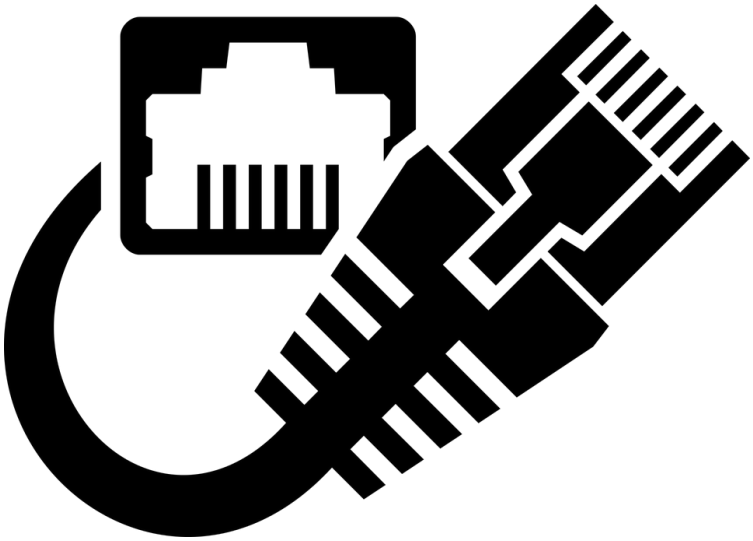
P1 - Topologías de Red



Pablo Rodríguez Solera

Diseño de Infraestructura de Red

➔ Prof. Javier Ayllón

➔ Escuela Superior de Informática.

➔ Universidad de Castilla – La Mancha

Índice:

P1 – Topologías de Red:

1. Red Toroide

1.1 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 toroide de lado L,los L x L 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 menor 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(raiz_cuadrada(n)) Con n número de elementos de la red.

1.2 Planteamiento de Solución:

Para llegar a la solución del problema planteado el programa realiza los siguientes pasos:

1. El proceso 0 abre el fichero `datos.dat` y obtiene el número de elementos que contiene.
2. Se guardan los elementos del fichero en un array de elementos del tipo float.
3. El proceso 0[1] distribuye los números a los distintos procesos lanzados.
4. Cada uno de los procesos lanzados (a excepción del 0) recibe su número.
5. Estos procesos obtienen un array de tipo int que guarda el identificador de sus vecinos en la red toroide.
6. Se calcula el número mínimo de entre los distribuidos en los distintos nodos del sistema de la siguiente manera:
    1. Cada uno de los nodos manda a su vecino del sur su número.
    2. Cada uno de los nodos recibe de su nodo norte el número de este.
    3. Cada nodo se queda con el número mínimo de entre estos dos.
    4. Se repite la operación con el vecino del este, mandándole su número.
    5. Se recibe el número del vecino del oeste.

1 En el diseño de esta solución se ha optado por que el proceso 0 solo se encargue de distribuir los números a los demás procesos e imprimir el resultado de la operación conjunta, sin participar en el cálculo de la misma.

6. Se vuelven a comparar los números y el nodo se queda con el número menor.

7. Cada paso de envío y recepción se repite tantas veces como grande sea el lado de la red toroide, de esta manera nos aseguramos que cada nodo se va a quedar con el número menor de toda la red.

8. Como a la finalización de este proceso, todos los nodos disponen del número menor, cualquiera de ellos puede mandarle un mensaje al proceso con identificador 0 comunicándole el resultado de la operación. En este caso hemos optado por el proceso 1.

9. El proceso con identificador 0 recibe el número resultado y lo muestra por pantalla.

1.3 Diseño del Programa:

El programa está diseñado en el lenguaje de programación c y utiliza los comandos y estructuras propios de MPI (Message Passing Interface).

En este caso se ha optado por incluir las funciones comunes a la red Toroide y a la red Hipercubo en un archivo a parte llamado `funciones.c`, que contiene algunas funciones como:

- Abrir el archivo `datos.dat`.
- Obtener el número de elementos del mismo.
- Obtener los elementos del archivo y guardarlos en un array.
- Distribuir los números.
- Recibir el número correspondiente a cada proceso.

Por otra parte tenemos el fichero `MinimoToroide.c` que incluye toda la lógica del programa así como la estructura MPI del mismo.

Este archivo se encarga de distinguir entre las funciones del proceso 0 y los demás procesos, además, contiene una gestión de errores básica y se encarga de ejecutar el algoritmo que da solución al problema, además de parte de los requisitos de esta como sería obtener los vecinos de cada uno de los nodos de la red.

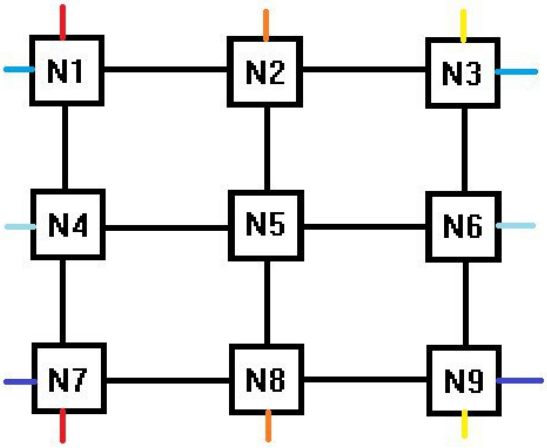
Además, muestra por pantalla el resultado final de la operación si no ha habido ningún error.

En el caso de que exista algún error también se muestra por pantalla el mismo así como una posible solución. Estos son los errores que se han gestionado:

- El archivo `datos.dat` no contiene suficientes números para la red indicada.
- No se han lanzado suficientes procesos para la red indicada.
- Error en la introducción de argumentos del programa.

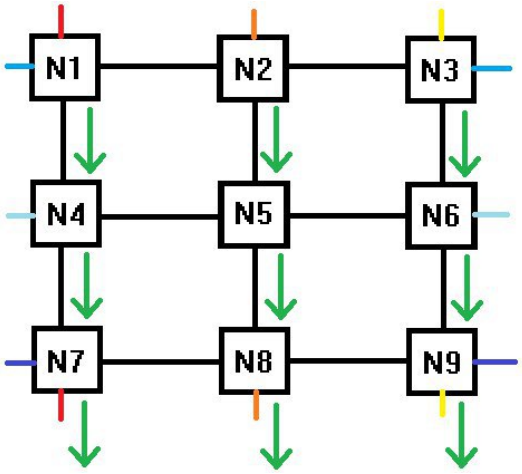
1.4 Explicación de flujo de datos en la red para cada comando MPI usado en la solución:

Supongamos que esta es una red toroide de lado 3.

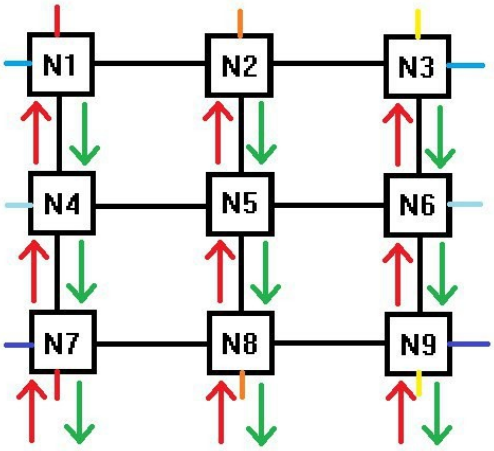


Todos los elementos tienen 4 vecinos, NORTE, SUR, ESTE Y OESTE.

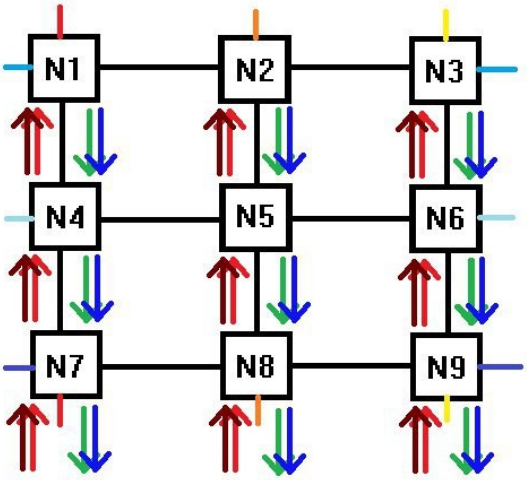
Primero se realiza un `MPI_Bsend` sobre los vecinos del sur.



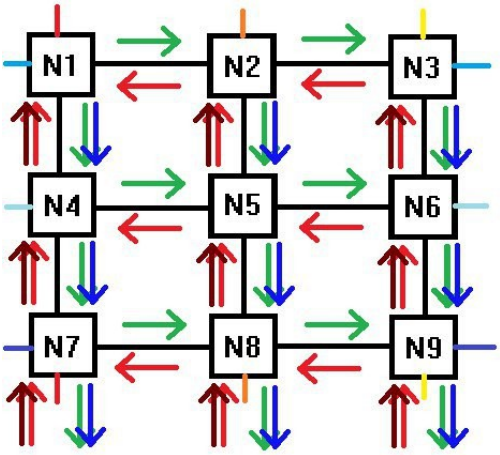
Después se realiza un `MPI_Receive` sobre los vecinos del norte.



Esta operación se realiza tantas veces como el lado de la red toroide. (En nuestro caso 3, es decir, cada nodo hace 3 envíos y 3 recepciones.

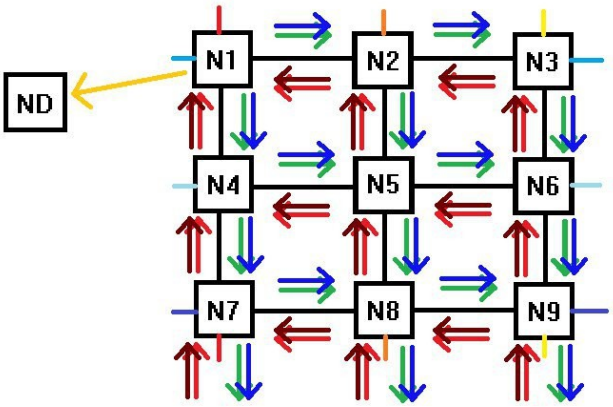


Cuando se realizan todos los envíos y recepciones correspondientes se pasa a enviar al nodo este y recibir del oeste:



Igual, se repite la operación tantas veces como el lado del toroide.

Finalmente, cuando ya se han terminado todas las iteraciones y comparaciones, el proceso 1 realiza un `MPI_Bsend` al nodo distribuidor, que realiza un `MPI_Receive` para obtener el resultado.



1.5 Instrucciones:

Las instrucciones de lanzamiento del programa están basadas en su compilación y ejecución siguiendo el `Makefile`. Además, están también explicadas en el README.

- Para compilar el programa:

  ```
  make compilar
  ```

- Para lanzar el programa:

  ```
  make MinimoToroide PROCESOS=numero_procesos LADO=lado_toroide
  ```

  - Siendo `numero_procesos` el número de procesos que se desean lanzar y `lado_toroide` el tamaño del lado de la red toroide que queremos utilizar.

- Para eliminar el archivo de ejecución:

  ```
  make limpiar
  ```

Nota: Siempre se debe ejecutar al menos un proceso más que el número de nodos de la red.

2. Red Hipercubo

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(logaritmo_base_2(n)))) Con n número de elementos de la red.

2.1 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(logaritmo_base_2(n)))) Con n número de elementos de la red.

2.2 Planteamiento de Solución:

Para llegar a la solución del problema planteado el programa realiza los siguientes pasos:

1. El proceso distribuidor abre el fichero `datos.dat` y obtiene el número de elementos que contiene.
2. Se guardan los elementos del fichero en un array de elementos del tipo float.
3. El proceso distribuidor[2] distribuye los números a los distintos procesos lanzados.
4. Cada uno de los procesos lanzados (a excepción del distribuidor) recibe su número.
5. Estos procesos obtienen un array de tipo int que guarda el identificador de sus vecinos en la red hipercubo, es decir, sus vecinos en cada dimensión.
6. Se calcula el número mínimo de entre los distribuidos en los distintos nodos del sistema de la siguiente manera:
    1. Cada uno de los nodos manda a sus vecinos en las distintas dimensiones su número.
    2. Cada uno de los nodos recibe de todos sus vecinos el número de estos.
    3. En cada envío y recepción se calcula el número mínimo de entre el del propio proceso y el recibido y el proceso se queda con el mayor.
7. Cada paso de envío y recepción se repite tantas veces como dimensiones tenga la red hipercubo, para asegurarnos de que al final de la operación todos tienen el mismo resultado.
8. Como a la finalización de este proceso, todos los nodos disponen del número mayor, cualquiera de ellos puede mandarle un mensaje al proceso distribuidor comunicándole el resultado de la operación. En este caso hemos optado por el proceso 1.
9. El proceso distribuidor recibe el número resultado y lo muestra por pantalla.

2 En el diseño de esta solución, existe un proceso distribuidor, que en este caso es siempre el proceso con identificador más alto (para poder mantener el mismo formato que en la otra solución donde el distribuidor solo manda e imprime el resultado sin participar en la operación.

2.3 Diseño del Programa:

El programa está diseñado en el lenguaje de programación c y utiliza los comandos y estructuras propios de MPI (Message Passing Interface).

En este caso se ha optado por incluir las funciones comunes a la red Toroide y a la red Hipercubo en un archivo a parte llamado `funciones.c`, que contiene algunas funciones como:

- Abrir el archivo `datos.dat`.
- Obtener el número de elementos del mismo.
- Obtener los elementos del archivo y guardarlos en un array.
- Distribuir los números.
- Recibir el número correspondiente a cada proceso.

Por otra parte tenemos el fichero `MaximoHipercubo.c` que incluye toda la lógica del programa así como la estructura MPI del mismo.

Este archivo se encarga de distinguir entre las funciones del proceso distribuidor y los demás procesos, además, contiene una gestión de errores básica y se encarga de ejecutar el algoritmo que da solución al problema, además de parte de los requisitos de esta como sería obtener los vecinos de cada uno de los nodos de la red.

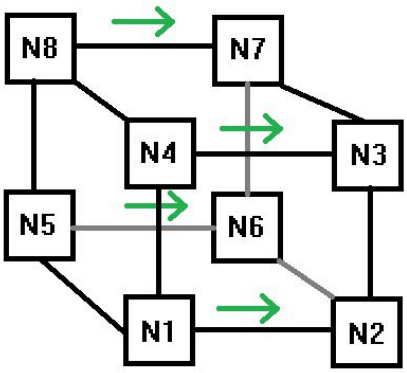
Además, muestra por pantalla el resultado final de la operación si no ha habido ningún error.

En el caso de que exista algún error también se muestra por pantalla el mismo así como una posible solución. Estos son los errores que se han gestionado:

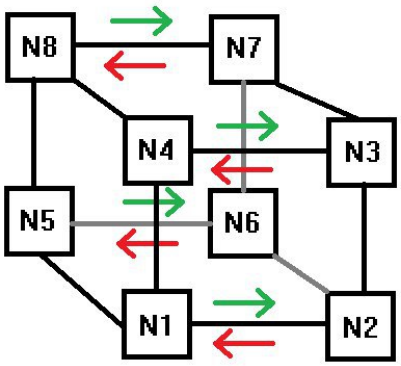
- El archivo `datos.dat` no contiene suficientes números para la red indicada.
- No se han lanzado suficientes procesos para la red indicada.
- Error en la introducción de argumentos del programa.

2.4 Explicación de flujo de datos en la red para cada comando MPI usado en la solución:

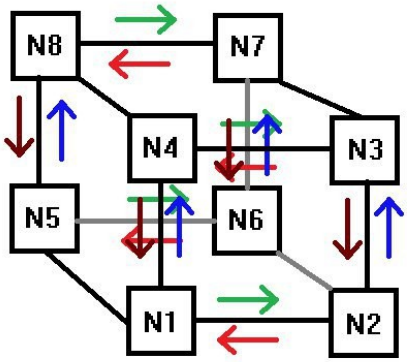
Supongamos una red hipercubo de 3 dimensiones. Primero se realiza un `MPI_Bsend` al nodo vecino de la primera dimensión:

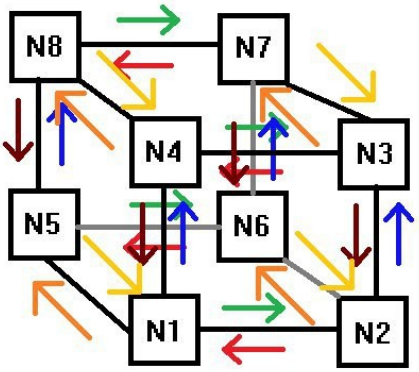


Después se realiza un MPI_Receive del nodo vecino de la primera dimensión.

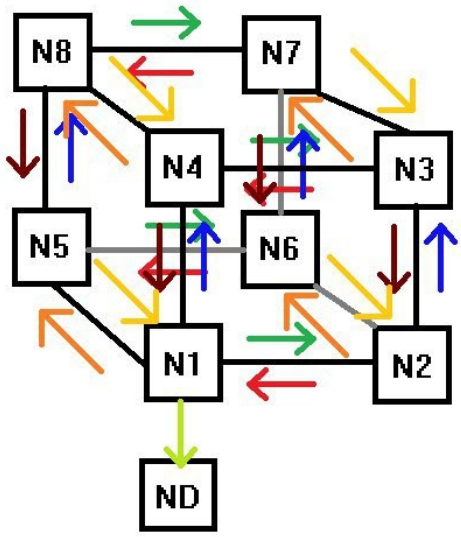


Se realizan estas operaciones en las dimensiones restantes:





Finalmente, el nodo 1 envía al nodo distribuidor un mensaje con el resultado de la operación.



2.5 Instrucciones:

Las instrucciones de lanzamiento del programa están basadas en su compilación y ejecución siguiendo el `Makefile`. Además, están también explicadas en el README.

- Para compilar el programa:

  ```
  make compilar
  ```

- Para lanzar el programa:

  ```
  make MaximoHipercubo PROCESOS=numero_procesos DIMENSIONES=numero_dimensiones
  ```

  - Siendo `numero_procesos` el número de procesos que se desean lanzar y `numero_dimensiones` las dimensiones de la red hipercubo que vamos a utilizar.

- Para eliminar el archivo de ejecución:

  ```
  make limpiar
  ```

Nota: Siempre se debe ejecutar al menos un proceso más que el número de nodos de la red.

3. Conclusiones

Las principal conclusión que he obtenido de la realización de esta práctica es que el uso de topologías de computación distribuida y una interfaz de paso de mensajes

puede aumentar en gran medida el rendimiento y la rapidez de una operación siguiendo unos patrones de diseño adecuados.

En este caso es algo más o menos trivial como el cálculo del número mayor y menor de una red pero sirve para observar el funcionamiento de las distintas topologías y los resultados que se obtienen al usar las mismas.

4. Fuentes

Para la realización del programa se ha utilizado la documentación oficial de MPI.

https://www.mpich.org/documentation/guides/

https://www.mpich.org/documentation/manpages/