Identificación de usuarios mediante el uso de reconocimiento de huellas dactilares.

Dal Verme Tomás, De Arias Axel, Figueredo Nicolás Leandro, Quinteiro Lucas, Rodríguez Maximiliano Pablo.

Universidad Nacional de La Matanza,
Departamento de Ingeniería e Investigaciones Tecnológicas,
Florencio Varela 1903 - San Justo, Argentina
tdalverme@gmail.com; maxirodriguez91@gmail.com; axeldearias@gmail.com; nico_1507@live.com.ar; quinteiro.lucas95@gmail.com

Resumen. El objetivo de la siguiente investigación es el de ofrecer el reconocimiento de los distintos usarios de SmartBarman mediante sus huellas dactilares, de forma eficiente y rápida, utilizando técnicas de paralelismo para el procesamiento de la información de las msimas. Para ello se utilizará un algoritmo de extracción de puntos característicos de las huellas (minucias), para su posterior comparación con la base de datos de SmartBarman que contiene las huellas de todos los usuarios, asociadas a sus preferencias de bebidas.

Palabras claves: CUDA, thread, huellas dactilares, IOT, paralelización, HPC.

1 Introducción

El trago perfecto perfecto no existe, pero si existe el mejor trago para cada persona. El principal objetivo de SmartBarman es el de aprender las preferencias de sus usuarios y así lograr preparale su trago ideal. Para lograrlo, cuenta con un sistema embebido encargado de servir la bebida obteniendo información de sus distintos sensores, y una aplicación movil, que se conecta con el mismo a través del protocolo Bluetooth, en donde el usuario podrá consultar su nivel de alcohol en sangre y elegir el trago a preparar. Una vez finalizado, el usuario deberá responder una serie de preguntas que ayudará a SmartBarman a comprender sus preferencias y actualizarlas en la base de datos para que el próximo trago se aproxime mejor a su trago "perfecto". Además, se utilizan sensores del smartphone (sensor de luz, giroscopio, etc.) que serán empleados para enviar instrucciones al sistema embebido, como por ejemplo, empezar a servir o prender LEDs.
Actualmente, la identificación de los usuarios se hace a través del smartphone, es decir, un usuario está asociado a un dispositivo movil, por lo que si cambiara de teléfono, perdería todos sus datos y preferencias asociadas. Este es el principal motivo que impulsa la incorporación del reconocimiento por medio de huellas dactilares, de forma que si un usuario desea utilizar la aplicación en otro smartphone (o en un futuro en otro dispositivo inteligente que soporte la aplicación), pueda contar con sus datos y poder preparar el trago que desee a su medida.
Los sensores de huella digital hoy en día permiten conseguir imágenes de alta reslución, llegando hasta los 2000 DPI en el iPhone por ejemplo. La huella obtenida será procesada y comparada con las presentes en la base de datos (que podrían cer cientos o miles) para poder identificar a la persona y así obtener sus preferencias asociadas. Para el procesamiento de estas imágenes y la gran cantidad de comparaciones (realizando transformaciones matemáticas) a realizar con la base de datos proponemos utilizar la tecnología GPU con la que cuenta el smartphone paralelizando la carga de trabajo en distintos threads, y así lograr una respuesta rápida y eficiente.

2 Desarrollo

Existen principalmente dos métodos para el procesamiento e identificación de huellas dactilares [1]: características locales y patrones globales [Fig.1].

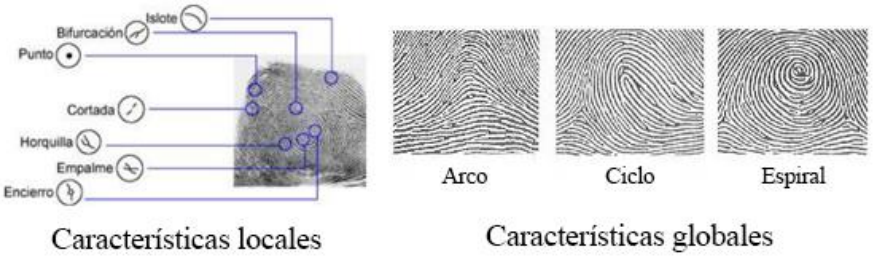

Fig. 1

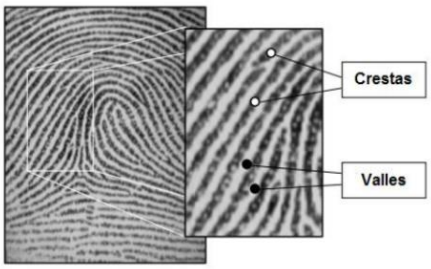

Fig. 2

Las huellas dactilares están compuestas por valles y crestas que forman patrones únicos en cada persona [Fig. 2]. El primer método se basa en el análisis y comparación de lo que se conoce como minucias, que son bifurcaciones o terminaciones de las crestas de las huellas. En cambio, el segundo método realiza una aproximación macroscópica de la huella, observando el flujo de las crestas, clasificándolas en arcos, lazos y espirales.

En nuestro caso, utilizaremos el método de localización y comparación de minucias (características locales) [3]. Cada huella tiene en promedio unas 30 minucias y es posible identificar una huella (con un margen de error pequeño) al compararla con una ingresada y ver que 6 de ellas coinciden [7]. Por eso mismo limitaremos la cantidad de minucias encontradas en una huella a un máximo de 32. El método consta de seis etapas [Fig. 3]:

1. **Adquisición.** En esta etapa se obtiene la huella del usuario a comparar mediante el sensor de huella digital.
2. **Aclaración.** En la práctica, las condiciones de la piel, el ruido de los sensores, la presión del dedo al apoyarlo en el sensor, entre otros aspectos pueden resultar en una mala calidad de la imagen y por lo tanto la detección de falsas minucias. Para ello se utilizan técnicas de mejora de la imagen que permiten recuperar las zonas borrosas e interrumpidas usando la información contextual [1][2]. Además, se realiza el proceso de binarización de la imagen, donde la imagen resultante solo queda formada por unos (pixel blanco) y ceros (pixel negro).
3. **Adelgazamiento.** Durante esta etapa se obtiene una imagen donde las crestas son de un solo pixel de grosor [4].
4. **Extracción de minucias.** Se divide la imagen en ventanas de 3x3, y se calcula el número de píxeles que cruzan el píxel central. Si resulta igual a 7, se dice que hay una minucia del tipo terminación, si es igual a 6 se dice que no hay minucia y si es menor o igual a 5 se dice que existe una minucia del tipo bifurcación [4].
5. **Eliminación de falsas minucias.** Se analizan las minucias encontradas y se descartan aquellas que se cree que son minucias falsas [4].
6. **Reconocimiento.** En esta etapa se comparan las minucias encontradas con cada una de las minucias de las huellas de la base de datos. Ya que la huella puede haberse ingresado en otro ángulo se deben realizar varias rotaciones para encontrar la orientación que genera las mayores similitudes entre ambas [5]. Si al compararla con alguna, resulta que las minucias coinciden (dentro de ciertos umbrales establecidos), el usuario es identificado correctamente.

Fig. 3

Realizar todos estos pasos es muy costoso computacionalmente y es aquí donde entra en escena la tecnología GPU que nos da la posibilidad de realizar multiprocesamiento a través de miles de threads ejecutando paralelamente [6].

3 Explicación del algoritmo

Si bien en todas las etapas hay tareas que pueden paralelizarse, los sensores de huellas digitales actuales ya realizan los pasos (1), (2) y (3) de forma muy rápida. Es por esto que nos centraremos en los pasos (4), (5) y especialmente en el paso (6) para explotar las ventajas de la tecnología GPU, ya que es en estas etapas donde se requiere de un mayor procesamiento de datos ya sea por la complejidad del algoritmo o el volumen de datos.

Debido a que la imagen se trata en bloques de 3x3 en la etapa de extracción de minucias [3], podemos explotar las capacidades que nos brinda la tecnología GPU para paralelizar la carga de trabajo, y realizar el procesamiento más rápidamente. Si las imágenes obtenidas por el sensor fueran de 512x512 píxeles, podríamos organizar la imagen en una grilla bidimensional de 512x512 bloques (uno por píxel) y cada bloque dispuesto en dos dimensiones de 3x3, resultando en 9 hilos por bloque, donde cada thread representaría un píxel vecino al píxel central. Haciendo uso de la directiva ParallelReduction de CUDA [6] podemos reducir cada matriz de 3x3 de forma paralela, debido a que cada píxel es independiente del resto, y de acuerdo con el resultado de la suma obtenido clasificarlo en minucia o no.

Si bien se podrá notar una mejora en la etapa descrita anteriormente, no será mucho mayor a la ya obtenida por el sensor del smartphone. La mayor diferencia y donde intentaremos aprovechar al máximo la tecnología GPU es en la etapa de reconocimiento, ya que si las comparaciones se realizaran en CPU tendríamos que comparar de forma secuencial cada una de las minucias de cada huella de la base de datos hasta obtener la correspondiente. Para cotejar dos huellas hay que comparar todas sus minucias en diferentes ángulos hasta encontrar la orientación que mayores similitudes genera y ahí determinar si las minucias realmente coinciden o no [5]. Para comparar las minucias se toman de a pares (una de la huella ingresada y una de la huella de la base de datos) y se verifica que la distancia y ángulo que forman entre ellas no superen ciertos umbrales ya definidos previamente [4]. Gracias a CUDA podemos realizar las comparaciones en las distintas orientaciones [5] (8 en nuestro caso) de forma paralela ya que son independientes una de otra, y a su vez podemos cotejar una minucia en la huella ingresada con todas las minucias de una huella en la base de datos también al mismo tiempo. Si en la base de datos hubiese 1024 huellas, podemos organizar los CUDA cores en una grilla bidimensional de 1024x8 bloques. Cada fila de bloques contendrá una huella en la base de datos en sus ocho orientaciones predefinidas, conteniendo cada bloque 32x32 threads donde las filas serán las minucias de la huella ingresada y las columnas serán las minucias de la huella de la base de datos [7]. Así estaremos comparando de forma paralela las 8 orientaciones y todas las minucias entre sí, ahorrando un gran tiempo de procesamiento. Una vez realizadas las comparaciones tendremos por bloque la cantidad de

minucias que coincidieron, por lo que debemos buscar el bloque con la máxima cantidad de similitudes y determinar si supera el umbral establecido [7]. Si lo supera, se permitirá el ingreso y se obtendrán las preferencias del usuario. A continuación se presenta el pseudocódigo de cómo se implementaría el algoritmo:

```
/** Obtiene la huella dactilar a comparar **/
Huella huella = leerHuella(SENSOR_HUELLA_DACTILAR);

/** Se mejora la calidad de la imagen usando la información contextual **/
Huella huellaMejorada = mejorarHuella(huella);

/** Convierte la imagen en unos (pixeles blancos) y ceros (píxeles negros) **/
Huella huellaBinaria;
huellaBinaria = binarizar(huellaMejorada, 0, 255);

/** Se ejecuta el algoritmo de adelgazamiento **/
Huella huellaAdelgazada = adelgazarHuella(huellaBinaria);

/** Se sube la grilla de 512x512 bloques a la GPU en bloques de 3x3 threads **/
subirGPU(huellaAdelgazada , 512, 512, 3, 3);

/** Se obtienen las minucias de la huella ejecutando el algoritmo sobre cada bloque
paralelamente en la GPU. El método obtenerMinucias() implementaría la primitiva
ParallelReduction de CUDA para hacerlo aún más eficiente **/
Huella huellaConMinucias = obtenerMinucias();

/** Se eliminan aquellas minucias que se cree falsas (dos que se encuentran separadas por pocos
píxeles, por ejemplo) **/
Huella huellaFinal = eliminarMinuciasFalsas(huellaConMinucias);

/** Obtengo las 1024 huellas de la BD, cada una con sus 8 orientaciones **/
Huella[1024][8] huellasBD = getHuellasBD();

/** Se sube la grilla de 1024x8 bloques a la GPU en bloques de 32x32 threads **/
subirGPU(huellasBD, 1024, 8, 32, 32);

/** Se realizan todas las comparaciones con la huella ingresada en paralelo en la GPU y se
obtiene la que tiene mayor cantidad de similitudes entre todas las huellas de la BD **/
int[1024][8] similitudes = compararHuellas(huellaFinal, UMBRAL_DISTANCIA,
UMBRAL_ANGULO);
Huella huellaCandidata = getMaxSimilitudes(similitudes);

/** Si supera el umbral se obtienen las preferencias del usuario y se permite el ingreso **/
Preferencia preferencias;
if (huellaFinal.compare(huellaCandidata) > UMBRAL_MINUCIAS) {
    permitirIngreso();
    preferencias = huellaCandidata.getPreferencias();
}
```

4 Pruebas que pueden realizarse

Para probar que todo funcione correctamente simplemente hay que registrarse en la aplicación ingresando la huella dactilar por primera vez, preparar varios tragos de manera que las preferencias por defecto se vean modificadas, y por último intentar utilizarla en otro smartphone y verificar que el sistema nos permite el ingreso correctamente, obteniendo las mismas preferencias que se tenían en el anterior dispositivo.

5 Conclusiones

Este estudio nos permitió demostrar que la idea de agregar el reconocimiento de huellas dactilares a SmartBarman es totalmente factible y una importante mejora, ya que actualmente si un usuario cambia de dispositivo, perderá todos sus datos y preferencias.
Actualmente los sensores de los dispositivos móviles son realmente muy eficientes permitiendo el reconocimiento de minucias en cuestión de milisegundos, por lo que la paralelización en la GPU en estas etapas no es una importante mejora. Sin embargo, si podemos notar una reducción en el tiempo de procesamiento en la etapa de reconocimiento ya que se deben realizar miles y miles de comparaciones entre todas las huellas presentes en la base de datos [7][8]. Es aquí donde podemos sacar partido de las ventajas de la tecnología GPU, paralelizando las comparaciones y evitando la secuencialidad de utilizar la CPU.
Mediante esta investigación pudimos dilucidar las complejidades que conlleva la computación de altas prestaciones, pero que son indispensables cuando el volumen de datos a procesar es muy grande. Además, nos permitió conocer en profundidad el funcionamiento de los algoritmos de reconocimiento de huellas dactilares y lo complejos que son, por lo que resulta de vital importancia la necesidad de paralelizar su procesamiento y maximizar el uso de los recursos disponibles (en este caso, la tecnología GPU).
En un futuro se podría agregar al sistema un módulo que permita el pago de los tragos, de forma que el sistema pueda comercializarse a boliches donde cualquier persona pueda elegir el trago a preparar y pagar haciendo uso de su celular (ya sea mediante un código QR, NFC, tarjeta de crédito, etc.)

6 Referencias

1. Chambi Mamani, E.W.: Reconocimiento y detección biométrico basado en imágenes de huellas digitales. (2016)
2. Sarzuri Flores, V.: Algoritmo de clasificación de huellas dactilares basado en redes neuronales función base radial. (2014)
3. Ocaña Diez de la Torre, M.: Algoritmos de matching entre huellas dactilares. (2017)
4. Aguilar, G., Sánchez, G., Toscano, K., Nakano, M., Pérez, H.: Reconocimiento de huellas dactilares usando características locales. (2008)
5. Bai, S., Marques, J.P., McMahon, M. T., Barry, S. H.: GPU-Accelerated Fingerprint Matching. (2011)
6. Fuliang, W., Kaiyuan, Y.: The Implementation of a Fingerprint Enhancement System Based on GPU via CUDA. (2017)
7. Sartasov, S.: On Fingerprint Processing Involving Modern Massively Parallel Architectures. (2015)
8. Awad, A.: Fingerprint Local Invariant Feature Extraction on GPU with CUDA. (2013)