COMPUTACIÓN PARALELA Y CÁLCULO DISTRIBUIDO
Proyecto Final

Vojtěch Obhlídal

1 de julio de 2022

La tarea del proyecto final era programar un script que simulara el conocido juego BINGO. Para ello, necesitábamos utilizar MPI con **P** procesos, donde el proceso 0 representaba el introductor del juego y los restantes P-1 procesos eran los jugadores.
Para recapitular brevemente las reglas, cada jugador recibe una tarjeta con números antes de comenzar el juego. A continuación, el introductor extrae al azar los números que llama. Los jugadores marcan secuencialmente los números llamados de sus tarjetas (si están en sus tarjetas). Una vez que un jugador ha marcado todos los números de su tarjeta, es el ganador y grita "¡BINGO!".
Nuestro programa funciona de la misma manera. Las cartas se generan aleatoriamente para los jugadores y por defecto hemos limitado el rango de números del 0 al 99. Así, puede ocurrir que algunos de los números de la tarjeta de un jugador se repitan.
En la siguiente sección, describimos brevemente las partes más importantes del código, incluyendo algunas de las funciones que utilizamos.

1. Funciones

En primer lugar, enumeramos las funciones importantes que hemos utilizado repetidamente en la simulación de BINGO. Omitiremos las funciones que sólo envían texto a la consola.

```
void fillList(int* data, int n, int range)
{
    for(int j = 0; j < n; j++){
        data[j] = rand() % range;
    }
}

int checkBingo(int* data, int size_data, int bingo_lim){
    int num_check = 0;
    int bingo = 0;
    for(int j = 0; j < size_data; j++){
        if(data[j] == bingo_lim){
            num_check++;
        }
    }
    if(num_check == size_data){
        bingo = 1;
    }

    return bingo;
}
```

Primera función `fillList` crea la carta con números para el jugador. Esta tarjeta está llena de números de 0 a **range** − 1.
La segunda función `checkBingo` comprueba si el jugador ya ha llegado a BINGO. Cuando el introductor llama a un número, nuestro programa cambia este número por el valor **bingo_lim**. Esta función comprueba entonces si sólo hay valores **bingo_lim** en la tarjeta.

2. Proceso 0

A continuación, nos centraremos en el proceso 0, que funciona como el introductor y es prácticamente la parte más importante de nuestro programa.

Primero asignamos un campo de datos para cada jugador, lo llenamos con números y lo enviamos a los otros procesos.

```
//process 0
if(mpi_id == 0){
    //create cards with numbers and send them to players
    for(int i = 1; i < num_procs; i++){
        int *card_i = (int*) malloc(sizeof(int)*list_size);
        fillList(card_i,list_size,range);
        printCard(card_i, list_size,i);
        MPI_Send(card_i,list_size,MPI_INT,i,0,MPI_COMM_WORLD);
    }
```

A continuación, creamos un campo de datos con todos los números posibles que se pueden elegir. Usando la función `random_shuffle` del paquete `<algorithm>` permutamos estos números y luego simplemente los tomamos uno por uno, ya que hemos barajado los números.

```
    int in_progress = 1;
    // numbers that can be chosen
    int *numbers = (int*) malloc(sizeof(int)*range);
        for(int j = 0; j < range; j++){
            numbers[j] = j;
        }
    random_shuffle(numbers, numbers + range);
    int ind = 0;
```

En la última parte del bucle while, seleccionamos gradualmente los números a enviar a los otros procesos. A continuación, esperamos una respuesta sobre ellos para ver si el juego sigue en marcha y ninguno de ellos ha completado el BINGO. Si el juego continúa, notificamos a los jugadores enviando un valor binario SÍ/NO y el bucle while continúa. Si alguno de los jugadores ha conseguido el BINGO, se registra el ganador, los jugadores reciben un mensaje de que el juego ha terminado y el bucle se ha terminado.

```
    int winner;
    printf("\nSoy el proceso 0 y he elegido siguientes numeros: \n");
    while(in_progress == 1){
        //game progress
        int rn_num = numbers[ind];
        ind++;

        for(int i = 1; i < num_procs; i++){
            MPI_Send(&rn_num,1,MPI_INT,i,0,MPI_COMM_WORLD);
        }
        int stop_game = 0;
        for(int i = 1; i < num_procs; i++){
            int rec_progress;
            MPI_Recv(&rec_progress,1,MPI_INT,i,0,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
            if (rec_progress == 1){
            } else{
                winner = i;
                stop_game++;
            }
        }
        if (stop_game > 0){
            in_progress = 0;
            printNums(numbers,ind);
        }
        for(int i = 1; i < num_procs; i++){
            MPI_Send(&in_progress,1,MPI_INT,i,0,MPI_COMM_WORLD);
        }
    }
    printf("\n Ha ganado el proceso %d, fin de la partida.\n",winner);
}
```

3. Otros procesos

A continuación, nos fijamos en los jugadores individuales que están representados con los otros procesos.

El proceso primero prepara un campo de datos y almacena la tarjeta numérica que recibió del proceso 0.

```
} else if (mpi_id > 0 && mpi_id < num_procs){
    int *card_player = (int*) malloc(sizeof(int)*list_size);
    MPI_Recv(card_player,list_size,MPI_INT,0,0,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
```

A continuación, en el bucle while, aceptamos los números extraídos del proceso 0. Al escanear la tarjeta del jugador, marcamos todos los números coincidentes.

```
    int prog = 1;
    while(prog == 1){
        //get number from 0 and check the card
        int num_choice;
        MPI_Recv(&num_choice,1,MPI_INT,0,0,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
        for(int i = 0; i < list_size; i++){
            if(card_player[i] == num_choice){
                card_player[i] = range;
            }
        }
```

A continuación, hay que comprobar si el proceso ha llegado a BINGO utilizando la función `checkBingo`. A continuación, alerta al proceso 0 con una respuesta binaria. En el caso del BINGO, el proceso imprime un texto anunciando su victoria.

```
        int bingo = checkBingo(card_player, list_size, range);
        int in_prog = 1 - bingo;
        MPI_Send(&in_prog,1,MPI_INT,0,0,MPI_COMM_WORLD);
        if(bingo == 1){
            printf("\n Soy el %d... Bingo!\n",mpi_id);
        }
```

En la última parte, el proceso sólo recibe un mensaje binario del puerto 0 para indicar si el juego continúa. Si pierde, imprime un texto indicando la pérdida.

```
        int prog_resp;
        MPI_Recv(&prog_resp,1,MPI_INT,0,0,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
        prog = prog_resp;


        if(bingo == 0 && prog == 0){
            printf("\n Soy el %d... he perdido esta vez.\n",mpi_id);
        }
    }
}
```