Universidad
de La Laguna

Escuela Superior de
Ingeniería y Tecnología
Sección de Ingeniería Informática

Trabajo de Fin de Grado

Programación en paralelo en dispositivos móviles.

Parallel programming on mobile devices

Gabriel García Ventura

La Laguna, 30 de agosto de 2016

D. **Francisco C. Almeida Rodríguez**, con N.I.F. 42831571-M profesor Titular de Universidad adscrito al Departamento de Nombre del Departamento de la Universidad de La Laguna, como tutor

C E R T I F I C A (N)

Que la presente memoria titulada:

“Programación en paralelo en dispositivos móviles.”

ha sido realizada bajo su dirección por D. **Gabriel García Ventura**, con N.I.F. 42192484-L.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firma la presente en La Laguna a 30 de agosto de 2016.

Agradecimientos

A mis padres, tantas cosas tengo que agradecerles que en este caso las resumiré en darles las gracias por la educación que me han dado. Que si bien esta no ha sido de carácter académico sí que ha sido la que me ha hecho llegar hasta aquí junto con su apoyo incondicional y algún que otro tironcito de oreja: ¡gracias mamá! ¡Gracias papá!

A mis hermanos, ¡por aguantarme desde hace tanto y tantas cosas! Y por saber lo que necesitas aún cuando yo no me había dado cuenta.

A mi amigo y compañero de estudios sin cuya ayuda ¡posiblemente aún estaría por mitad de carrera!

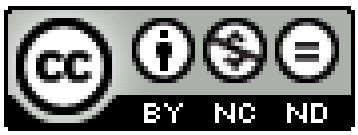
A ella, por enseñarme su forma de ver el mundo y por hacerme comprender que los pequeños problemitas pueden ser de las mejores cosas que nos pasen.

A todos los profesores con los que he ido a clase que, si bien es cierto que no con todos se congenia igual de bien, mirándolo ahora con perspectiva, puedo decir que todos se han preocupado a su manera para brindarme la posibilidad de mejorar, aprender y dar lo mejor de mí.

A mi tutor, sin cuya ayuda este proyecto no habría salido adelante, quién siempre ha tenido las respuestas a las preguntas que me bloqueaban y quién me ha tendido la mano una y otra vez para superar los baches del camino. ¡Gracias Francisco!

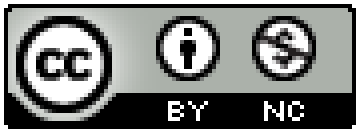
Licencia

* Si NO quiere permitir que se compartan las adaptaciones de tu obra y NO quieres permitir usos comerciales de tu obra indica:

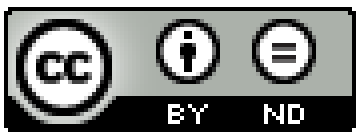


* Si quiere permitir que se compartan las adaptaciones de tu obra mientras se comparta de la misma manera y NO quieres permitir usos comerciales de tu obra indica:

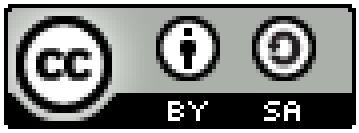
* Si quiere permitir que se compartan las adaptaciones de tu obra y NO quieres permitir usos comerciales de tu obra indica:



* Si NO quiere permitir que se compartan las adaptaciones de tu obra y quieres permitir usos comerciales de tu obra indica:



* Si quiere permitir que se compartan las adaptaciones de tu obra mientras se comparta de la misma manera y quieres permitir usos comerciales de tu obra (licencia de Cultura Libre) indica:



* Si quiere permitir que se compartan las adaptaciones de tu obra y quieres permitir usos comerciales de tu obra (licencia de Cultura Libre) indica:

Resumen

El objetivo de este trabajo ha sido el de desarrollar, y posteriormente analizar el rendimiento de aplicaciones en dispositivos móviles Android.

Este estudio se ha realizado sobre el algoritmo de procesamiento de imágenes conocido como “Zhang-Suen” o “Adelgazamiento de Patrones”, implementando este algoritmo tanto en una versión secuencial en lenguaje Java así como en otra versión paralela haciendo uso de la tecnología RenderScript.

Todo el desarrollo del estudio se ha llevado a cabo en la plataforma Android utilizando el entorno de desarrollo oficial de Google Android Studio, lo que supone la familiarización y aprendizaje de estas tecnologías y herramientas de desarrollo.

Como medida para las comparaciones y estudio del rendimiento se han usado los tiempos de ejecución de dicho algoritmo en las dos versiones programadas descritas anteriormente.

El estudio lo podemos dividir en tres grandes etapas:

1- Codificación en secuencial
2- Codificación en paralelo
3- Análisis del rendimiento

El desarrollo del estudio comienza con la codificación del algoritmo “Zhang-Suen” en su versión secuencial y una vez ésta estuvo desarrollada se procedió a la paralelización del algoritmo usando la tecnología que nos proporciona RenderScript.

El proyecto finaliza con un análisis comparativo de los tiempos de ejecución de los resultados obtenidos mediante las dos versiones de la implementación.

Palabras clave: RenderScript, Android, Android Studio, Zhang-Suen, Adelgazamiento de Patrones, Paralelismo.

Abstract

The goal of this work has been develop and then analyze the performance of application on Android mobile device.

This study was conducted on the algorithm of image processing known as "Zhang-Suen" or "thinning patterns", implementing this algorithm on both a sequential version in Java as well as in other parallel version using the RenderScript technology.

All development study has been realized on the Android platform using the development official environment of Google: Android Studio, which means familiarization and learning of these technologies and development tools.

As a measure for comparisons and performance study the running times of such algorithms programmed in the two versions described above have been used.

The study can be divided into three stages:

1- Coding sequence

2- Parallel coding

3- Performance analisys

The development of the study begins by coding the algorithm "Zhang-Suen" in its sequential version and once it was developed I proceeded to the parallelization of the algorithm using the technology that provides RenderScript.

The project ends with a comparative analysis of the runtimes of the results obtained by the two versions of the implementation.

Keywords: RenderScript, Android, Android Studio, Zhang-Suen, Thinning Patterns, Parallelism.

Índice General

Índice de figuras

Índice de tablas

Capítulo 1. Introducción

Actualmente los dispositivos móviles (principalmente Smartphones y Tablets) se han instaurado en nuestras vidas y están siendo ampliamente usados cada día por un mayor número de personas. Tanto es así que para 2016 se habrán distribuido alrededor de 1600 millones de Smartphones.

Por un lado, el hardware sigue una progresión acelerada en cuanto a prestaciones se refiere, ya que se hace uso de CPUs y GPUs de varios núcleos, incrementos en capacidad de memoria y RAM, etc. Por otro lado esta progresión se ha visto frenada por el software, ya que éste aún, podemos decir que, sigue un modelo de desarrollo secuencial (similar al de procesadores de un solo núcleo).

Es por ello que se plantea hacer uso de las ventajas que proporciona la arquitectura del hardware con el fin de lograr un mejor rendimiento y eficiencia de las aplicaciones mejorando tanto la velocidad de cómputo de éstas como consiguiendo un ahorro de energía.

El sistema operativo Android para dispositivos móviles nos permite enfocar la creación de algoritmos desde varios modelos de programación, que presentan unas características determinadas que pueden hacer que un mismo algoritmo diseñado en uno u otro modelo presente distinto rendimiento sobre un mismo dispositivo.

En este Trabajo de Fin de Grado se aborda la codificación del algoritmo de adelgazamiento de patrones Zhang-Suen, desarrollándolo siguiendo dos modelos distintos de programación que permite Android. Por un lado el modelo secuencial (programado haciendo uso del lenguaje Java) y por el otro lado bajo el modelo del paralelismo haciendo uso de la tecnología RenderScript.

De este modo podremos evaluar el rendimiento de los distintos modelos de programación disponibles para los dispositivos permitiéndonos hacer uso del modelo más eficaz y eficiente para nuestros objetivos.

Capítulo 2.
El sistema operativo Android.

2.1 Android

Android es un sistema operativo propiedad de Google, inicialmente desarrollado por Android Inc y posteriormente adquirido por su actual dueño en 2005. Su kernel está basado en Linux y está diseñado principalmente para dispositivos móviles con pantallas táctiles tales como Smartphones y Tablets.

La primera versión de Android fue anunciada el 5 de noviembre de 2007 por la Open Handset Alliance (un conglomerado de fabricantes y desarrolladores de hardware, software y operadores de servicio) bajo el nombre de Android 1.0 Apple Pie. No obstante, el primer móvil con el sistema operativo Android fue el HTC Dream, sacado al mercado en octubre de 2008.

El 12 de noviembre de 2007 se lanzó una versión de prueba del Android SDK (el Software Development Kit de Android, que cuenta con un conjunto de herramientas de desarrollo, depuración, simulador de dispositivos, documentación, biblioteca, ejemplos de código, tutoriales, herramientas de medición, etc.).

El 19 de agosto de 2008 es cuando aparece la versión Beta de este entorno de desarrollo y un mes después, el 23 de septiembre se lanza la versión 1.0.

Android hace uso de la máquina virtual Dalvik para ejecutar sus aplicaciones. La máquina virtual Dalvik (DVM) fue desarrollada por Dan Bornstein con contribución de otros ingenieros de Google y permite la ejecución de prácticamente cualquier aplicación programada en Java. DVM sacrifica la portabilidad que caracteriza a Java (la de poder ser ejecutada en cualquier sistema operativo) por conseguir crear aplicaciones sobre Android con un mejor rendimiento y un menor consumo de energía, característica sumamente importante en dispositivos móviles debido a las restricciones de las baterías.

DVM está optimizada para necesitar poca memoria y permitir ejecutar varias instancias de la máquina virtual de forma simultánea, delegando en el sistema operativo el aislamiento de procesos, la gestión de la memoria e hilos de ejecución.

No obstante, a partir de la versión Lollipop de Android, Dalvik fue sustituido por ARN, que a diferencia del anterior en el que se hace uso del just-in-time para compilar el código cada vez que se inicia una aplicación, ART crea un archivo de compilación de la aplicación tras su instalación, evitando así la continua compilación reduciendo el uso del procesador del dispositivo lo que se traduce en un aumento de la duración de la batería.

2.2 Modelos de desarrollo en Android

Android proporciona la posibilidad de crear aplicaciones siguiendo varios modelos de desarrollo.

2.2.1 SDK

Tenemos por un lado el Android SDK (Software Development Kit), que constituye la forma más generalizada de desarrollar aplicaciones para Android realizándolas haciendo uso del lenguaje Java y XML principalmente.

2.2.2 NDK

Por otro lado tenemos el Android NDK (Native Development Kit) que permite implementar aplicaciones usando lenguajes nativos tales como C o C++, sin embargo, aunque el uso de código nativo pueda suponer una mejora con respecto a la codificación de ciertas aplicaciones, no se aconseja su uso por razones tales como son la preferencia de crear aplicaciones usando estos lenguajes ya que el uso del NDK aumenta en gran medida la complejidad [7].

2.2.3 RenderScript

También se nos presenta la posibilidad de usar RenderScript [8], que es un framework para la ejecución de tareas de cálculo de alto rendimiento en Android. Está orientado para usarlo en aplicaciones paralelas aunque también podemos hacer uso de esta tecnología para computación secuencial con alta carga de trabajo.

Su forma de actuar es distribuyendo en tiempo de ejecución el trabajo entre los procesadores disponibles del dispositivo tanto en CPUs de varios núcleos, GPU o DSPs.

RenderScript es especialmente útil para aplicaciones de tratamiento de imágenes y para desarrollar en él se usa el lenguaje C99.

2.3 Ciclo de creación de una app en Android.

El ciclo de creación de una aplicación [1] incluye varias herramientas y procesos para pasar de nuestro proyecto a un paquete de aplicación Android (APK por sus siglas en inglés de Android Aplication Package). El proceso de construcción de una aplicación típica en Android se puede dividir en los siguientes pasos:

1- El compilador convierte un código en archivos DEX (Dalvik Executable), que incluyen el bytecode y todos los recursos compilados (por ejemplo los ficheros XML que definen la interfaz de usuario) que pueden ser ejecutado en los dispositivos Android.

2- El APK Packager combina los archivos DEX y los demás recursos compilados creando el .apk que ya puede ser instalado y usado en un dispositivo Android, sin embargo, el .apk debe poseer una firma.

3- El APK Packager se encarga de proveer de esta firma dependiendo del tipo de aplicación que se esté creando:

   a- Si se está construyendo una aplicación de prueba que solo se va a usar para testear, el packager firmará la aplicación con una debug keystore.

   b- Si se está realizando una versión final de la aplicación, el packager firmará la aplicación con una release keystore.

4- Una vez que ya tengamos la versión final de nuestro APK, el Packager usa la herramienta Zipalign para optimizar la aplicación con el fin de

conseguir que use menos memoria cuando se esté ejecutando en un dispositivo.

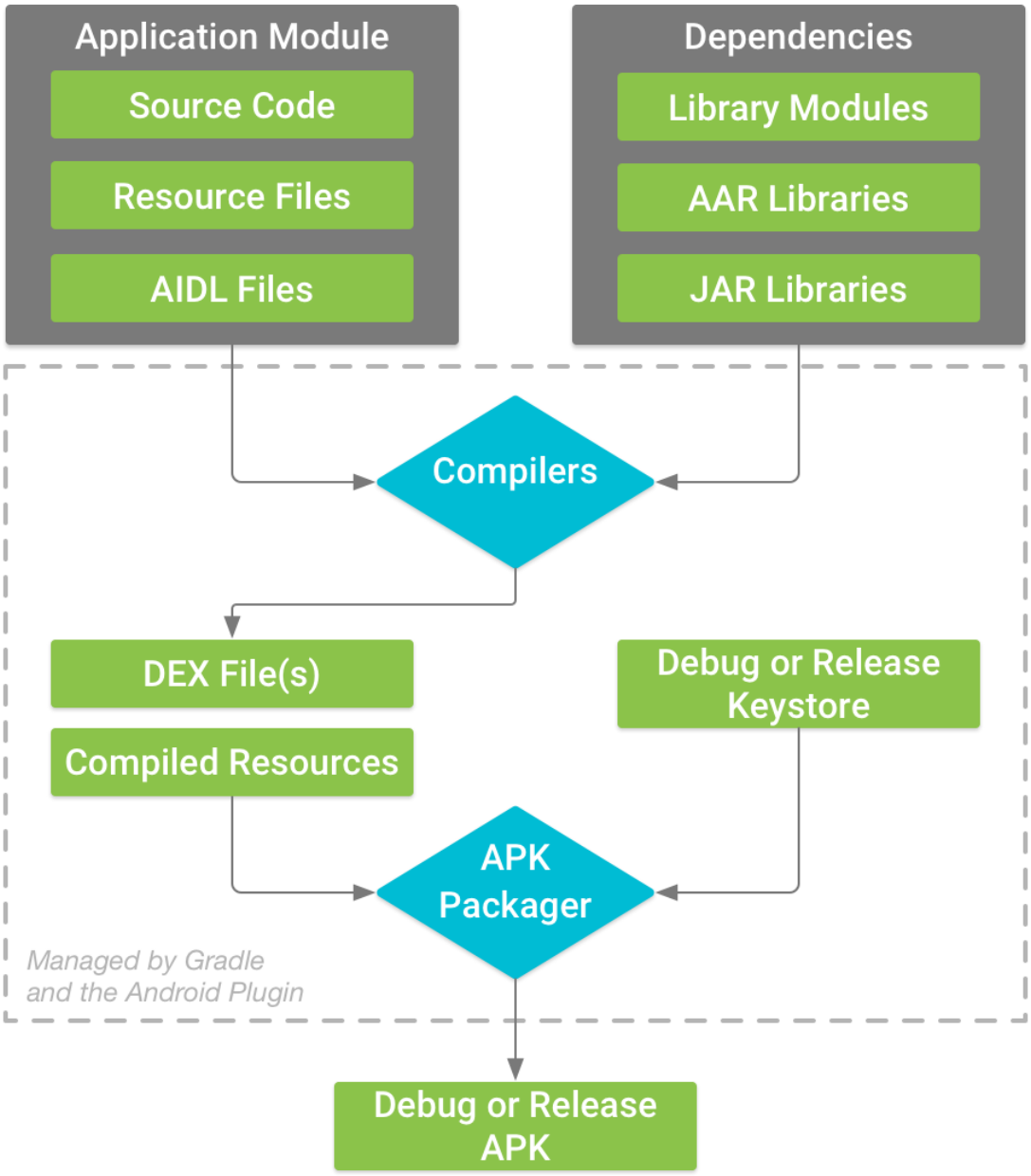


Figura 2.3.1. Diagrama de creación del apk [11].

2.4 Ciclo de vida de una actividad

Las actividades (activity) en Android son clases públicas que representan cada una de las pantallas de nuestra aplicación.

Las aplicaciones en esta plataforma funcionan con un ciclo de vida [2] controlado por el sistema operativo. Desde que se crean hasta que se destruyen, las actividades pueden encontrarse en distintos estados (que son almacenados en un objeto de la clase Bundle), estos estados son:

1- Running: está la primera en la pila de ejecución, el usuario puede interactuar con ella.

2- Paused: ha pasado a segundo plano pero aún es visible, en este caso puede ser cerrada por el sistema si necesita liberar recursos para la actividad activa.

3- Stopped: ha pasado a segundo plano, pero en este caso sí que está completamente tapada por la actividad nueva. El sistema también la puede eliminar en caso de necesitar los recursos.

4- Destroyed: la actividad ya no está disponible, sus recursos han sido liberados y en caso de ser llamada de nuevo esta comenzará un nuevo ciclo de vida.

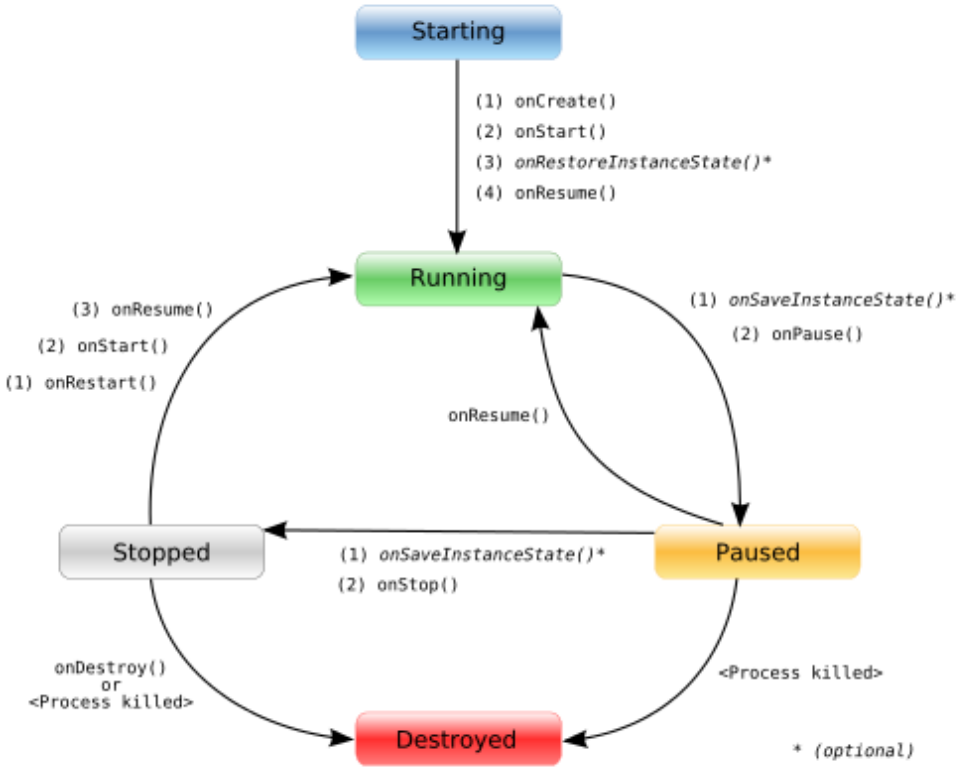


Figura 2.4.2. Diagrama del ciclo de vida de una activity [10].

2.5 Gestión de memoria en Android

La memoria RAM es en cualquier sistema de cómputo, un recurso muy valioso, más aún en dispositivos móviles los cuales tienen unas limitaciones físicas tan grandes. A pesar de que la máquina virtual de Android (Dalvik) lleva a cabo la recolección de basura [3], tenemos que ser consientes de qué espacios de memoria tienen nuestras aplicaciones asignadas y cuando se liberan, puesto que para que el recolector de basura libere dicha memoria es necesario que todas las instancias y recursos que no vayan a ser usados sean liberados explícitamente pasando ya luego a encargarse del resto el recolector de basura.

Android no dispone de espacio de intercambio de memoria (swwap space), por ello con el fin de repartir la memoria RAM entre todas las aplicaciones que la necesiten Android puede hacerlo de las siguientes maneras:

1- Cada proceso de cada aplicación se bifurca de un proceso llamado Zygote, que es lanzado cuando se inicia el sistema y que se encarga de cargar las librerías y recursos. Para lanzar un nuevo proceso el sistema realiza un fork del Zygote, luego carga y ejecuta el código de la aplicación en este nuevo proceso permitiendo que la mayoría de las páginas de la memoria RAM asignadas a ese código y a sus recursos puedan ser compartidas por todos los procesos de la aplicación.

2- La mayoría de los datos estáticos (código Dalvik, recursos de aplicación, código nativo) son mapeados dentro del proceso. Este método permite que los datos sean compartidos entre procesos y además también permite que sean paginados cuando son necesarios.

3- Android comparte la misma memoria RAM entre procesos usando de forma explícita la asignación de estas regiones de memoria.

2.6 RenderScript

RenderScript [5], [6] es una API de computación intensiva a bajo nivel que permite al desarrollador incrementar el rendimiento de sus aplicaciones, proveyéndolos de una API de propósito general para poder ser usada con diferentes tipos de hardware y en un lenguaje “familiar” que consiste en un C99 con extensiones. El cual se compila a un formato intermedio independiente del dispositivo en el que se esté desarrollando y se coloca en el paquete de la aplicación.

En el momento en el que la aplicación es ejecutada, el script es compilado a código máquina y optimizado para el dispositivo en el que se está ejecutando. Esto elimina la necesidad de tener que crear un script para cada arquitectura diferente.

Así pues, podemos decir que RenderScript posee tres objetivos fundamentales:

1. **Portabilidad**: el código puede ser ejecutado en todos los dispositivos, incluso en los que tiene un hardware totalmente diferente al del dispositivo en el que se desarrolló.

2. **Rendimiento**: pretende mejorar el rendimiento de las aplicaciones dentro de las limitaciones del dispositivo.

3. **Usabilidad**: se busca simplificar en lo máximo posible el proceso de desarrollo.

Dentro del marco del proyecto, RenderScript se usó como medio para paralelizar el algoritmo de Zhang-Suen persiguiendo el objetivo de mejorar el rendimiento en cuanto a la determinación del esqueleto de las imágenes.

Llegados a este punto es necesario recalcar que el principal problema que nos podemos encontrar a la hora de trabajar con RenderScript es el de la falta de documentación y ejemplos en los que basarnos, lo que sin duda supone un gran hándicap a la hora de desarrollar aplicaciones con esta tecnología, así como por otro lado, el hecho de que a día de hoy RenderScript es un proyecto que no se sabe si seguirá o por el contrario desaparecerá.

Capítulo 3.
Algoritmo Zhang-Suen

En este apartado se explicará en qué consiste el algoritmo de adelgazamiento de patrones Zhang-Suen [4], quién lo propuso, para que se usa y como es su funcionamiento. Así como las aplicaciones creadas durante el desarrollo del proyecto.

3.1 Zhang-Suen

Nos encontramos ante un algoritmo muy popular dentro del procesamiento de imágenes, que nos permite obtener el esqueleto de las mismas.

El uso del esqueleto de una imagen para obtener las características de un patrón es comúnmente usado para aplicaciones de por ejemplo reconocimiento automático de texto entre otras.

El objetivo final que se persigue es el de obtener una figura de la imagen procesada que sea una serie de trazos de un pixel de grosor que representen la forma de la figura original y que contenga la menor distorsión posible.

El algoritmo que vamos a tratar fue propuesto por T.Y Zhang y C.Y. Suen en 1984 en el artículo llamado "*A Fast Parellel Algorith from Thinning Digital Patterns*" en la revista Communications of the ACM.

Es un algoritmo ampliamente utilizado debido a los buenos resultados que produce y a la sencillez de su implementación. Trabaja sobre imágenes binarizadas y funciona por iteraciones dividiéndose cada una en dos sub-iteraciones en las que se evalúa cada pixel de la imagen de forma independiente y en la que la imagen resultado de cada sub-iteración constituye la entrada a la otra sub-iteración.

En cada una de estas sub-iteraciones se evalúa cada pixel de forma independiente en base a cuatro condiciones, que de cumplirse, permitirían que el pixel fuera borrado puesto que no se le considera parte esencial para el esqueleto de la imagen.

La imagen resultado con los pixeles borrados será la entrada de la siguiente sub-iteración en la que se volverá a evaluar para cada pixel si cumple o no las condiciones de formar parte del esqueleto de la imagen.

Las iteraciones finalizan cuando en las dos sub-iteraciones no se borra ningún pixel.

Dadas las características de este algoritmo, lo hacen perfecto para el estudio que se pretende con este Trabajo de Fin de Grado, puesto que nos permite expresarlo tanto de forma secuencial como mediante la programación en paralelo de la tecnología de RenderScript.

3.1.1 Funcionamiento del algoritmo

Ahora explicaremos como trabaja el algoritmo. Para ello, teniendo en cuenta que la imagen está definida como una matriz de 1 o 0, en la que 1 representa un pixel de la imagen (o un pixel de color negro) y 0 representan un espacio en blanco, empezamos por calcular los pixeles vecinos al pixel a evaluar cómo se muestra en la siguiente imagen:

P_9 $(i-1, j-1)$	P_2 $(i-1, j)$	P_3 $(i-1, j+1)$
P_8 $(i, j-1)$	P_1 (i, j)	P_4 $(i, j+1)$
P_7 $(i+1, j-1)$	P_6 $(i+1, j)$	P_5 $(i+1, j+1)$

Figura 3.1.1.Matriz de pixeles vecinos.

En esta imagen podemos ver que el pixel evaluado es el que nombraremos como P1 y que sus vecinos son todos aquellos pixeles que colindan con este.

La primera condición para ambas sub-iteraciones es: **2<=B(P1)<=6** donde **B(P1)** hace referencia al número de pixeles vecinos a P1 no blancos (es decir, que son parte de la figura, tienen valor 1). Esta condición es la que permite asegurar que no se borren pixeles que sean puntos finales o que se encuentren en el centro de la figura ya que el algoritmo lo que en realidad va haciendo es borrando los pixeles de los bordes.

La segunda condición común también a las dos sub-iteraciones es: **A(P1)=1** donde **A(P1)** representa el número de de patrones **01** en el conjunto de **P2-P3**, **P3-P4**, **P4-P5**, **P5-P6, P6-P7, P7-P8, P8-P9, P9-P2**. Esto quiere decir que tenemos que comprobar de izquierda a derecha el número de veces que se produce un cambio de un pixel con valor 0 a uno con valor 1. Un ejemplo de ello es la imagen siguiente:

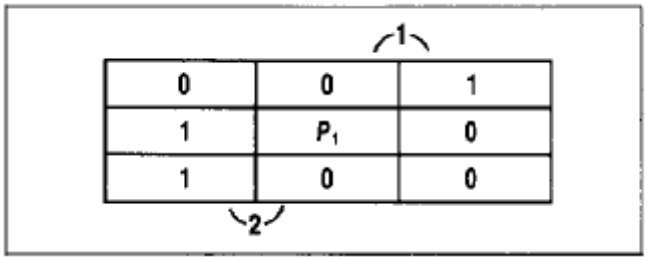


Figura 3.1.2.Matriz ejemplo patrón 01.

Como podemos observar, en este ejemplo el patrón 01 se encuentra dos veces, por lo tanto para este ejemplo la condición **A(P1)=2** lo que supone que la condición no se cumpla.

Esta condición es la encargada de que no se borren pixeles que se encuentran entre dos puntos finales.

Pasemos ahora a explicar las condiciones 3 y 4 de cada sub-iteración. En la primera sub-iteración tenemos que **P2*P4*P6=0** y que **P4*P6*P8=0,** lo que nos indica que debemos multiplicar el valor de los pixeles indicados y lo que hace es verificar que el pixel a borrar pertenece a un borde en la parte este-sur o que es parte de la esquina norte-oeste.

En el caso de la segunda sub-iteración tenemos que **P2*P4*P8=0** y que **P2*P6*P8=0** con lo que estamos evaluando que el pixel pertenezca a un borde de la parte norte u oeste o bien sea la esquina sureste.

Por último, como hemos dicho, el algoritmo finaliza cuando en las dos sub-iteraciones no se borra ningún pixel.

3.1.2 Primera sub-iteración

Se evalúan para cada pixel las cuatro siguientes condiciones, y si las cumplen, el pixel se marca para borrar:

1. 2 <= B(P1) <= 6
2. A(P1) = 1
3. P2*P4*P6 = 0
4. P4*P6*P8 = 0

Una vez evaluados todos los pixeles, se borran aquellos marcados.

3.1.3 Segunda sub-iteración

Aquí tomamos como entrada la imagen resultante de la primera sub-iteración, evaluamos cada pixel en base a las siguientes condiciones y en caso de cumplirlas marcamos el pixel para borrar:

1. 2 <= B(P1) <= 6
2. A(P1) = 1
3. P2*P4*P8 = 0
4. P2*P6*P8 = 0

Una vez evaluados todos los pixeles, procedemos a borrar todos aquellos marcados para borrar.

Como podemos apreciar, ambas iteraciones difieren únicamente en las condiciones 3 y 4.

3.2 Aplicaciones creadas

En este apartado hablaremos sobre las aplicaciones creadas, las imágenes sobre las que se probaron, las tecnologías usadas y los principales problemas encontrados.

3.2.1 Aplicaciones en java

Durante el desarrollo del proyecto se crearon dos aplicaciones en java:

1- Una aplicación para consola que simplemente pretendía ser una aproximación al algoritmo de Zhang-Suen, permitiendo familiarizarme con la implementación del mismo y sobre el que hacer pruebas para entender completamente el funcionamiento de este algoritmo. Con respecto a esta aplicación cabe destacar el hecho de que se trabajaba con una entrada de datos proveniente de ficheros de texto plano.

2- Una vez que el resultado de la aplicación de consola fue satisfactoria se procedió a adaptar esa implementación a Android así como hacer los cambios necesarios para permitir la entrada de datos mediante imágenes.

Aclarar que estás dos aplicaciones en Java representa la implementación del algoritmo Zhang-Suen en versión secuencial, lo cual supone que para cada sub-iteración se van evaluando los pixeles de uno en uno.

3.2.2 Aplicación en RenderScript

Para la aplicación codificada en RenderSrcipt se definió un kernel que es ejecutado para cada pixel y que se encarga de decidir si ese pixel pertenece

o no a la imagen resultado, es decir, en el kernel está definido el algoritmo de adelgazamiento de patrones.

Este kernel es llamado desde Java, lugar desde donde se controla el número de iteraciones o de llamadas al script que se han de realizar, también desde aquí se le provee al script de la imagen a procesar, así como del tamaño de la misma (alto y ancho), una variable para controlar que sub-iteración a de ejecutarse y un contador para registrar el número de pixeles borrados.

A continuación, el kernel se ejecuta para cada pixel, determinado si este pertenece a la imagen resultado y devolviendo dicha imagen resultado a la parte de Java desde donde supondrá la imagen de entrada para otra nueva iteración, llamando de nuevo al script con esta imagen como entrada, o desde donde será usada para mostrar el resultado del algoritmo en caso de este haber finalizado.

Capítulo 4. Resultados

En este capítulo trataremos tanto los resultados obtenidos así como la metodología para medir los mismos y las características del dispositivo en el que se llevó a cabo la ejecución de la aplicación.

4.1 Análisis de rendimiento

El análisis del rendimiento se decidió hacer usando únicamente el tiempo de ejecución del algoritmo implementado es sus dos versiones (secuencial en Java y paralelo en RenderScript) puesto que, como veremos a continuación, las diferencias entre implementaciones son tan grandes que carece de sentido emplear alguna otra métrica tales como podría ser el uso del procesador, la memoria consumida, la memoria RAM empleada por la aplicación, etc.

El dispositivo móvil sobre el que se realizaron las pruebas del algoritmo fue un Sony Xperia Z2 de aproximadamente dos años de antigüedad y cuyas características hardware podemos ver en la tabla siguiente y del mismo modo ver su aspecto presente en la figura de a continuación.

Componente	Característica
Modelo	Sony Xperia Z2
Procesador	Qualcomm MSM8974AB Snapdragon 801 quad-core 2.3GHz
GPU	Adreno 330 (cuatro cores)
RAM	3 GB
Pantalla	1080x1920pixeles, 5.2 pulgadas
Versión Android	6.0.1 Marshmallow

Tabla 4.1.Características dispositivo de prueba [9].

Figura 4.1.Apariencia dispositivo de prueba.

Las pruebas se realizaron sobre un conjunto de imágenes con extensión .jpg, que veremos a continuación en el siguiente apartado y junto a estas también se presenta la imagen resultado obtenido tras ser procesadas por el algoritmo Zhang-Suen.

Es preciso mencionar el hecho de que ambas implementaciones de este algoritmo (tanto su versión secuencial como su versión paralela) producen exactamente el mismo resultado para la misma imagen de entrada, es decir, los esqueletos obtenidos de las imágenes procesadas son iguales en ambas implementaciones, difiriendo únicamente en los tiempos de ejecución.

4.2 Resultados obtenidos

A continuación se presenta los resultados obtenidos, empezando por las algunas de las imágenes de entrada y salida del algoritmo:

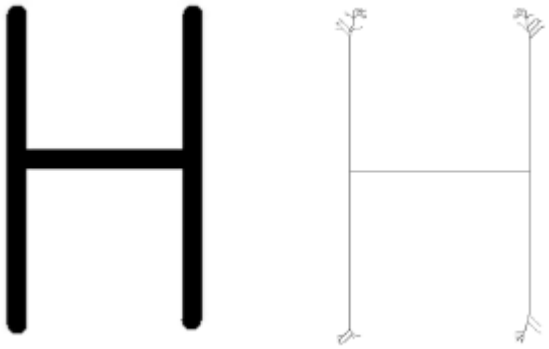


Figura 4.2.1.

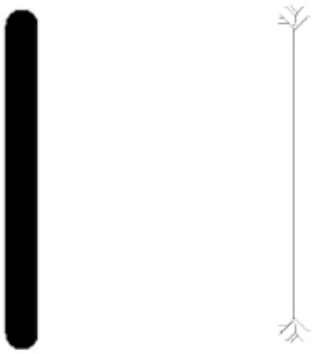


Figura 4.2.2.

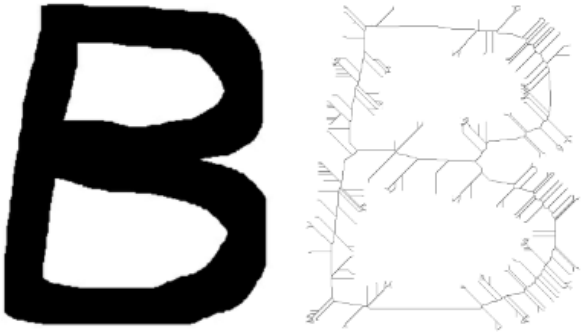


Figura 4.2.3.

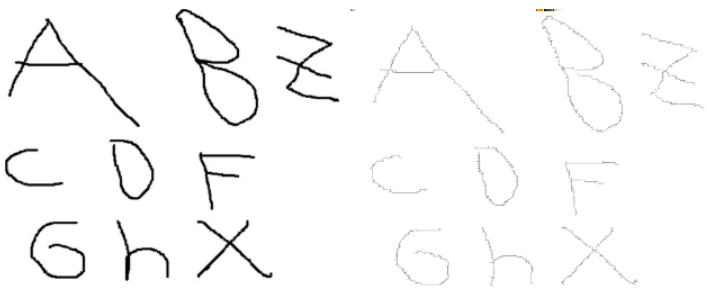


Figura 4.2.4.

Continuando bajo este párrafo podemos encontrar una tabla en la que se indican los tiempos de ejecución, en segundos, que fueron necesarios para resolver cada una de las imágenes llevadas a cabo por cada una de las versiones implementadas del algoritmo.

FIGURA	SECUENCIAL	PARALELO
1	924’09	1’64
2	1173’05	2’94
3	>1200	4’22
4	584’34	1’42

Tabla 4.2.Comparativa de tiempos.

Así mismo, se exponen seguidamente los datos obtenidos en una gráfica con el fin de poder apreciar mejor la comparativa de tiempos entre ambas implementaciones.

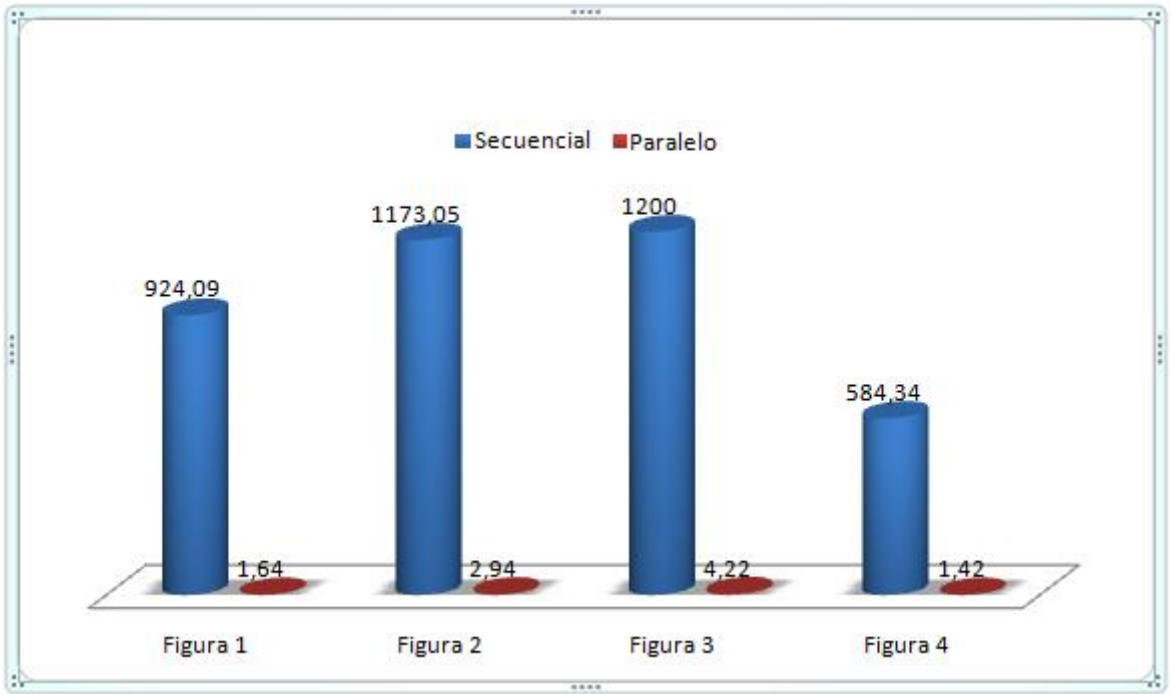


Figura 4.2.5. Gráfica de comparativa.

Como podemos apreciar, las diferencias en los tiempos de ejecución son abismales entre la versión secuencial y la paralela. Esto es debido sin

duda a que se tratan de dos enfoques de programación totalmente distintos y en el que el paralelo parte con una gran ventaja, además de que RenderScript nos provee de una tecnología, que como ya mencionamos anteriormente, se enfoca en la resolución de tareas de alto rendimiento y está muy ligado al paralelismo.

Así mismo, mientras que el secuencial se ejecuta con el lenguaje Java y sobre la MV Dalvick, el paralelo usa C y además se ejecuta a más bajo nivel, presentando una ejecución en nativo. E incluso tenemos el hecho de que RenderScript se ejecuta sobre todos los procesadores disponibles, teniendo en este caso a su disposición la CPU y la GPU.

Capítulo 5.
Conclusiones y líneas futuras

5.1 Conclusión.

Para la conclusión nos basaremos no solo en los resultados obtenidos, sino también en otros aspectos que creo que son necesarios mencionar puesto que es donde he encontrado los principales problemas a la hora de llevar a cabo el proyecto.

Con respecto a los resultados obtenidos, se hace patente que la mejora de rendimiento entre el algoritmo secuencial y su homólogo en paralelo usando la tecnología de RenderScript es increíblemente enorme. Esto sin duda debido a las características de la programación en paralelo.

Llegados a este punto se hace necesario matizar lo increíblemente sencillo que es crear un algoritmo paralelo en RenderScript, para el tratamiento de imágenes en este caso, una vez superada la curva de aprendizaje.

La curva de aprendizaje es otro de los puntos a tener en cuenta con respecto al desarrollo mediante RenderScript, puesto que esta es muy pronunciada debido a aspectos como son la falta de documentación y la escasés y sencillez de los ejemplos de código que existen.

También nos encontramos con un compilador que ofrece escaza información sobre los errores que detecta, esto unido a lo difícil que se hace la depuración nos hace estar ante un proceso de debug tedioso y complicado.

Un factor a tener en cuenta es el hecho de que a día de hoy no se sabe si Google le seguirá dando continuación a la tecnología RenderScript o si por el contrario dejarán de darle soporte y desaparecerá.

Hablando ahora desde la experiencia personal me gustaría recalcar que me fue arto complicado preparar el entorno de desarrollo para poder escribir código en RenderScript, tal vez debido al hecho de que Android Studio (el IDE en el que se desarrolló el proyecto) es relativamente joven.

Ya para finalizar hacer mención al hecho de que, una vez superada la curva inicial de aprendizaje, RenderScript ofrece una tecnología que sin duda

se debe tener en cuenta si se pretende crear aplicaciones eficientes además de que nos ofrece la posibilidad de trabajar de forma muy natural y fluida con el paralelismo.

5.2 Líneas futuras

Este proyecto puede servir como base para seguir realizando un estudio acerca de la eficiencia de las aplicaciones programadas haciendo uso de la tecnología de RenderScript. Podemos, por ejemplo, mejorar el algoritmo secuencial y optimizarlo o incluso programar en Java una versión en paralelo, con el fin de seguir realizando comparativas con su homólogo codificado en RenderScript.

Por otro lado, también se nos presenta la oportunidad de seguir desarrollando aplicaciones basadas en este algoritmo, ya que el mismo, junto con la tecnología RenderScript y las características de los dispositivos móviles que hacen uso del sistema operativo Android (principalmente smartphones y tablets) nos abren un amplio abanico de posibilidades.

Capítulo 6.
Summary and Conclusions

To conclude we will rely not only on results, but also in other aspects that I believe are necessary to mention since it is where I found the main problems in carrying out the project.

According to the results obtained, it becomes clear that the improved performance between the sequential algorithm and its equivalent in parallel using technology RenderScript is incredibly huge. This is certainly due to the characteristics of parallel programming.

At this point it is necessary to clarify how incredibly easy it is to create a parallel algorithm with RenderScript for image processing in this case, once surpassed the learning curve of course.

The learning curve is another point to consider regarding the development by RenderScript, because it is very pronounced since there is not too much documentation and the code examples are poor and simple.

We also find a compiler that offers a little information about errors that detects, this coupled with the difficulty of debugging makes a tedious and complicated debug process.

One factor to consider is the fact that today is not known if Google will continue with the RenderScript technology project or conversely cease to support it and disappear.

Speaking now from my personal experience I would like to tell that It was complicated for me to install the development environment to write code in RenderScript, perhaps due to the fact that Android Studio (the IDE in which the project was developed) is relatively young.

And to finish to mention the fact that, once the initial learning curve has been passed, RenderScript offers technology that definitely should be

consider to create efficient applications besides offering the opportunity to work very naturally and fluid with parallelism.

Capítulo 7. Presupuesto

En este apartado se refleja una aproximación del costo del proyecto. En el que se refleja el coste de los dispositivos necesarios para el desarrollo, habiendo empleado como máquina de trabajo un MacBook Pro y como dispositivo para pruebas el Sony Xperia Z2.

Así como la tarifa salarial del desarrollador tomando como referencia un precio de diez euros la hora, siendo una aproximación total de las horas necesarias de desarrollo 300.

Concepto	Cuantía
Máquina de trabajo	999 €
Smartphone para pruebas	275 €
Total salarial	3000 €
Total:	4274 €

Tabla 7.1. Resumen desglose de costes.

Bibliografía

[1] https://developer.android.com/studio/build/index.html.

[2] **https://developer.android.com/training/basics/activity-lifecycle/index.html**.

[3] **https://developer.android.com/training/articles/memory.html**.

[4] T.Y Zhang y C.Y. Suen. *A Fast Parellel Algorith from Thinning Digital Patterns*, Communications of the ACM, 1984.

[5] **http://androiddevelopers.blogspot.com.es/2011/02/introducing-renderscript.html**.

[6] **http://androiddevelopers.blogspot.com.es/2011/03/renderscript.html**.

[7] https://developer.android.com/ndk/index.html.

[8] https://developer.android.com/guide/topics/renderscript/index.html.

[9] http://www.sonymobile.com/es/products/phones/xperia-z2/features/.

[10] http://www.proyectosimio.com/es/programacion-android-ciclo-de-vida-de-una-app/

[11] https://developer.android.com/studio/build/index.html