Procesamiento HPC en Sistema de Reciclaje inteligente

Facal Ernesto
Oria Joaquín Ariel
Marsón Tomás
Schiro Andrés

Universidad Nacional de La Matanza,
Departamento de Ingeniería e Investigaciones Tecnológicas,
Florencio Varela 1903 - San Justo, Argentina

Resumen. El objetivo de esta investigación se basa en el análisis, manipulación y reconocimiento de imágenes para un sistema reciclador de basura integrado. Se utiliza un algoritmo para la detección de formas de imágenes haciendo uso de la Distancia de Hausdorff y se presenta una implementación en forma paralela utilizando OpenMP.

Palabras claves: OpenMP, Paralelismo, Hausdorff

Introducción

La investigación que se desarrolla en este documento está aplicada a un Sistema de reciclaje de materiales inteligente integrado. Este sistema cuenta con una cámara que monitorea el recipiente donde se colocan los materiales, dicha cámara proveerá al sistema las imágenes a procesar. Una vez identificado el material el sistema procederá a ubicar el material en el cesto indicado según su clasificación.

Para solucionar el problema de reconocimiento de imágenes o de elementos en una imagen de forma eficiente se propone utilizar la Distancia de Hausdorff.

Este concepto mide el nivel de discrepancia entre dos grupos de puntos de un mismo espacio métrico. Este algoritmo propuesto ofrece la posibilidad de paralelizar la parte mas pesada de la computación.

Dado que para obtener una aproximación fiable acerca del tipo de material debemos compararlo con un número considerable de imágenes distintas de ese tipo, va a haber un gran procesamiento de información para cada comparación antes de llegar a un resultado final. En esta etapa comienza a surgir la idea de paralelizar estas tareas.

Para la paralelización se utiliza OpenMP, una API ideada para la programación de tareas multiproceso con memoria compartida. La API consiste en una agrupación de

directivas para el compilador, rutinas de biblioteca y variables de entorno que permiten modificar el comportamiento de la ejecución de un programa. El funcionamiento básico de OpenMP, se basa en la idea de fork-join, donde el hilo de ejecución maestro se divide en diversos hilos.

Desarrollo

Método Hausdorff

La idea tras la distancia Hausdorff es que puede determinarse una distancia d que indica el grado de disparidad entre conjuntos de puntos pertenecientes al mismo espacio. Si la distancia vale 0 ambos grupos serán el mismo, y cuanto mayor sea el valor, mas diferentes entre sí. Esta propiedad puede usarse como base para el análisis de imágenes en formato mapa de bits, ya que, en este caso, las coordenadas de los pixeles forman los elementos del espacio y es posible comparar grupos que pertenezcan a éste. Es común aplicar la distancia de Hausdorff para obtener la medida de similitud entre dos contornos.

La definición formal de la distancia Hausdorff viene dada por la siguiente expresión matemática:

Sean $A = \{a_1, ..., a_p\}$ y $B = \{b_1, ..., b_q\}$ dos conjuntos finitos no vacíos, subconjuntos de un espacio métrico (M, d), se define la distancia Hausdorff $d_H (A, B)$

$$d_H (A, B) = max \{h (A, B), h (B, A)\}$$

Donde

$$h (A, B) = max\, min||a - b||$$

$$h (B, A) = max\, min||b - a||$$

Con a ∈ A, b ∈ B y $\| \cdot \|$ es una norma entre los puntos de A y B.

Implementación

Como herramienta de paralelización se ha decidido usar OpenMP porque ofrece una interfaz sencilla que permite realizar la paralelización introduciendo pocos cambios sobre la implementación secuencial. Para la solución del problema es necesario hacer uso de una herramienta de apoyo que nos permita leer y manipular imágenes. En este caso se ha elegido OpenCV, puesto que son unas bibliotecas basadas en C/C++, probadas, con amplia documentación y, en su mayor parte, optimizadas, que nos permiten realizar con sencillez las operaciones que son necesarias para el proyecto.

Primeramente, se transforman los mapas de bits en un formato que permita su manipulación desde OpenCV, para luego poder aprovechar la funcionalidad de esta biblioteca para detectar los bordes de las imágenes de entrada. Para facilitar el proceso de detección de contornos las imágenes se han convertido a escala de grises utilizando funciones brindadas por OpenCV.



Procesamiento

El sistema de reciclaje cuenta con una platafoma donde se colocara el residuo, dicha plataforma es monitoreada por una camara que obtendra una imagen del material a reciclar. Una vez obtenida la imagen, se procedera a compararla con las imágenes guardadas en una base de datos utilizando el método Distancia de Hausdorff. Se estiman unas 500 imágenes distintas por cada tipo de material (vidrio, papel, plástico, metal, otros). A través de los resultados obtenidos se clasificará al residuo en uno de los distintos tipos de materiales. Por último, ya obtenida la clasificación, la plataforma colocara el objeto en el contenedor correspondiente a través de dos servomotores que controlaran los ejes rotación y giro de la caja.

Explicacion del algoritmo

La función consiste en dos bucles anidados, el primero recorre los píxeles de los contornos de la primera imagen, el segundo, para cada píxel, recorre todos los píxeles de los contornos de la segunda imagen y calculará las distancias entre estos y se guardará siempre la menor. Al finalizar el recorrido para un punto dado de la primera imagen seleccionará aquella distancia que sea mayor entre la nueva y el máximo

anterior. Este proceso se realiza en el sentido imagen1 → imagen2 y luego en el sentido imagen1 ← imagen2. Finalmente devolverá el mayor valor de entre ambas.

```
double hausdorff_distance(vector<coordenadas>&template, vector<coordenadas>
&Image) {
tsize = template.size();
msize = Image.size();
#pragma omp parallel {
#pragma omp single {
nthreads = omp_get_num_threads ();
distA = new double[nthreads];
distB = new double[nthreads];
for (int i = 0; i < nthreads; i++) {
        distA [i] = -1;
        distB [i] = -1;
}}}
// Cálculo de la distancia del conjunto A al B
#pragma omp parallel for Schedule(static)
for (int t = 0; t<tsize; t++) {
int tid = omp_get_thread_num ();
double temp = 0, dist = -1, aux;
dist = distance_h1(template[t].i,template[t].j,Image[0].i,Image[0].j);
for (int k = 1; k<msize;k++) {
temp = distance_h1(template[t].i,template[t].j,Image[k].i,Image[k].j);
dist= std::min(dist,temp);
}
distA [tid] = std:max(dist,distA[tid ]);
aux = std::max(dist,aux);
}
// Cálculo de la distancia del conjunto B al A
#pragma omp parallel for Schedule(static)

for(int t = 0; t<msize;t++){
int tid = omp_get_thread_num();
double temp = 0, dist = -1, aux;
dist = distance_h1(Image[t].i,Image[t].j,template[0].i,template[0].j);
for(int k = 1; k<tsize;k++){
temp = distance_h1(Image[t].i,Image[t].j,template[k].i,template[k].j);
```

```
dist = std::min(dist,temp);
}
distB[ tid ] = std::max(dist,distB[tid ]);
aux = std::max(dist,aux);
}

for (int i = 0; i < nthreads;i++) distance_A = std::max(distA[i],distance_A);
for (int i = 0; i < nthreads; i++) distance_B = std::max(distB[i],distance_B);
delete distA;
delete distB;
return std::max(distance_A,distance_B);
}
```

Pruebas que pueden realizarse

Una de las pruebas que pueden realizarse es implementar el algoritmo con y sin el empleo de OpenMP, para así demostrar la diferencia en los tiempos de ejecución entre la versión secuencial y la optimizada paralelizable con OpenMP.

Los casos de prueba que se pueden realizar son colocando residuos en la plataforma del sistema de reciclaje, previamente habiendo cargado imágenes en la base de datos de ese material para que el sistema pueda comparar y realizar el procesamiento.

Conclusiones

Podemos ver que a través del uso de la propiedad de la distancia de Hausdorff es posible realizar un reconocimiento de patrones en imágenes y de esta forma poder identificar los distintos tipos de materiales para que luego el sistema integrado una vez ya clasificado el material lo recicle según corresponda.

Se utilizo OpenMP para ejecutar el algoritmo comparando paralelamente la imagen del objeto con las de la base de datos y así aumentar notablemente su rendimiento.

Referencias

1. OpenMP versus Threading in C/C++ (2014)

2. Measuring synchronization and cheduling overheads in openMp (2016)

3. Paralel computing using openMp (2017)

4. Comparación de Huellas Dactilares Usando la Distancia Hausdorff (2008)