UNIVERSIDAD DE GRANADA

Práctica 3: Detector de bordes para imágenes

Arquitectura y Computación de altas prestaciones

Luis González Romero

23 de marzo de 2019

Índice

Índice de figuras

1. Enunciado

- Desarrollar un programa paralelo con *MPI* que sea escalable partiendo de un algoritmo secuencial previamente contrastado con la profesora.
- Compilarlo y ejecutarlo en el aula de prácticas y en *atcgrid.ugr.es* con una carga homogénea por proceso, comprobando la evolución del tiempo de ejecución.
- Representar gráficamente el tiempo de ejecución total y la ganancia en velocidad en función del número de PCs.

2. Selección del algoritmo

En mi caso decidí usar un detector de bordes para imágenes, concretamente el operador Sobel, el cual aplica una máscara(calculada con los gradientes) a cada píxel de la imagen:

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \quad G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Figura 1: Gradientes Sobel

Así calculamos el gradiente de la intensidad de una imagen en cada píxel. Obteniendo, para cada punto, la magnitud del mayor cambio posible y la dirección de éste.

El resultado muestra cómo de abruptamente o suavemente cambia una imagen en cada punto, interpretando si representa un borde y la orientación a la que tiende el borde.

Figura 2: Imagen de entrada

Figura 3: Imagen de salida

3. Implementación secuencial

Para implementarlo, decidí usar imágenes *Mat* con la biblioteca *OpenCV*

```
for(int y = 0; y < src.rows; y++){
  for(int x = 0; x < src.cols; x++){
    gx = x_gradient(src, x, y);
    gy = y_gradient(src, x, y);
    sum = abs(gx) + abs(gy);
    sum = sum > 255 ? 255:sum;
    sum = sum < 0 ? 0 : sum;
    dst.at<uchar>(y,x) = sum;
  }
}
```

Figura 4: Tratado secuencial de la imagen

En los métodos para calcular los gradientes, se tienen en cuenta casos especiales para evitar fallos de acceso a memoria.

En esta versión secuencial, lo único que hace el programa es abrir, tratar la imagen y guardarla con el prefijo "*sobel-*".

4. Implementación con MPI

Partiendo del código secuencial, con un buffer intermediario comparto el fragmento de la imagen que le corresponde a cada *rank* del *grupo de procesos* con los que se ejecute el programa.

Cuando cada *rank* tiene los datos necesarios, realizan el siguiente tratado y termina haciendo un *gathering* de las soluciones parciales para así guardar esa imagen final obtenida al igual que en la versión secuencial.

```
for(int y = 0; y < rows_av; y++){
  for(int x = 0; x < cols  ; x++){
    ip_gx = x_gradient(img, x, y+rows_extra-1);
    ip_gy = y_gradient(img, x, y+rows_extra-1);
    sum = abs(ip_gx) + abs(ip_gy);
    sum = sum > 255 ? 255:sum;
    sum = sum < 0 ? 0 : sum;
    picAux[y*cols+x] = sum;
  }
}
MPI_Gather(picAux, cols*rows_av, MPI_UNSIGNED_CHAR, pic, cols*rows_av, MPI_UNSIGNED_CHAR, 0, MPI_COMM_WORLD);
```

Figura 5: Tratado paralelo de la imagen

5. Tiempo de ejecución y ganancia

Para obtener los tiempos de ejecución se han hecho distintas mediciones y se ha obtenido: la media de cada caso de las configuraciones y la dispersión estándar de estas.

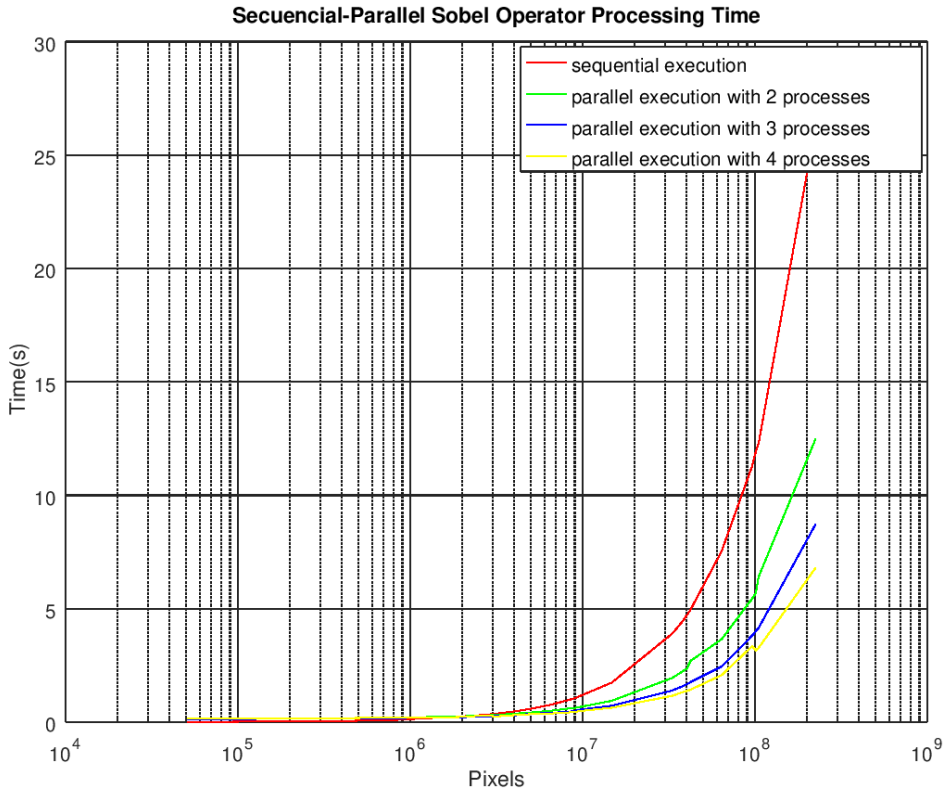

Figura 6: Tiempos de ejecución de todas las configuraciones

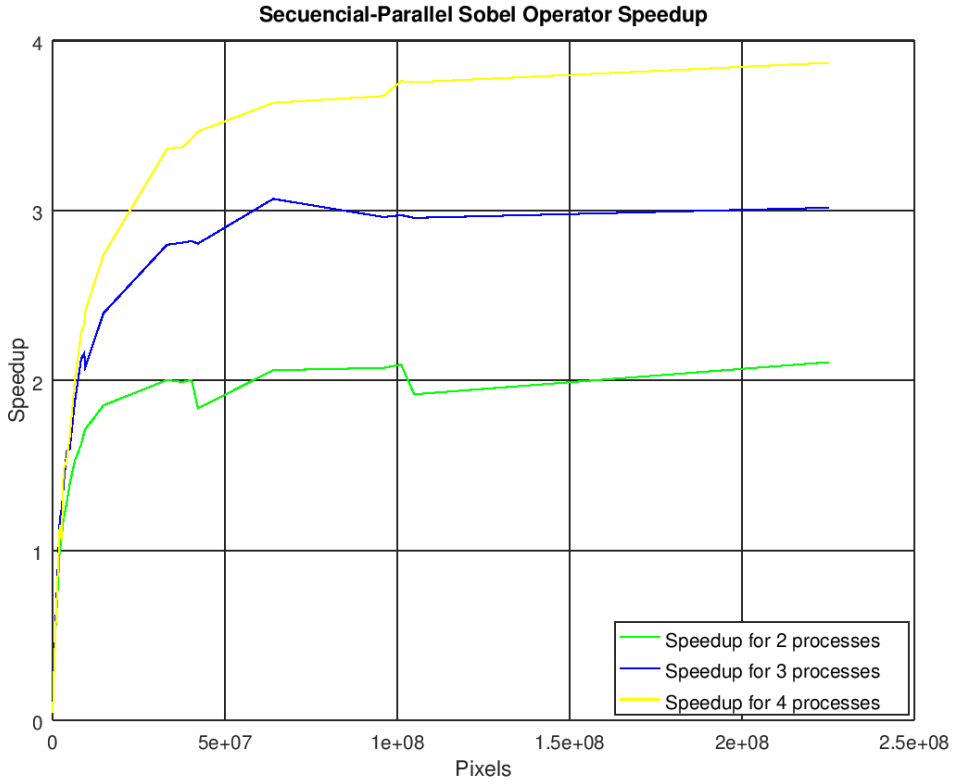

Figura 7: Ganancia para todas las configuraciones

La ganancia tiende al igual que en la anterior práctica a *np*, la diferencia es que ha sido ejecutado en mi ordenador personal. Se ven afectados los tiempos al no ser ejecutados en los nodos del aula conectados en estrella. He usado hasta *np*=4, que son los cores físicos que tiene mi laptop.

Todos los tiempos obtenidos junto a la media y desviación de estos, están en los ficheros *calculos-estadisticos-seq.txt* y *calculos-estadisticos-mpi.txt*

6. Conclusión

La conclusión es similar a la de la *práctica 2*, pero en mi caso se ve un poco afectada al no haber sido ejecutada en los nodos del aula o en los de *atcgrid*(por la falta de la biblioteca *OpenCV*).