Utilizacion de procesamiento paralelo para optimizar las tareas de Reconocimiento Optico de Caracteres

Tamashiro Santiago, Vogel Facundo, Robles Wagner Sebastián,
Gómez Markowicz Federico, Gronski Sebastián

[1]Universidad Nacional de La Matanza,
Departamento de Ingeniería e Investigaciones Tecnológicas,
Florencio Varela 1903 - San Justo, Argentina
stamashiro@unlam.edu.ar
fvogel@unlam.edu.ar
sebastiangronski@gmail.com
Fede.markoo@gmail.com
seba.95.95@hotmail.com

Resumen.
El objetivo de el presente documento será explorar una posible aplicacion practica de la tecnologia de procesamiento en paralelo utilizada en el campo de la computacion de altas prestaciones (HPC) para optimizar la tarea de reconocimiento optico de caracteres (OCR) en el entorno de desarrollo de aplicaciones moviles.

Palabras claves: Reconocimiento optico de caracteres, Computacion de altas prestaciones, procesamiento paralelo, desarrollo movil, Android

1 Introducción

La presente investigación tendrá como fin evaluar la posibilidad de implementar la tecnología de reconocimiento de caracteres ópticos para nuestro proyecto denominado AsistenteDeCocina (que consiste en movilizar los ingredientes contenidos en un recipiente hacia otro que los colecta, indicando la cantidad de peso que se desea extraer de cada uno por la aplicación Android).

A través del procesamiento óptico de una factura de compra que el usuario cargue tomándole una foto, dichos productos se cargarán en la aplicación sin la necesidad de que el usuario manualmente ingrese cada ingrediente luego de sus compras diarias. Esta automatización de cargas de ingredientes deberá requerir de un módulo de GPU que permita paralelizar las tareas de análisis de una imagen.

Algunas razones por las que se implenmentaría GPU son que las operaciones de procesamiento comunes de imágenes iteran de pixel a pixel, calculan usando los valores de dichos píxeles y devuelven el resultado. Dicho procesamiento de pixeles puede ser en paralelo con un módulo GPU. Por otro lado, cuando procesamos una

imagen necesitamos rápido acceso a los valores de los píxeles, y GPU es diseñado para fines gráficos (el hardware para acceder y manipulizar pixeles es muy óptimo). Por esto, GPU nos proporciona muchas ventajas sobre la secuencialidad de la CPU ya que con ésta tecnologia tenemos operaciones que se pueden paralelizar.

2 Desarrollo

Primero que nada, necesitamos disponer de un dispositivo Android con cámara digital para obtener la imagen a procesar de la factura de compra, para así tratarla y cargar el resultado en la lista de ingredientes de la aplicación.

La aplicación tendrá una opción de "Cargar Ingredientes" en donde al seleccionarla se abrirá la cámara y permitirá tomar la imagen. El usuario, al tomar la imagen de la factura de compra que desea cargar en la aplicación, confirmará que desea cargar esos ingredientes y así la imagen es procesada en el módulo GPU.

Utilizaríamos CUDA C (extensión de programación en C usado para interfaces y programas con NVIDIA GPUs) para el código de manejo de la imagen, donde se buscarán patrones vectoriales conocidos previamente para reconocer caracteres y así lograr convertir en texto los símbolos detectados en la imagen.

Se buscará explotar el paralelismo inherente a la aplicación: el número de threads óptimo y su despliegue en bloques. Al obtener la imagen, el módulo GPU procesará la misma en diversos threads que van generando los caracteres reconocidos. Así logramos optimizar un uso secuencial que tardaría un tiempo más prolongado y paralelizamos el procesamiento de reconocimiento óptico de la imagen.

Una vez obtenidos los resultados, la aplicación reconocerá los textos procesados y mediante patrones de texto reconocerá que ingrediente es y la cantidad que se desea ingresar (dando por hecho que los tickets de compra tendrán los pesos de cada uno), y los cargará en la aplicación.

Gracias a todo este procesamiento, el usuario se ahorrará la tarea de cargar todos los nuevos ingredientes que haya comprado en la aplicación, y así podrá usar dichas cantidades en las próximas recetas.

3 Explicación del algoritmo.

Primero, deberemos normalizar la imagen a un tamaño y resolución. Luego, el módulo de GPU recibirá dicha imagen y comienza su ejecución.

El proceso de Reconocimiento Óptico de Caracteres requiere de 5 subtareas, donde se busca que cada proceso involucrado explote al máximo el paralelismo.

```
1  //Pseudocodigo de procesamiento de imagen en GPU
2  Cadena generarTexto(Imagen, ancho, alto){
3
4  vec filasEnBlanco[] = chequearSaltosDeLinea(Imagen,ancho,alto);
5  vec columnasEnBlanco[] = chequearLimitesHorizontales(Imagen,ancho,alto),
6
7  Cadena texto = obtenerMatricesYCalcularVector(filasEnBlanco,columnasEnBlanco);
8
9  retornar texto;
10 }
```

Pseudocódigo del procesamiento general.

El proceso de reconocimiento consta de 5 subtareas:

1. Analizar horizontalmente los bits para encontrar las líneas (comúnmente llamados “renglones”: 0 píxeles en blanco indica un espacio entre líneas). Nos da X filas de píxeles.

```
13 //Genera un vector con la cantidad de pixeles de alto, indicando en que pixeles hay saltos de linea
14 chequearSaltosDeLinea(Imagen,ancho,alto){
15 vec[alto] saltosDeLinea;
16 paraCadaFila(alto){
17 todosBlancos = true;
18
19 paraCadaColumna(ancho){
20     siEsNegro(fila,columna) todosBlancos=false;
21 }
22 si(todosBlancos) saltosDeLinea[fila] = true;
23 }
24 retornar saltosDeLinea;
25 }
```

2.Analizar verticalmente los bits para encontrar los “espacios” entre los símbolos (0 píxeles en blanco en un símbolo indica el límite vertical del mismo). Nos da Y líneas de píxeles.

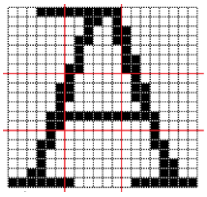
```
27 //Genera un vector con la cantidad de pixeles de ancho, indicando en que pixeles hay horizontalmente de las letras
28 chequearLimitesHorizontales(Imagen,ancho,alto){
29 vec[alto] limitesHorizontales;
30 paraCadaColumna(ancho){
31 todosBlancos = true;
32
33 paraCadaFila(alto){
34     siEsNegro(fila,columna) todosBlancos=false;
35 }
36 si(todosBlancos) limitesHorizontales[fila] = true;
37 }
38
39 retornar limitesHorizontales;
40 }
```

3. Analizar horizontalmente los espacios entre el tope superior de cada símbolo y la distancia que tiene del tope superior del salto de línea más próximo (de las líneas encontradas en paso 1).

4. Contar regiones de 3 x 3 para cada símbolo encontrado, y así producir el vector de densidad global.

Densidad global: está tomada por cada región, que indica la cantidad de píxeles negros sobre los totales.

Así, tomando estos 9 bloques armamos un vector de densidad (cada elemento del vector representa un bloque, y se coloca la densidad calculada por bloque en cada uno).



Ejemplo de división en regiones de 3x3 de un símbolo

5. *Búsqueda para identificar cada símbolo*. Se tiene un conjunto de vectores conocidos, con la densidad global de cada región tomada. Así, el algoritmo compara el vector generado en el punto 4 con los símbolos conocidos, y le asigna el que más se aproxime en tanto a la densidad global.

```
42  obtenerMatricesYCalcularVector(vec[] filasEnBlanco,vec []columnasEnBlanco){
43      cadena palabras; //Inicializamos la cadena que va a llenar con los caracteres.
44
45  paraCadaFilaEnBlanco(tamañoDeFilasEnBlanco){
46      paraCadaColumnaEnBlanco(tamañoDeColumnasEnBlanco){
47          si(EncontróLímiteSuperior)
48              si(EncontróPrimerLímiteHorizontal)
49              continuarHastaInferior y hasta segundo limite horizontal
50      }
51      mat[][] caracter = nueva mat[altoCaracterEncontrado][anchoCaracterEncontrado];
52      vec[] densidad = calcularVectorDensidad(caracter);
53      palabras.concatenar(obtenerCaracterConVector(densidad));
54  }
55      retornar palabras;
56  }
```

Pasos 3, 4 y 5. Es un pseudocódigo de procesamiento al encontrar los caracteres y enviarlos a procesar (calcularVectorDensidad dividirá la matriz encontrada en 3x3 y calculará el vector de densidad con el cálculo especificado).

4 Pruebas que pueden realizarse

Luego de las implementaciones, podremos comprobar si el algoritmo funciona primero con una foto en donde estén bien claros los símbolos de la factura de compra que se suba. Luego, se irán testeando los márgenes que soporta dicho reconocimiento óptico al someter al algoritmo con imágenes donde hay más ruido o donde hayan distintas variaciones de luz, para ver que tan factible puede ser su uso por parte de un usuario en una vida cotidiana sin perder tiempo al buscar el mejor ángulo posible de imagen.

En todas las pruebas se pueden probar con ingredientes que ya existen en las recetas y los que todavía no, y con diferentes pesos. También se puede probar si no existe el peso, como se comporta el algoritmo y someterlo a todas las posibles fallas que puedan ocurrir con un procesamiento de imagen.

5 Conclusiones

En este trabajo, analizamos una posible mejora a nuestro proyecto AsistenteDeCocina a través de un módulo GPU que procese paralelamente imágenes y detecte ópticamente caracteres. Así, podríamos hacer más cómoda la aplicación para el usuario e implementar tecnologías que aceleren el procesamiento de la misma.

En la introducción, pudimos ver un marco general de que trata la idea y cómo vamos a tratar la misma (respondiendo a la pregunta de ***qué*** se desea proponer). Luego, en el desarrollo analizamos las herramientas que usaríamos en caso de implementar esta mejora del proyecto, respondiendo a la pregunta de ***con qué***

desearíamos hacerlo, y en la explicación del algoritmo definimos el ***cómo*** se realizaría.

Esta investigación fue una aproximación con una abstracción general de implementar una tecnología como GPU en un proyecto de sistema embebido, y su implementación requerirá una profundidad en el análisis de las herramientas y el código para el mismo.

Respecto a GPU, hay muchas investigaciones con este tópico, muchos investigadores están desarrollando métodos más efectivos teniendo en cuenta limitaciones y dificultades, y siempre se busca la optimización y tratar de expandir y buscar otras soluciones más allá de la computación secuencial. Hemos encontrado una gran cantidad de trabajos y exposiciones de investigadores que tratan de mejorar lo ya existente, y proponer nuevas soluciones para un mundo en donde la velocidad de procesamiento será clave.

Para un próximo trabajo, sugerimos la implementación de tecnologías como OpenCL que permite un estándar abierto y libre que no dependa de un hardware de un determinado fabricante, como sí lo hace CUDA.

6 Referencias

Zlatanov, Nikola. CUDA and GPU Acceleration of Image Processing. 10.13140/RG.2.1.4801.0004. (2016).
https://www.researchgate.net/publication/297459183_CUDA_and_GPU_Acceleration_of_Image_Processing

Reed, Jeremy, "An Optical Character Recognition Engine for Graphical Processing Units" Theses and Dissertations—Computer. (2016).
https://uknowledge.uky.edu/cs_etds/54

Autorzy. Wieloch, K. , Stokfiszewski, *Time effectiveness optimization* of *cross correlation methods* for *ocr systems* with the *use* of *graphics processing units*. (2015)
http://it.p.lodz.pl/file.php/12/2015-2/jacs-2-2015-wieloch-stokfiszewski.pdf