Programación Paralela (2015-2016)
Lenguajes y Sistemas de Información

Grado en Ingeniería Informática

E. T. S. de Ingenierías Informática y de Telecomunicación
Universidad de Granada

Práctica 1: Implementación distribuida de un algoritmo paralelo de datos usando MPI

Francisco Javier Bolívar Lupiáñez

27 de marzo de 2016

Índice

1. Planteamiento **2**
1.1. Algoritmo de Floyd 2
1.1.1. Pseudocódigo 2

2. Solución **2**
2.1. Versión unidimensional 2

Índice de figuras

1. Planteamiento

En esta práctica se llevará a cabo la paralelización del algoritmo Floyd para la búsqueda del camino más corto en un grafo.
Se desarrollarán dos versiones:

- **Unidimensional**: Se repartirán las filas de la matriz a los procesos.
- **Bidimensional**: Se repartirán submatrices de la matriz a los procesos.

1.1. Algoritmo de Floyd

El algoritmo de Floyd deriva una matriz en N pasos (tantos como número de nodos), obteniendo en cada paso una matriz intermedia con el camino más corto entre cada par de nodos.

1.1.1. Pseudocódigo

```
1 I_ij = A;
2 for k = 0 to N-1
3         for i = 0 to N-1
4                 for j = 0 to N-1
5                         I_ij = min(I_ij, I_ik + I_kj);
```

2. Solución

2.1. Versión unidimensional

Para solucionarlo con este enfoque, asumiendo que el tamaño del problema es divisible entre el número de procesos, cada proceso tendrá una matriz local de tamaño $N/P \times N$.

El reparto se realizará por bloques. Es decir. Al primer proceso le corresponderán las primeras N/P filas, al segundo las siguientes... Por ejemplo, si el tamaño del problema es 8 y tenemos 4 procesos, al P_0 le corresponderás las filas 0 y 1, al P_1 las 2 y 3, al P_2 las 4 y 5 y al P_3 las 6 y 7.

En el cálculo de cada submatriz resultado, cada proceso necesitará, en el paso k, la fila k y puede tener suerte y ser suya o no y corresponderle a otro proceso. Entonces debería comunicarse con éste para poder realizar el cálculo. Por tanto, en cada iteración del primer bucle k, el proceso detectará si la fila k le pertenece y si es así, hace un *broadcast* al resto de procesos.

Por tanto, para solucionar el problema, nos basta con un *scatter* para repartir la matriz, un *broadcast* para difundir cada fila k y un *gather* para recolectar la matriz resultado y el único problema que nos podríamos encontrar es el traducir de local a global un índice según lo que se necesite.