Práctica 1

Implementación del algoritmo de Floyd para obtener el camino más corto.

Jesús Ángel González Novez
76440070F

Tabla de contenidos

- Solución 1D
  - Función floyd
  - Modificaciones sobre la clase Graph
  - Medidas de tiempo para 1D

Solución 1D

Para inicializar el entorno lo primero que debemos realizar MPI_Init y justo a continuación usar las funciones MPI_Comm_size y MPI_Comm_rank relativas al número de procesos, el comunicador y la id de cada proceso.

```
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
MPI_Comm_rank(MPI_COMM_WORLD, &id);
```

A continuación tenemos una serie de variables que nos servirán para indicar el número de vértices, el donde debe comenzar cada proceso en el bucle, el número de filas, y el tamaño del bloque. Además se crea un array para hacer copia del array que tiene internamente el tipo Graph.

```
int nverts,mystart,nfilas,blockSize;;
int * matrizGlobal;
```

Estas inicializaciones las realiza el proceso 0.

```
if(id==0){
  G.lee(argv[1]);
  nverts=G.vertices;
  cout << "EL Grafo de entrada es:"<<endl;
  G.imprime(false);
  matrizGlobal = G.getMatriz();
}
```

Es necesario que todos los procesos conozcan el número de vértices(nverts) por ello hacemos Broadcast.

```
MPI_Bcast(&nverts, 1, MPI_INT, 0, MPI_COMM_WORLD);
```

Una vez conocen ese valor ya podemos asignarle a cada uno los valores mystart, nfilas,blocksize.

```
void calculaFilas(int & mystart,int & nfilas,const int & nverts,const int & nprocs,int & blockSize,const int & id){
   mystart = 0;
   nfilas = nverts / nprocs;
   blockSize = nverts%nprocs;

   if(blockSize>id){
      nfilas++;
      mystart = id*nfilas;
   }
   else{
      mystart = id*nfilas+blockSize;
   }
}
```

El siguiente paso es repartir esa matrizGlobal en pedazos equitativos para cada proceso, de forma que usaremos la función MPI_Scatter que sirve justamente para eso, también necesitamos declarar un array matrizLocal para cada proceso.

```
int *matrizLocal = new int[nverts*nfilas];
MPI_Scatter(matrizGlobal, nfilas*nverts, MPI_INT, matrizLocal, nfilas*nverts, MPI_INT, 0,
MPI_COMM_WORLD);
```

Dado que necesitamos asegurarnos que cada proceso recibió su "porción" pondremos una barrera con MPI_Barrier.

```
MPI_Barrier(MPI_COMM_WORLD);
```

Lo siguiente es ya ejecutar el algoritmo principal, del cual mediremos su tiempo de ejecución para lo que usaremos la función MPI_Wtime.

```
double t=MPI_Wtime();
floyd(nverts,id,nfilas,mystart,matrizLocal);
t=MPI_Wtime()-t;
```

Función floyd

En cada iteración k del algoritmo necesitamos que el proceso actual difunda la fila k, por ello haremos un Broadcast, tras acabar el algoritmo (que por el resto no difiere del original) los datos se irán guardando en una matriz local a cada proceso.

```
void floyd(const int & nverts, const int & id, const int & nfilas, const int & mystart, int * matrizLocal){
   int *filak = new int[nverts];
   int vijk;
   for(int k=0; k<nverts; k++){
      if (id == k/nfilas)
        for (int i=0; i<nverts; i++)
          filak[i] = matrizLocal[(k%nfilas)*nverts+i];
      MPI_Bcast(filak, nverts, MPI_INT, k/nfilas, MPI_COMM_WORLD);
      for(int i=0; i<nfilas; i++){
         for (int j=0; j<nverts; j++){
            if (mystart+i!=j && mystart+i!=k && j!=k){
               vijk = matrizLocal[i*nverts+k] + filak[j];
               vijk = min(vijk,matrizLocal[i*nverts+j]);
               matrizLocal[i*nverts+j] = vijk;
            }
         }
      }
   }
}
```

Tras acabar el algoritmo es necesario recolectar todos los datos en la matriz global que se declaró al principio del código, para ello tenemos la función MPI_Gather.

```
MPI_Gather(matrizLocal, nfilas*nverts, MPI_INT, matrizGlobal, nfilas*nverts, MPI_INT, 0, MPI_COMM_WORLD);
```

Hecho esto ya tendríamos calculado todo, ahora solo queda mostrar los resultados, por comodidad se ha ampliado la funcionalidad de la clase Graph permitiéndonos copiar un array sobre su array interno y así poder usar el método imprime()

```
if(id==0){
	cout << endl<<"EL Grafo con las distancias de los caminos más cortos es:"<<endl<<endl;
	G.setMatriz(matrizGlobal);
	G.imprime(true);
	cout<< "\nTiempo gastado = " << t << endl;
}
```

Finalmente debemos llamar a la función MPI_Finalize como en cualquier software que use la librería MPI.

```
MPI_Finalize();
```

Modificaciones sobre la clase Graph

Se han añadido dos métodos nuevos:

```
int * Graph::getMatriz(){
   return &A[0];
}
void Graph::setMatriz(int * m){
  A = m;
}
```

Dichos métodos sirven para obtener el array interno de la clase Graph o para sustituirlo por otro externo a la clase.
Además se ha añadido otro método imprime por comodidad para mi uso, en el que mostrará una pequeña anotación dependiendo de si se trata de mostrar el grafo de entrada o bien el resultado final, para alterna entre uno y otro se usa un parámetro booleano y un if/else simple.

```
void Graph::imprime(bool final){
  if(!final){
    cout << "El coste de ir del nodo i al mismo nodo i es 0" << endl;
    cout << "Si un nodo no puede ir directo a otro nodo su coste es INF" << endl;
    cout << "En esta matriz se indica el coste del viaje directo de un nodo a otros" << endl;
    imprime();
  }else{
    cout << "En esta matriz se indica el menor coste posible para ir de un" << endl;
    cout << "nodo a otros" << endl;
    imprime();
  }
}
```

Medidas de tiempo para 1D

El equipo de pruebas ha sido un portatil Intel Dual Core(2xT1600), por tanto al tener 2 núcleos es normal que con P>2 salgan peores resultados que con P=2, es por ello que se tendrá en cuenta esa columna(P=2) y no la P=4 para medir la ganancia.

	P = 1	**P = 2**	**P = 4**	**Ganancia**
N = 4	0.000004053	0.000016928	0.0337272	**0,24**
N = 60	0.0028398	0.00089407	0.048254	**3,17**
N = 240	0.25511	0.0943968	0.146372	**2,70**
N = 1024	15.3236	7.07731	10.998	**2,17**
N = 2048	123.208	63.3733	116.122	**1,94**

Como vemos para valores muy pequeños como puede ser N=4 tarda más el despachador interno de la librería MPI en gestionar los subprocesos que en lo que tarda en sí mismo el algoritmo, sin embargo conforme crece un poco el valor de N vemos que ya si va siendo una ganancia muy a tener en cuenta.