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. En el siguiente documento vamos a abordar el tema computación de altas prestaciones y cómo podemos utilizar e implementar los métodos propuestos por los distintos paradigmas de HCP para procesamiento de algoritmos de alta complejidad o por manejo de grandes volúmenes de datos. A lo largo del documento nos enfocaremos principalmente en explicar cómo podríamos llevar a cabo extensiones de nuevas funcionalidades que requieran y justifiquen el uso de paralelismo dentro del proyecto "SelfieHouse" y de qué manera se podrían implementar. El proyecto "SelfieHouse" tiene como principal objetivo administrar una casa inteligente de manera remota haciendo uso de una app desarrollada para Android que se vincula a un microcontrolador Arduino que controla y gestiona el funcionamiento componentes de la casa como la puerta de entrada, ventilador, alarma. Durante el trabajo se llevará a la práctica una explicación del algoritmo a utilizar, las pruebas a realizar y por ultimo efectuar una conclusión sobre el uso de estos métodos y las ventajas que nos brindan.

Palabras claves: HPC, Android, Microcontrolador, 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
- 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. Otra 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.

- 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.

- Forense: Tienen lugar en investigaciones criminales, identificación de cadáveres, terrorismo, identificación de personas desaparecidas, etc.

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. Continuando con la tendencia, apareció el OnePlus 5T con una solución más rápida.

2 Desarrollo

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, proponemos realizarlo mediante un entorno de programación móvil OpenMP, llamado MOMP en el paper "An OpenMP Programming Environment on Mobile Devices" [1]. Usando 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 para resolver 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. 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.

Para implementar MOMP hay dos trabajos que deben hacerse. El primero es importar el compilador OpenMP de OMPICUDA [2] dentro del dispositivo móvil, es una herramienta de desarrollo para clústers CPU/GPU híbridos, donde los usuarios pueden hacer uso de modelos de programación familiares, compuestos por OpenMP y MPI.

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"[3]. 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. 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. Otra es una base de datos llamada Life Photos que contiene alrededor de 10 imágenes de 400 personas diferentes.
Habiendo entrenado el algoritmo en estos conjuntos de datos, finalmente lo soltaron en la base de datos Faces in the Wild. El objetivo es 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.
Aunque se introducen varias técnicas, como la aproximación de Laplace y el gráfico de anclaje, para acelerar el proceso de inferencia y predicción en el modelo GaussianFace, todavía lleva mucho tiempo entrenarlo para el alto rendimiento. 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. Para abordar el problema de la memoria, se puede buscar una representación escasa más eficiente para la gran matriz de covarianza que es obtenida por el Discriminative GaussianProcess Latent Variable Model (DGPLVM), uno de los principales modelos de GaussianFace para automatizar el aprendizaje de las características discriminatorias de cada rostro.

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.

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.

3 Explicación del algoritmo.

Para mostrar un poco como funciona esta tecnología, se citará una breve reseña sobre como deberíamos implementar las funciones de reconocimiento facial con directivas OpenMP.

Pseudocodigo

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

        cargarDataset();
        Imagen = ProcesarImagen();
        Resultado = BuscarCoincidencia();

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

int BuscarCoincidencia() {
        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 == true) Return 1; Else Return 0;
}
```

Lo que hicimos fue paralelizar la función que compara las imágenes, de manera que la figura pueda ser comparada con los datos existentes en la base de datos. De esta manera, podemos recudir considerablemente el tiempo de procesamiento que esto conlleva para poder dar una respuesta aceptable al usuario.

La directiva utilizada fue **single** en donde el bloque ejecuta por un único thread pasando como cláusula inicial la variable global *ok* que va a ser utilizada por los demás procesos. También incluimos la cláusula de cierra *noWait* que indica que el hilo no debe esperar por nadie, es decir, que es no bloqueante.

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.

4 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.

5 Conclusiones

<<Las conclusiones del trabajo deben tener:>>

- <<Breve repaso del trabajo realizado>>.
  Este trabajo da una breve introducción de cómo podemos aprovechar distintas tecnologías que podemos integrar a nuestros proyectos, que son muy optimas y pueden funcionar en un entorno de trabajo con limitaciones, como puede ser un teléfono celular.
- <<Sugerencias para un próximo trabajo, ya sea funcionalidad o del algoritmo>>.

6 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. H.-F. Li, T.-Y. Liang, and J.-Y. Chiu.: A compound OpenMP/MPI program development toolkit for hybrid CPU/GPU clusters. The Journal of Supercomputing, vol. 66, no. 1, pp. 381–405, (2013).

3. 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).