Universidad de Castilla-La Mancha
Escuela Superior de Informática

Diseño de Infraestructura de Red

Grado en Ingeniería Informática
3º - Ingeniería de Computadores
2019 – 2020

Práctica 1: Red Hipercubo

Autor:
Mario Pérez Sánchez-Montañez

Fecha:
18 de marzo de 2020

Índice

1. Enunciado

Dado un archivo con nombre datos.dat, cuyo contenido es una lista de valores separados por comas, nuestro programa realizará lo siguiente: El proceso de rank 0 destribuirá a cada uno de los nodos de un Hipercubo de dimensión D, los 2^D numeros reales que estarán contenidos en el archivo datos.dat. En caso de que no se hayan lanzado suficientes elementos de proceso para los datos del programa, éste emitirá un error y todos los procesos finalizarán. En caso de que todos los procesos han recibido su correspondiente elemento, comenzará el proceso normal del programa. Se pide calcular el elemento mayor de toda la red, el elemento de proceso con rank 0 mostrará en su salida estándar el valor obtenido. La complejidad del algoritmo no superará $O(\log_2(n))$ Con n número de elementos de la red.

2. Planteamiento de la solución

Las redes de hipercubos son un tipo de topología de red utilizada para conectar múltiples procesadores. Las redes de hipercubos constan de 2^m nodos, que forman los vértices de los cuadrados para crear una conexión entre redes. Un hipercubo es básicamente una red de malla multidimensional con dos nodos en cada dimensión, como se puede ver en la Figura 1.

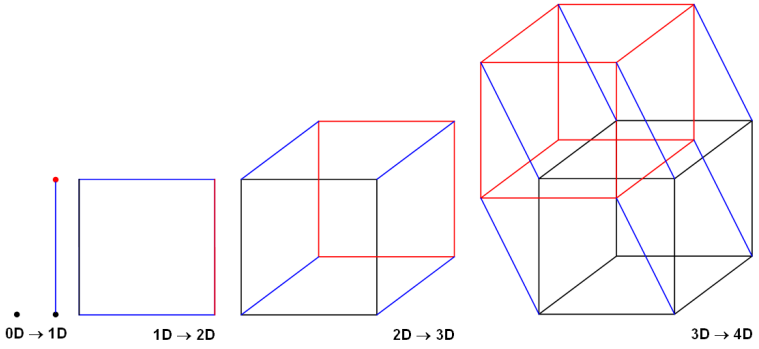

Figura 1: Ejemplo topología hipercubo [5]

La solución planteada se basa en crear 2^D procesos (siendo D la dimensión del hipercubo). En este caso planteo una solución de dimensión 3, por lo tanto, 8 procesos. En el fichero ***hypercube.c*** se especifica una constante para la dimensión. Por ello, si se quiere cambiar la dimensión del hipercubo se debe modificar esta constante y los nodos indicados en el ***Makefile***.

El el proceso con rank 0 se encarga de leer el fichero ***datos.dat*** y de enviar el respectivo número a cada nodo, incluido sí mismo. Si este proceso se realiza con éxito los procesos

comenzarán a enviar sus número a sus vecinos en busca del máximo número de la red, en caso contrario la variable ***end*** se activará todos los procesos finalizarán.

2.1. Cálculo de vecinos

La parte fundamental de la solución es el cálculo de los respectivos vecinos, ya que esto especificará la topología de la red [4]. Cada nodo tiene dos vecinos, que se calculan aplicando la operación binaria ***XOR*** [1] (que produce verdadero solo cuando las entradas difieren), aplicada según este algoritmo [2] (ver Figura 2)

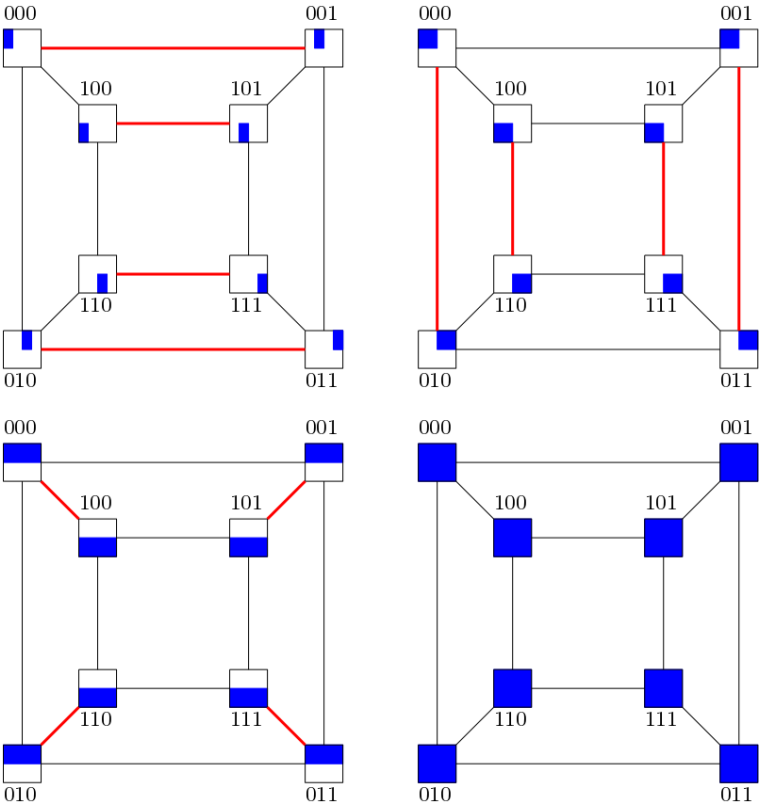

Figura 2: Patrón de comunicación hipercubo [3]

3. Diseño de la solución

3.1. Estructura de la función principal

En esta función se inicializan las variables principales de nuestro programa y se realizan las llamadas al resto de funciones, según la acción que se deba realizar. En cuanto a *MPI*

se realizan las siguientes acciones:

- Inicializar la estructura de comunicación de MPI entre los procesos [9].
- Determinar el *rank* (identificador) del proceso que lo llama dentro del comunicador seleccionado [7].
- Determinar el tamaño del comunicador seleccionado, es decir, el número de procesos que están actualmente asociados a este [8].

```
1  int main(int argc, char *argv[]){
2          double *data = malloc(N_NODES * sizeof(double));
3          int length;
4          int rank, size;
5          double num;
6
7          MPI_Init(&argc, &argv);
8          MPI_Comm_rank(MPI_COMM_WORLD,&rank);
9          MPI_Comm_size(MPI_COMM_WORLD,&size);
10
11         if (rank == 0){
12                 get_data(data, &length);
13
14                 check_data(length, LENGTH_MSG);
15
16                 if (!end) check_data(size, SIZE_MSG);
17
18                 if (!end) send_data(data);
19         }
20
21         /* Get confirmation to continue from first node */
22         MPI_Bcast(&end,1,MPI_INT,0,MPI_COMM_WORLD);
23
24         if(!end){
25                 /* Wait the number */
26                 MPI_Recv(&num, 1, MPI_DOUBLE, 0, MPI_ANY_TAG,
                       MPI_COMM_WORLD, NULL);
27                 calculate_max(rank, num);
28         }
29
30         MPI_Finalize();
31
32         return EXIT_SUCCESS;
33 }
```

A continuación, se van a describir las distintas acciones que se llevan a cabo en el código anterior con las respectivas funciones utilizadas.

3.2. Lectura del fichero

La lectura del fichero para obtener los distintos números se realiza con se realiza con la siguiente función:

```
void get_data(double *data, int *length){
        /* For load data from datos.dat */

        FILE *file;
        char *aux = malloc(1024 * sizeof(char));
        char *n;

        if ((file = fopen(FILE_NAME, "r")) == NULL){
                fprintf(stderr, "Error opening file\n");
                end = TRUE;
        }else{
                *length = 0;

                fscanf(file, "%s", aux);
                fclose(file);

                data[(*length)++] = atof(strtok(aux,","));

                while((n = strtok(NULL, ",")) != NULL)
                        data[(*length)++] = atof(n);
        }

        free(aux);

}
```

Los parámetros de la función son un array de *double* para almacenar los números leídos del fichero y una variable entera que representa la cantidad de números obtenidos. Ambos parámetros se introducen por referencia, ya que las modificaciones las necesitamos posteriormente.

3.3. Tratamiento de errores

Una vez concluida la carga de los datos, se realizarán dos comprobaciones para iniciar los cálculos.

- En primer lugar, se comprueba el tamaño obtenido de MPI (variable ***size***)
- En segundo lugar, si la comprobación anterior ha resultado exitosa se comprueba el número de elementos obtenidos, almacenado en la variable ***length***.

Ambas comprobaciones se realizan con la siguiente función, ya que las dos valores a comprobar se comparan con el número de nodos totales (2^D).

```
1  void check_data(int var, char *type){
2          /* For check length or size */
3
4          if (var != N_NODES){
5                  fprintf(stderr, "Error in data %s\n", type);
6                  end = TRUE;
7          }
8  }
```

3.4. Envío de datos

Las comprobaciones activarán la variable global ***end*** si hay algún valor incorrecto. Si los valores comprobados son correctos se procede a enviar los datos, utilizando la siguiente función.

```
1  void send_data(double *data){
2          /* Send numbers to all the nodes */
3
4          double buff_num;
5          int i;
6
7          for(i=0; i < N_NODES; i++){
8                  buff_num = data[i];
9                  MPI_Send(&buff_num, 1, MPI_DOUBLE, i, 0,
                      MPI_COMM_WORLD);
10                 printf("%.2f sended to %d node\n",buff_num, i);
11         }
12
13         free(data);
14 }
```

En lo relativo a MPI, se hace uso de la función de envío como se puede observar en la línea 9 del bloque de código anterior.

3.5. Recepción de datos

Antes de realizar la recepción de los datos se realiza un envío de la variable de comprobación (***end***) a todos los procesos utilizando la función de *broadcast* de MPI [6]. La razón es que los diferentes procesos no deben recibir sus datos si las comprobaciones no han sido

exitosas. Para la recepción del número que le corresponde a cada nodo se ha utilizado la primitiva básica de recepción de MPI [10].

3.6. Cálculo del máximo número de la red

El objetivo del programa es que todos los nodos de la red se queden con el número máximo de todo el conjunto de nodos. Para ello, los nodos se deben enviar los distintos números de unos a otros hasta encontrar el máximo. Tras esto, el primer nodo (con *rank* 0) imprimirá el número que tiene actualmente, es decir, el mínimo de toda la red.

3.6.1. Implementación de los vecinos

Los distintos nodos de la red enviarán y recibirán los números de sus vecinos respectivamente. Es por ello, que lo más importante para que el cálculo del máximo de la red se realice correctamente es calcular los cuatro vecinos que tiene cada nodo de forma correcta.

El siguiente bloque de código representa la implementación del cálculo de los vecinos para un determinado nodo de la red. La forma de obtener los vecinos se ha explicado de manera teórica en la sección 2.1.

```
1 void hypercube_neighbors(int rank, int neighbors[]){
2         /* Calculate the neighbors */
3
4         int i;
5
6         for(i = 0; i < D; i++)
7                 neighbors[i] = XOR(rank, (int)pow(2,i));
8 }
```

En la topología hipercubo, cada elemento de procesamiento itera ejecutando la expresión $XOR(rank, 2^i)$ para obtener cada vecino. Siendo i la variable del bucle que toma valores desde 0 hasta *D-1*. Por ejemplo, para 3 dimensiones cada nodo tendrá tres vecinos.

La función tiene como parámetro un array, donde se almacenarán los *ranks* de los vecinos del nodo que haya ejecutado la función. Como los vectores en C se pasan por referencia, los valores de este vector los podremos utilizar después de ejecutar está función.

3.6.2. Envío y recepción de los números

Una vez calculados los vecinos de cada nodo, los procesos comenzarán a enviar su número a sus vecinos y recibir otros valores para comparar el que se recibe con el que se tiene actualmente.

```
void calculate_max(int rank, double num){
        /* Calculate the maximum number of all the nodes */

        int neighbors[D];
        double his_num;
        int i;

        hypercube_neighbors(rank, neighbors);

        for(i=1; i < D; i++){
                MPI_Send(&num, 1, MPI_DOUBLE, neighbors[i], 1,
                    MPI_COMM_WORLD);
                MPI_Recv(&his_num, 1, MPI_DOUBLE, neighbors[i], 1,
                    MPI_COMM_WORLD, NULL);
                num = MAX(num, his_num);
        }

        if(rank == 0) printf("\nThe maximum number is: %.2f\n",num)
            ;

}
```

El algoritmo para calcular el máximo de la red hipercubo primero calcula todos los vecinos utilizando la función explicada anteriormente y se realiza el intercambio entre vecinos utilizando las funciones de MPI de envío [11] y recepción [10].

3.6.3. Implementación final

Tras un estudio de la práctica y basándome en [2] he realizado un nueva implementación. En la cual se realiza el cálculo del vecino correspondiente y el intercambio de números en el mismo bucle, como se puede ver en el siguiente bloque de código:

```
void calculate_max(int rank, double num){
        /* Calculate the maximum number of all the nodes

        double his_num;
        int neighbor, i;

        for(i=0; i < D; i++){
                neighbor = XOR(rank, (int)pow(2,i));
                MPI_Send(&num, 1, MPI_DOUBLE, neighbor, 1,
                    MPI_COMM_WORLD);
                MPI_Recv(&his_num, 1, MPI_DOUBLE, neighbor, 1,
                    MPI_COMM_WORLD, NULL);
                num = MAX(num, his_num);
```

```
12          }
13
14          if(rank == 0) printf("\nThe maximum number is: %.2f\n",num)
                ;
15
16  }
```

Se puede observar como se calcula el vecino correspondiente a la iteración del bucle en la que se encuentre el programa y, a continuación, se realiza el intercambio de números buscando el mayor.

Referencias

[1] https://en.wikipedia.org/wiki/Exclusive_or. Or exclusiva.

[2] https://en.wikipedia.org/wiki/Hypercube_(communication_pattern)#Algorithm_outline. Algoritmo hipercubo.

[3] https://en.wikipedia.org/wiki/Hypercube_(communication_pattern)#/media/File:Hypergraph_Communication_Pattern.png. Ejemplo gráfico algoritmo.

[4] https://en.wikipedia.org/wiki/Hypercube_internetwork_topology. Topología hipercubo.

[5] https://en.wikipedia.org/wiki/Hypercube_internetwork_topology#/media/File:Hypercube-construction-4d.png. Ejemplo hipercubo.

[6] https://lsi.ugr.es/jmantas/ppr/ayuda/mpi_ayuda.php?ayuda=MPI_Bcast. Broadcast mpi.

[7] https://lsi.ugr.es/jmantas/ppr/ayuda/mpi_ayuda.php?ayuda=MPI_Comm_rank. Determinar *rank* mpi.

[8] https://lsi.ugr.es/jmantas/ppr/ayuda/mpi_ayuda.php?ayuda=MPI_Comm_size. Determinar tamaño del comunicador mpi.

[9] https://lsi.ugr.es/jmantas/ppr/ayuda/mpi_ayuda.php?ayuda=MPI_Init. Inicialización mpi.

[10] https://lsi.ugr.es/jmantas/ppr/ayuda/mpi_ayuda.php?ayuda=MPI_Recv. Recepción mpi.

[11] https://lsi.ugr.es/jmantas/ppr/ayuda/mpi_ayuda.php?ayuda=MPI_Send. Envío mpi.