Trabajo de Investigación
Computación de Altas Prestaciones en Proyecto de Integracion Android-IoT

Dezerio Sandro
Jalid Fernando
Nestrojil Lucas
Trotta Mauro
Ibaceta Leandro

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

Resumen. El siguiente documento es un trabajo de investigación que aborda la temática computación de altas prestaciones y explica de qué manera podemos implementar los beneficios del paralelismo para adquirir una nueva funcionalidad dentro del proyecto *SelfieHouse*. Dicho proyecto tiene como una de las principales funciones el acceso controlado a la casa administrada desde una app desarrollada para Android. Por este motivo, el principal enfoque de este documento es dilucidar de qué manera podemos integrar el reconocimiento facial a nuestra aplicación.

Palabras claves: HPC, Paralelismo, Android, Dispositivo Móvil, Sistema Embebido, OpenMP, OpenCL, MOMP.

1 Introducción

El trabajo de investigación consiste en el relevamiento y estudio de los paradigmas sobre el uso de paralelismo para aplicaciones en sistemas que requieran altos niveles de cómputo y manejo de datos. Se estudiará cuál es el método que más se adapta a nuestras necesidades, y revisaremos el algoritmo que debemos aplicar para el caso propuesto.

En el módulo se va a interiorizar sobre el uso de paralelismo para reconocimiento facial en tiempo real para implementación de esta funcionalidad dentro del proyecto anteriormente mencionado. Brindaremos una explicación sobre qué herramientas se usan y de qué manera se desarrollan e implementan, adentrando en los algoritmos y software adicional a introducir.

En la actualidad existen 3 métodos de reconocimiento facial: **de rasgos locales** (reconocen los ojos, la nariz, la boca, miden las distancias y los ángulos de la cara), **de rasgos globales** (Aportan información de toda la cara) y mixtos: Combinación de los dos anteriores.

Puede que el método de reconocimiento facial resulte un método no intrusivo para los usuarios ya que los datos pueden ser adquiridos sin que el sujeto se percate de ello. Por otra parte, a partir de la identificación automática de una persona podría ser muy beneficioso para cualquier aplicación. Sin embargo, a la hora de identificar a una persona a partir de su aspecto facial, existen una serie de dificultades, principalmente por la variabilidad. Es muy complejo el reconocimiento facial cuando la variabilidad entre individuos es muy pequeña, o cuando la variabilidad entre las distintas imágenes que han sido adquiridas en diferentes condiciones de posición o iluminación. Otro de los grandes obstáculos es el crecimiento superlativo de la base de datos, lo que conlleva a un gasto computacional alto. En la actualidad, existen diversas aplicaciones del reconocimiento facial. Podemos dividirlo en tres grandes grupos: **Comercial**, aplicaciones en redes de ordenadores, seguridad electrónica, acceso a internet, cajeros automáticos, controles de acceso, teléfonos móviles. Entre las aplicaciones que comparten el primer grupo son importantes los aportes de empresas como Google o Facebook. Por ejemplo, la última innovación de Google ha sido un nuevo sistema facial que permite al usuario el acceso a la información de su Smarthone basándose en un análisis de la forma y el tamaño de la nariz, mandíbula y pómulos. Google ha patentado este desbloqueo de dispositivos con un simple reconocimiento facial. Por otro lado, Facebook ha creado una herramienta que permite reconocer al usuario mediante una foto. Otro grupo es el **Gubernamental**, sus aplicaciones son las vinculadas con documentos identificativos (DNI, Pasaporte), seguridad social, control de fronteras, aeropuertos, etc. La aplicación más conocida y moderna, es la utilizada para el chequeo de pasaportes, en donde se pone un pasaporte y se comprueba que la foto coincide con la del mismo, registrada en la base de datos. Aquí el método de reconocimiento facial utilizado es el de rasgos globales, debido a que este posee mejor nivel de detalle. Por último, en el ámbito **forense**, donde tienen lugar en investigaciones criminales, identificación de cadáveres, terrorismo, identificación de personas desaparecidas, etc. [1]

[1] http://audias.ii.uam.es/seminars/PFC_Luis_Blazquez.pdf

A finales del año 2017 se generó un cambio en la tendencia de la seguridad a la hora de desbloquear el móvil. Por ejemplo, Samsung introdujo su sistema de lectura iris. Apple aposto a deshacerse del lector de huellas y apostarlo todo a Face ID.[1] Continuando con la tendencia, apareció el OnePlus 5T con una solución más rápida.

Recientemente, la velocidad computacional y la capacidad de la batería de los dispositivos móviles se han incrementado enormemente. Por ese motivo, para lograr que el dispositivo móvil en cuestión logre procesar rápidamente el gran volumen de información que representa realizar el reconocimiento facial investigamos acerca de las herramientas disponibles en el entorno de programación Android en donde podamos escribir programas multihilos y aprovechar todo el poder del paralelismo. Para empezar, uno de los principales inconvenientes a la hora de usar paralelismo es la dificultad de programación al hacer uso de Pthread para desarrollar aplicaciones, ya que se debe lidiar manualmente con problemas de partición, sincronización de datos e hilos, tanto como el equilibrio de carga. Por otro lado, nos encontramos con que muchos dispositivos móviles actuales son compatibles con librerías de tiempo de ejecución CUDA/OpenCL pero no existen IDEs para desarrollar aplicaciones en los dispositivos, para eso deben de compilarse en las computadoras y luego descargarse a los dispositivos móviles para su ejecución. Esto resuelve sus problemas sin una conexión de red, porque si bien las redes inalámbricas están mejorando cada vez más, todavía no se acercan a las redes cableadas en términos de latencia y ancho de banda, particularmente para escenarios de alto tráfico.

Se propone, por ende, un entorno de programación móvil para OpenMP, llamado MOMP basado en el paper *An OpenMP Programming Environment on Mobile Devices* [1]. En esta aplicación, los usuarios pueden escribir, compilar y ejecutar programas OpenMP en sus dispositivos móviles basados en Android para explotar el CPU y GPU que llevan embebidos. Cuando los usuarios desarrollan aplicaciones paralelas con OpenMP, se necesitan agregar directivas OpenMP en el codigo C/C++. El compilador OpenMP de MOMP puede generar PThreads y código OpenCL basados en la semántica de las directivas OpenMP y puede agregar automáticamente las instrucciones necesarias para la partición de datos y sincronización en los programas generados.

Además, debido al alto grado de compatibilidad, MOMP hace que los usuarios porten fácilmente sus programas OpenMP desde las computadoras a sus dispositivos móviles sin necesidad de realizar ninguna modificación. También proporciona una interfaz sencilla para elegir CPU o GPU para ejecutar diferentes regiones paralelas en el mismo programa en función de las propiedades de las regiones paralelas. Por todo esto, MOMP puede reducir efectivamente la complejidad computacional de la computación heterogénea en dispositivos móviles y explotar el poder computacional de los mismos para mejorar el rendimiento de las aplicaciones de usuario.

---

[1] https://www.xatakandroid.com/moviles-android/reconocimiento-facial-en-android-de-tecnologia-olvidada-a-una-alternativa-al-lector-de-huellas

2 Desarrollo

Para implementar MOMP hay dos trabajos que deben hacerse. Lo primero es portar el compilador OpenMP de OMPICUDA[1] dentro del dispositivo móvil. Lo segundo es implementar una biblioteca de tiempo de ejecución OpenCL basada en CUDA driver API[2].

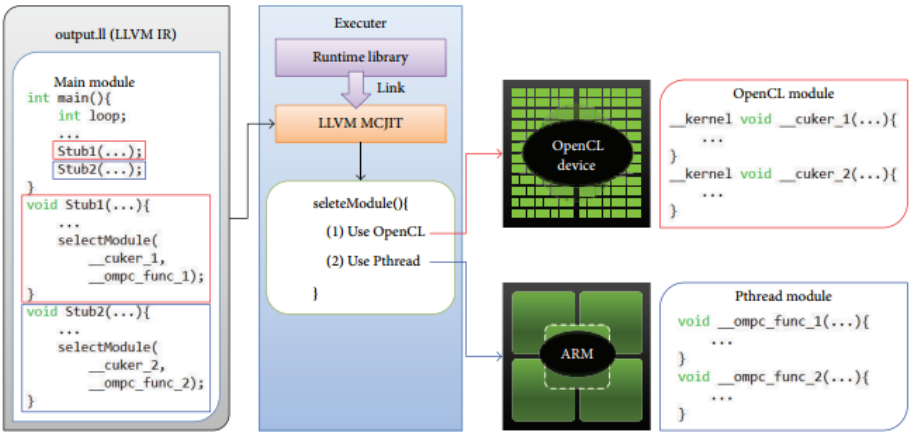

Figura 1 – Flujo de Ejecución en MOMP

Para implementar un sistema de reconocimiento de rostros se presentan 6 etapas bien definidas: Captura de imagen, preprocesamiento, localización, escalamiento y ajuste, extracción de características y por último la clasificación de la toma de decisión.

Se propone utilizar OpenCL para el procesamiento de imágenes y PThreads para la comparación de vectores y validación contra la base de datos aprovechando este hibrido anteriormente mencionado.

---

[1] Framework que permite seleccionar seleccionar el uso de CPU o GPU para la ejecución de diferentes regiones paralelas en el mismo programa. Admite la reasignación de recursos en función de los estados del CPUs y GPUs. https://ieeexplore.ieee.org/document/6270608/

[2] Hace posible compilar y vincular kernels CUDA en el ejecutable en tiempo de ejecución. https://docs.nvidia.com/cuda/cuda-driver-api/index.html

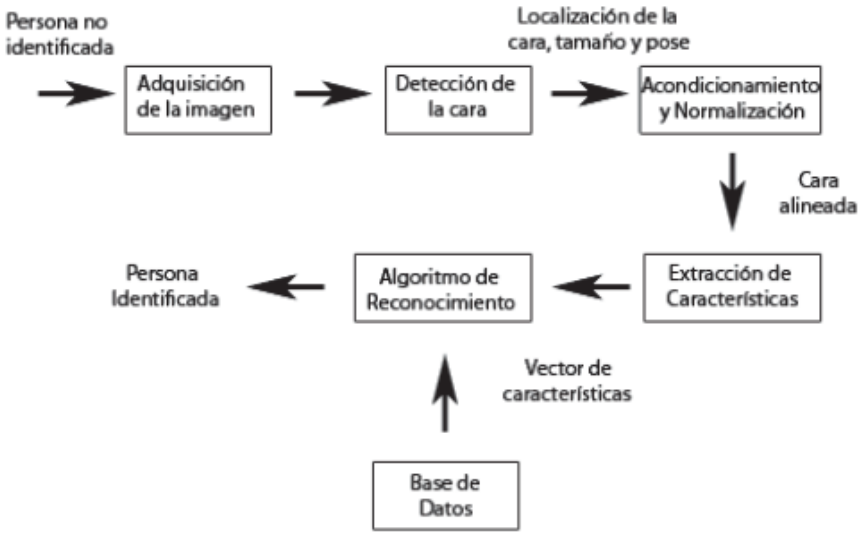

Figura 2 – Esquema general para implementar un sistema de reconocimiento facial

Para el desarrollo del algoritmo, nos basamos en un trabajo de investigación explicitado en el paper Surpassing Human-Level Face Verification Performance on LFW with GaussianFace. [2] Donde se explica la dificultad que conlleva determinar en base a la comparación de dos imágenes si se trata de la misma persona, debido a que, al recibir una nueva imagen, ésta es comparada con un dataset que proporciona un gran conjunto de imágenes faciales con una compleja variación de pose, iluminación, expresión, raza, etnia, edad, género, ropa, peinados y otros parámetros. Para lograr la verificación correcta de caras, el algoritmo trabaja normalizando cada cara en una imagen de 150 x 120 píxeles, transformándolo en base a cinco hitos de la imagen: la posición de ambos ojos, la nariz y las dos esquinas de la boca. Entonces, la imagen se divide en parches superpuestos de 25 x 25 píxeles y describe cada parche utilizando un vector, el cual captura sus características básicas, estas tareas son desarrolladas en las etapas preprocesamiento, localización, escalamiento y ajuste, y extracción de características anteriormente indicadas. Teniendo esto listo, el algoritmo está listo para buscar similitudes en las imágenes. Pero antes debe saber dónde buscar. Es acá donde el dataset de entrenamiento entra en juego. El enfoque habitual es utilizar un único conjunto de datos para entrenar el algoritmo y utilizar una muestra de imágenes del mismo conjunto de datos para probar el algoritmo.
Pero cuando el algoritmo se enfrenta a imágenes que son completamente diferentes del conjunto de entrenamiento, a menudo falla. Cuando la distribución de la imagen cambia, estos métodos pueden sufrir una gran caída de rendimiento. Sin embargo, han entrenado a GaussianFace en cuatro conjuntos de datos completamente diferentes con imágenes muy diferentes. Por ejemplo, uno de estos conjuntos de datos se conoce como la base de datos Multi-PIE y consiste en imágenes de caras de 337 sujetos de 15

puntos de vista diferentes bajo 19 condiciones diferentes de iluminación, tomadas en cuatro sesiones de fotos.
Habiendo entrenado el algoritmo en estos conjuntos de datos, finalmente lo soltaron en la base de datos Faces in the Wild. El objetivo fue identificar pares coincidentes y detectar pares que no coinciden.

Los humanos pueden hacer esto con una precisión del 97.53 por ciento. En cambio, el modelo GaussianFace puede mejorar la precisión al 98.52 por ciento, lo que por primera vez supera al rendimiento a nivel humano.
Además, la memoria grande también es necesaria. Por lo tanto, para una aplicación específica, se necesita equilibrar las tres dimensiones: memoria, tiempo de ejecución y rendimiento. En términos generales, un mayor rendimiento requiere más memoria y más tiempo de ejecución. Sin embargo, este tema puede abordarse utilizando el paralelismo distribuido, al realizarse siempre la misma tarea cuando se valida que dos imágenes corresponden a la misma persona, utilizamos el framework Mobile OpenMP para ejecutar la validación de varios pares de fotos en paralelo.

Si bien anteriormente indicamos que todo el procesamiento lo hacemos dentro del mismo dispositivo móvil utilizando MOMP para paralelizar el procesamiento, el guardar la base de datos para realizar la comparación de imágenes también dentro del dispositivo móvil no es una opción viable. Por ese motivo, la base de datos va a estar dentro del webservice de nuestra aplicación. Para traer los datos de las imágenes del webservice al dispositivo móvil utilizaremos un thread que se encargue de agregar cada vector que contiene las características de cada cara a una lista, de la cuál consumirán todas las tareas que se estén ejecutando en paralelo. Esta lista debe ser especialmente adaptada para tolerar concurrencia, porque por un lado se le agregan elementos del webservice, y por otro lado las tareas en paralelo le quitan elementos para realizar la comparación con el vector correspondiente a la nueva imagen de entrada. Debido al gran volumen de información que puede generarse en nuestro servidor, decidimos utilizar una técnica de compresión para que se reduzca sustancialmente el tamaño de los datos y los costos de comunicación. Dicha técnica esta descripta en el paper A Research Paper on Lossless Data Compression Techniques[3] en el mismo se indica que hay dos tipos de compresión de datos, Lossy Compression y Lossless Compression. El primero se utiliza en aplicaciones donde alguna pérdida de información es aceptable, utilizamos esta variante para guardar las fotos recibidas desde los terminales Android de quienes solicitan acceso a la casa y esa misma compresión genera un menor costo de comunicación cuando se envía del servidor al terminal Android destino. Por otra parte el paper muestra 4 variantes de Lossless Compression, en todos los casos la información no tiene pérdidas, nosotros elegimos el método Shannon Fano Coding para comprimir los los vectores que contienen las características de cada rostro. Éste metodo se basa en la generación de un árbol binario para representar las probabilidades de la ocurrencia de cada simbolo. Los símbolos son ordenados de manera que los de aparición más frecuente aparezcan arriba, y los menos frecuente abajo.

De esta manera, podemos integrar el algoritmo de GaussianFace a nuestra aplicación para realizar el reconocimiento de rostro de las personas que quieran acceder a cualquier domicilio que tenga integrado SelfieHouse.

Explicación del algoritmo

A continuación se detalla un pseudocódigo de como se podría implementar, de manera global, los métodos anteriormente mencionados. Se introduce en el código Android nuevos métodos de reconocimiento facial con directivas OpenMP y OpenCL.

Pseudocodigo

```
Void main{
        Global OK int;
        Global dataset List<data>
        img Imagen;
        Resultado int;

        cargarDataset();
        ProcesarImagen(*img);
        Resultado = BuscarCoincidencia();

        If (Resultado==1)HabilitarAcceso(TRUE); Else HabilitarAcceso(FALSE);
}

int ProcesarImagen(*img){ //PSEUDOCODIGO OPENCL EJEMPLO
cl_context clCreateConext(); //espacio o contexto que se crea para manejar objetos y recursos OpenCL
cl_mem clCreateBuffer(cl_context, * img) //creamos un objeto de memoria con la imagen que procesamos
cl_int clEnqueueWriteBuffer (cl_command_queue, cl_mem, writeBuffer, * instrucciones, * img)
                                //Inicializamos el objeto en memoria, y enviamos el puntero de la imagen que nos devuelve el procesamiento
}

int BuscarCoincidencia() { //PSEUDOCODIGO OpenMP EJEMPLO
        tamanoVector = Vector.CountItem;
        i = 0;
        While ( !ok or I < tamanioVector) {
                #pragma omp single (ok) {
                        Ok = CompararImagenes(Vector[i],imagen);
                } !$omp end single (noWait)
                I++;}
        If (ok) Return 1; Else Return 0;
}
```

En el código utilizamos la potencia que nos puede proporcionar GPU para el procesamiento de imágenes empleando código OpenCL. Mientras que para la comparación, dentro del bucle de búsqueda, paralelizamos las comparaciones con directivas OpenMP. De esta manera, podemos recudir considerablemente el tiempo de procesamiento que esto conlleva para poder dar una respuesta aceptable al usuario.

Explicación de las funciones

CargarDataset(): Esta función despliega un thread asincrónico encargado de establecer una comunicación con el servidor, y de esta forma traer los datos correspondiente a cada imagen alojada en la base de datos para realizar las comparaciones.
ProcesarImagen(): Esta función está basada en código CUDA, ya que requiere procesamiento de la imagen dividiéndola en módulos y calculando distintas características de la cara.
CompararImagenes(): La comparación se basa en la comparación de rasgos y características que fueron detectadas en las distintas imágenes, para determinar si son la misma persona o no. Esta función posee una complejidad computacional baja, por lo que creemos importante poder realizar una paralelización debido al incremento potencial de la base de datos de imágenes. Asi mismo, declaramos la variable global *ok* que es compartida por los distintos hilos, y manifestamos a través de el parámetro *noWait*, que el hilo lanzado sea no bloqueante.

3 Pruebas que pueden realizarse

Para realizar esta prueba, nuestra aplicación debe guardar la imagen en la base de datos cada vez que una persona quiere ingresar en la casa. Dejando al administrador darle permisos de acceso totales, parciales o denegado. Esta base de datos va a ir incrementando por lo que es muy importante para la performance de la app la paralelización en la búsqueda de coincidencias.

Esta funcionalidad va a dar mucha flexibilidad en el sistema, ya que nos va a permitir optimizar las peticiones de una manera más automática. Sin embargo, se debe tener en cuenta que estos algoritmos no son 100% fiables, ya que poseen una tasa de error, que puede ser reducida, pero que es una problemática en este tipo de aplicaciones.

Los casos de prueba que debemos realizar, son con distintos tipos de iluminación y de posición de la foto, ya que esto es de vital importancia para las verificaciones biométricas para poder reducir la tasa de fallos lo máximo posible.

4 Conclusiones

El presente trabajo da una breve introducción de cómo podemos aprovechar distintas tecnologías de procesamiento paralelo que podemos integrar a nuestros proyectos. Estas tecnologías son muy optimas ya que pueden funcionar en un entorno de trabajo con limitaciones, como puede ser un teléfono celular o una tableta. Por eso, tratamos de describir una alternativa para la programación paralela en dispositivos Android.

En este documento se presenta una herramienta que puede estar al alcance de cualquier desarrollador. Aunque es reciente el uso de este paradigma en celulares, vemos que en un futuro cercano aparecerán más medios disponibles que estén al alcance de los programadores, aprovechando también el crecimiento exponencial de las prestaciones que provee un celular.

Notamos un gran beneficio la explotación de estas herramientas y creemos que el tiempo y las nuevas investigaciones van a proveer a los programadores herramientas que faciliten, e incluso motiven, al uso de esta herramienta.

En futuras investigaciones sería interesante que las empresas provean dentro de sus IDEs herramientas y hardware para introducir la paralelización sistemas embebidos ya que creemos que mejoraría mucho la performance del producto y daría paso a desarrollos de software mucho más poderosos.

5 Referencias

[1] Tyng-Yeu Liang, Hung-Fu Li, and Yu-Chih Chen.: An OpenMP Programming Environment on Mobile Devices. National Kaohsiung University of Applied Sciences, No. 415, Taiwan (2016).

[2] Chaochao Lu, Xiaoou Tang.: Surpassing Human-Level Face Verification Performance on LFW with GaussianFace. Department of Information Engineering, The Chinese University of Hong Kong (2014).

[3] Prof. Dipti Mathpal, Prof. Mittal Darji, Prof. Sarangi Mehta.: A Research Paper on Lossless Data Compression Techniques, vol. 4, Issue 1, June 2017