ARQUITECTURA DE COMPUTADORES II

PRÁCTICA: Reconocimiento paralelo de patrones en MPI

AUTORES:

David Expósito Singh

Florin Isaila

Daniel Higuero Alonso-Mardones

Javier García Blas

Borja Bergua Guerra

Área de Arquitectura y Tecnología de Computadores

Departamento de Informática

Universidad Carlos III de Madrid

Julio de 2012

PRÁCTICA: Reconocimiento paralelo de patrones en MPI

El objetivo de esta práctica es implementar varias estrategias de particionado y trabajar con la biblioteca MPI.

Descripción

Esta práctica consiste en la implementación paralela de un algoritmo simplificado de reconocimiento de patrones en imágenes.

Se parte del programa ya implementado llamado genera_bmp cuya función es:

Generar una imagen en formato bmp con el fondo blanco, que contiene n objetos negros. Estos objetos serán cuadrados de distinto tamaño.

Generar un fichero donde se almacena cada uno de las figuras generadas. El formato del fichero es el siguiente:

coordX0 coordY0

coordX1 coordY1

...

coordXn coordYn

El programa genera dos distribuciones distintas de objetos dentro de la imagen:

Uniforme (las figuras son distribuidas uniformemente por la imagen).

Geométrica (la parte superior de la imagen es la más densa y la parte inferior la menos densa).

El objetivo de la práctica será desarrollar un programa paralelo (haciendo uso de la biblioteca MPI), que ejecute un patrón de reconocimiento de patrones. Para ello se aplicará una técnica de “fuerza bruta”: se recorrerá cada pixel de la imagen en busca de las figuras. Para ello, se dividirá la imagen en distintos cuadrantes, y de este modo, repartir la carga de trabajo entre todos los procesos.

La solución del problema se plantea de la siguiente forma:

El proceso 0 lee la imagen pasada como argumento.

El proceso 0 hace llegar la totalidad de la imagen al resto de procesos.

El proceso 0 calcula el número y coordenadas de cada uno de los cuadrantes necesarios para procesar la imagen. En el caso de ejecutar la aplicación con un particionado estático, se dividirá la imagen entre los distintos procesos. En caso contrario, el tamaño del cuadrante será definido como un argumento de la aplicación (tamaño de grano del problema).

El resto de procesos ejecutan el proceso de reconocimiento de figuras.

Una vez que el proceso de reconocimiento ha finalizado, se envía al proceso 0 el resultado (coordenadas y tipo de la figuras reconocidas).

Si hay más cuadrantes que procesar, el proceso 0 envía al último proceso que finalizó su tarea el siguiente cuadrante. Este proceso continua hasta completar todos los cuadrantes de la imagen.

Para el desarrollo de la práctica se recomienda seguir los siguientes pasos:

Escribir una versión secuencial del algoritmo (necesaria para el resto de prácticas).

Paralelizar el algoritmo desarrollado, particionando la imagen entre los p procesos (particionamiento estático) o mediante los n cuadrantes (particionamiento dinámico).

Se deberá implementar una solución paralela que funcione independientemente del número de procesos con que se ejecute. Con un solo proceso su funcionamiento será igual que en su versión secuencial, y con n procesos su funcionamiento será paralelo.

Para la versión paralela del programa, el particionado de la imagen se deberá hacer al principio del programa, únicamente por el proceso padre (proceso con el identificador 0). Al finalizar el proceso de

computo, el resultado deberá ser recolectado y enviado al proceso padre, que será encargado de escribir el resultado en el fichero de salida.

El programa desarrollado tomará como argumentos los siguientes parámetros, y en el siguiente orden:

El nombre de fichero que contiene la imagen

Un número entero n (n=1,2) que indica que tipo de particionado se debe usar:

Si n=1: cuadrantes con el mismo número de filasy columnas.

Si n=2: cuadrantes dinámicos.

El tamaño de grano en caso de ejecutar un particionado dinámico.

Un fichero de salida en el que se escribirá los objetos reconocidos mediante el formato (x y), donde x e y son las coordenadas del centro de la figura.

De esta forma, la invocación al programa será:

./bmp_mpi <fichero_imagen_original> <tipo_particionado> <granoX> <granoY> <fichero_centros_salida>

Requisitos

Para aprobar la práctica es necesario cumplir los siguientes requisitos:

Todos los accesos de E/S deberán ser realizados únicamente por el proceso 0, es decir, el proceso 0 será el único proceso que podrá acceder a los ficheros o a la pantalla.

Se prohibe el uso de la primitiva gettimeofday para la obtención de tiempos.

Se tomarán medidas de rendimiento sobre 1, 4, 8, 12 y 16 procesos y 2 imágenes distintas: una con una distribución uniforme y otra con una distribución geométrica. Para evaluación de la aplicación se recomienda usar los recursos proporcionados para la elaboracion de la práctica (aulas o cluster).

Es necesario hacer uso de al menos un tipo de dato complejo (MPI_Type_vector por ejemplo).

La solución implementada debe usar al menos 2 tipos distintos de llamadas colectivas (MPI_Bcast, MPI_Reduce, MPI_Scatter, etc).

Entregas

El código será entregado mediante un único fichero llamado bmp_mpi.c (el enlace para entregar las prácticas será publicado brevemente).

La memoria tendrá que contener los siguientes apartados:

Descripción de la metodología de paralelización basada en cuatro fases: descomposición, asignación, orquestación y mapeo.

Gráficas con la medida de tiempos y mejora del rendimiento (speedup) con los distintos tipos de particionado y varios tamaños de grano.

Descomposición aproximada del tiempo total de ejecución en: tiempo de computación y tiempo de comunicación.

Interpretación de las gráficas (muy importante).

Interpretación del análisis de rendimiento usando la herramienta de visualización jumpshot (para el conjunto de medidas que se consideren relevantes).

Conclusiones.

Normas

Las prácticas que no compilen o que no se ajusten a la funcionalidad, obtendrán una calificación de 0.

Un programa no comentado, obtendrá una calificación de 0.

Un programa que no se ajuste a los requisitos planteados, obtendrá una calificación de 0.

Pistas

1. Para la toma de tiempos se hará uso de la función MPI_Wtime():

```
double tstart = MPI_Wtime();
```

código fuente

```
double tfinish = MPI_Wtime();
double TotalTime = tfinish - tstart;
```

2. En el momento de particionar la imagen, es posible encontrar que un mismo objeto pertenezca a dos cuadrantes distintos. Para solucionar este problema, se permite invadir ligeramente otros cuadrantes.

3. Se propone definir tipo de datos complejos para el manejo de vectores y coordenadas.

4. Un mecanismo para el reconocimiento de imágenes:

```
x = coordX;
y = coordY;
....
black    = TRUE;
displ    = 0;
prev_cnt = 0;
figura   = FALSE;
while (black)
{
   cnt = 0;
   for (i = x-displ; i <= x+displ; i++)
      for(j = y-displ; j <= y+displ; j++)
         if (imagen[i][j] == BLACK)
            cnt++;
   if (cnt == (1+displ*2)^2) {
      black = TRUE;
      prev_cnt = cnt;
   }
   else
      black = FALSE;
   displ++;
```

```
}
if ((cont == prev_cnt) && (displ > 2))// Hay un borde blanco en la figura.
    figura = TRUE;
```