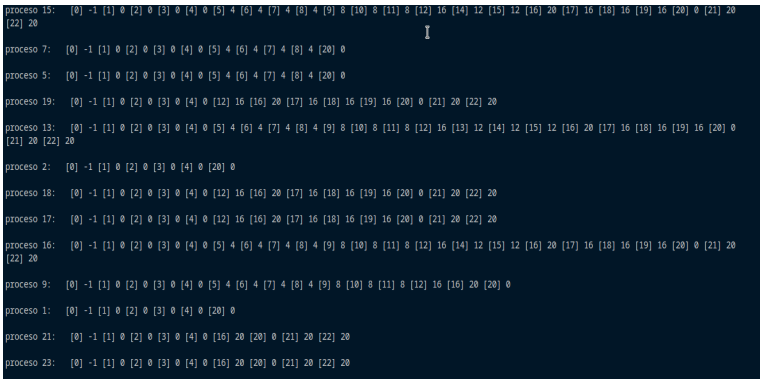

Figura 1: topología de red usada

4. Desarrollo y Resultados

Para saber los vecinos del nodo se usara la función *getNeighbors* que regresa una arreglo con los vecinos del nodo. Esta función toma en cuenta la topología de la red, la cual se compone de una red de anillo junto con cuatro redes estrella.

Ya teniendo la función se inicializara el programa de MPI junto con las variables para saber el identificador el nodo y la cantidad de procesos que se tienen en el programa.

```
1 MPI_Init(&argc, &argv);
2 int world_rank, world_size;
3 MPI_Comm_rank(MPI_COMM_WORLD, &world_rank); //Para obtener el ID
4 MPI_Comm_size(MPI_COMM_WORLD, &world_size); //Para obtener el
    numero de procesos
```

Después de inicializar las variables de MPI, inicializamos n que contiene el número de vecinos del nodo dada la topología. Después tenemos que el nodo no ha sido visitado y luego se inicializa el arreglo de vecinos del nodo

de acuerdo de su identificador en la red. Al final de usa una barrera para sincronizar todos los procesos y poder avanzar desde el mismo punto.

```
int n=(world_rank%4==0)?5:1;
int visitado = FALSE;
// Obtengo vecinos
int *neighbors = getNeighbors(world_rank
int mis_padres[N];
packet mi_nodo;
packet mensaje;
int visitado=FALSE;
for(int i=0;i<N;i++) mi_nodo.arr[i]=-2;
for(int i=0;i<N;i++) mis_padres[i]=-2;
MPI_Barrier(MPI_COMM_WORLD);
```

Se usara un ciclo while para saber cuando debe de terminar el intercambio de mensajes.

```
int isdone=0;
int l=0;
while(!isdone)
```

Tendremos dos casos para los nodos, el primero sera el caso del nodo raíz, en el cual el nodo 0 pero se puede escoger cualquier nodo como raíz. El nodo empieza este caso cambiando su estado de *visitado* a verdadero y luego procede a lanzar un mensaje tipo *broadcast*, que manda un mensaje a todo los vecinos del nodo:

```
if(!visitado){
    visitado=1;
    mi_nodo.padre = -1;
    mi_nodo.rango=world_rank;
    mi_nodo.arr[mi_nodo.rango]=mi_nodo.padre;
    mis_padres[mi_nodo.rango]=mi_nodo.padre;
    for (int i = 0; i < n; i++){
      MPI_Send(&mi_nodo, 2+N, MPI_INT, neighbors[i], 1,
   MPI_COMM_WORLD);
    }
}
```

Una vez que el nodo manda el mensaje a sus vecinos, espera los mensajes de de sus vecinos y manda el mensaje de retroalimentación y sale del ciclo:

```
else{
    for (int i = 0; i < n; i++){
      MPI_Recv(&mensaje, 2+N, MPI_INT, neighbors[i], 1,
   MPI_COMM_WORLD, MPI_STATUS_IGNORE);
```

```
4         for(int k=0;k<N;k++){
5           if(mensaje.arr[k]>-2 && mensaje.arr[k] <N) mis_padres[k
    ]=mensaje.arr[k];
6         }
7       }
8       for(int k=0;k<N;k++) mi_nodo.arr[k]=mis_padres[k];
9       //mandar que ya se recibio el mensaje, retro
10      for (int i = 0; i < n; i++){
11        MPI_Send(&mi_nodo, 2+N, MPI_INT, neighbors[i], 1,
    MPI_COMM_WORLD);
12        printf("Nodo %d mando retro a %d\n", world_rank, neighbors
    [i]);
13      }
14      isdone=1;
15  }
```

En el segundo caso los nodos esperan la respuesta de sus vecinos si ya fueron visitados y manda el mensaje de retroalimentación a su padre:

```
1   if(visitado){
2       for (int i = 0; i < n; i++){
3         MPI_Recv(&mensaje, 2+N, MPI_INT, MPI_ANY_SOURCE, 1,
    MPI_COMM_WORLD, MPI_STATUS_IGNORE);
4         for(int k=0;k<N;k++){
5           if(mensaje.arr[k]>-2 && mensaje.arr[k] <N) mis_padres[k
    ]=mensaje.arr[k];
6         }
7       }
8       isdone=1;
9       //nodo recibe toda retro
10      MPI_Send(&mi_nodo, 2+N, MPI_INT, mi_nodo.padre, 1,
    MPI_COMM_WORLD);
11  }
```

Si no ha sido visitado recibe el mensaje de su padre:

```
1   else {
2       MPI_Recv(&mensaje, 2+N, MPI_INT, MPI_ANY_SOURCE, 1,
    MPI_COMM_WORLD, MPI_STATUS_IGNORE);
3       for(int k=0;k<N;k++){
4         if(mensaje.arr[k]>-2 && mensaje.arr[k] <N) mis_padres[k]=
    mensaje.arr[k];
5       }
6   }
```

Si no ha sido visitado el nodo inicializa a su padre y manda mensaje a sus vecinos:

```
if(!visitado){
    mi_nodo.padre=mensaje.padre;
    mi_nodo.rango=world_rank;
    mi_nodo.arr[mi_nodo.rango]=mi_nodo.padre;
    mi_nodo.padre=mensaje.rango;
    mis_padres[mi_nodo.rango]=mi_nodo.padre;
    visitado=1;
    //paso 3->2
    for(int k=0;k<N;k++) mi_nodo.arr[k]=mis_padres[k];
    for (int i = 0; i < n; i++){
      MPI_Send(&mi_nodo, 2+N, MPI_INT, neighbors[i], 1,
   MPI_COMM_WORLD);
      printf("Nodo %d mando mensaje a %d\n", world_rank,
   neighbors[i]);
    }
}
```

Si ya fue visitado el nodo, manda un mensaje de retroalimentación a sus vecinos:

```
else{
    for(int k=0;k<N;k++) mi_nodo.arr[k]=mis_padres[k];
    for (int i = 0; i < n; i++){
      MPI_Send(&mi_nodo, 2+N, MPI_INT, neighbors[i], 1,
   MPI_COMM_WORLD);
      printf("Nodo %d mando retro a %d\n", world_rank, neighbors
   [i]);
    }
}
```

Después de esto se terminan de intercambiar mensajes entre vecinos y los padres están listos para recibir los mensajes de sus hijos:

```
for(int i=0;i<N;i++){
    if(mis_padres[i]==world_rank) MPI_Recv(&mensaje, 2+N,
   MPI_INT, MPI_ANY_SOURCE, 1, MPI_COMM_WORLD, MPI_STATUS_IGNORE
   );
    for(int k=0;k<N;k++){
            if(mensaje.arr[k]>-2 && mensaje.arr[k] <N)
   mis_padres[k]=mensaje.arr[k];
          }
  }
  for(int k=0;k<N;k++) mi_nodo.arr[k]=mis_padres[k];
```

En las figuras 2 y **??** se ve como se crearon los árboles para cada nodo. No son los mismos para cada nodo, por cuestiones de implementación algunos

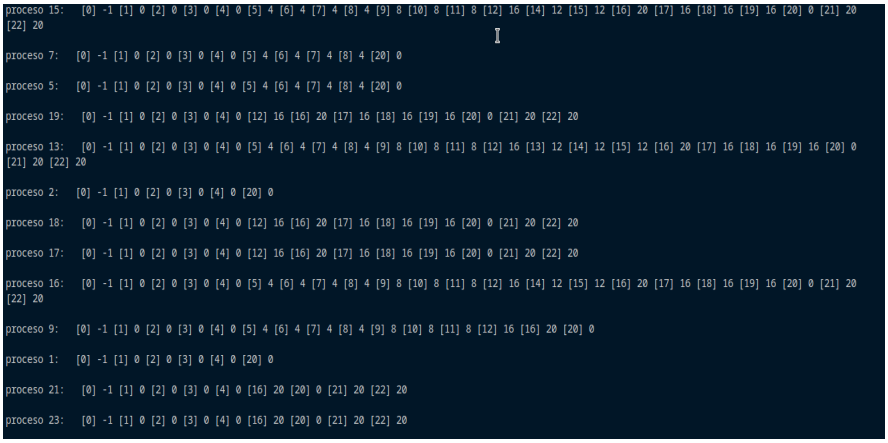

Figura 2: Padres de los nodos en texto

nodos no logran tener la respuesta final, pero algunos nodos si logran obtener el árbol generador completo.

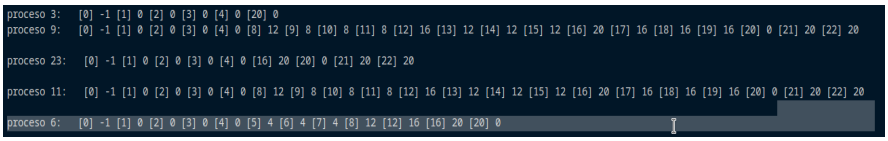

Figura 3: Padres de los nodos en texto

5. Conclusiones

Se logro la comunicación tipo broadcast de un nodo junto con sus vecinos. No se crearon grupos MPI y para lograr la comunicación broadcast se implemento la funcionalidad de MPI_Bcast, ésta siendo un ciclo para mandar o recibir mensajes de todos los vecinos. Por ésta razón la creación de grupos no fue necesaria, en vez se uso un arreglo con los vecinos de los vecinos. Esté arreglo fue creado de manera estática dado que se le dio la topología de red al programa, es decir, el programa no genero la topología de red.

Con el broadcast implementado usando un ciclo se logro inundar toda la red de mensaje para obtener el nodo padre de que cada nodo que es quien mando a llamar ese nodo. En los dos tipos de comunicación, síncrona y asíncrona se tuvo el mismo resultado con el mismo árbol abarcador mostrado

en la Figura **??**. El árbol creado por el programa cambia de acuerdo nodo donde se encuentra pero siempre es un árbol abarcador, aunque no se sabe si se crea el árbol abarcador mínimo.

6. Bibliografía

1. https://curc.readthedocs.io/en/latest/programming/MPI-C.html
2. https://mpitutorial.com/tutorials/introduction-to-groups-and-communicators/
3. https://mpitutorial.com/tutorials/mpi-broadcast-and-collective-communication/