OV5 – INP – Rapport TP4

Write an image converter application and optimize it using native code

Marleix Mathieu

Table des Matières

Introduction

L'objectif de ce travail a été d'implémenter un convertisseur d'image du format RGB vers BRG sous Android. Ce code est ensuite exécuté sur une carte Panda Board Omap v4 tournant sous une distribution Linaro. La première implémentation fournie est en langage Java auquel viennent s'ajouter une adaptation en code C natif suivi d'une implémentation en Assembleur à l'aide d'instructions Neon. Une comparaison affichant les différences de performance entre ces implémentations conclu le rapport.

Les documents relatifs TP peuvent être trouvés à l'adresse suivante :
http://www.aerian.fr/esiee/i5/ov5-inp

La programmation s'effectue sous Eclipse à l'aide des plugins SDK et NDK Android fournis par Google.

I. Installation de l'architecture

Il nous a fallu installer le SDK Android - donc je ne détaillerais pas l'installation ici - et télécharger le NDK. Pour plus d'information, voir http://developer.android.com/index.html .

Pour utiliser le NDK au sein d'Eclipse deux méthodes sont disponible, la première étant de compiler directement le code source C à la racine du projet à l'aide de l'utilitaire *ndk-build* ou d'utiliser le plugin NDK fournis par Google.

Pour des soucis d'automatisation nous avons choisi la seconde. Je reviendrais plus en détail sur cette partie lors de l'implémentation en C de l'algorithme.

II. Implémentation Java

Choisissant de récupérer l'image dans la mémoire externe de l'appareil, il nous fallait détecter le répertoire de stockage distant. Ceci peut être accomplis à l'aide de l'appelle à la fonction

« Environment.getExternalStorageDirectory(); »

Cela nous retourne alors l'adresse d'un répertoire à convertir en string. Nous utilisons alors *adb push* pour envoyer l'image, que nous allons par la suite convertir, dans la mémoire de l'appareil.

Souhaitant comparer les différents temps d'exécution, nous avons utilisé un *TextView* pour afficher les durées mesurées. Les différents résultats sont stockés au fur et à mesure du déroulement de l'application dans un *StringBuilder*. Il était d'autre part nécessaire d'afficher les deux images pour constater le résultat du programme à l'aide d'*ImageView*.

Le fichier XML contenant la vue est donc le suivant :

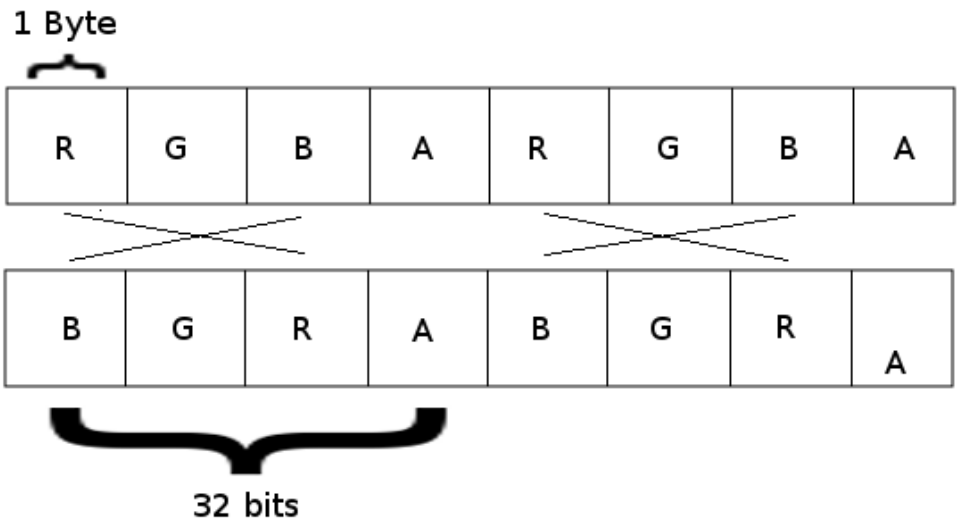
```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"
    android:background="@android:color/white" >

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="fill_parent">

            <ImageView
                android:id="@+id/src_img"
                android:layout_width="fill_parent"
                android:layout_height="match_parent"
                android:layout_weight="50"
                android:contentDescription="@string/src_img" />

            <ImageView
                android:id="@+id/dest_img"
                android:layout_width="fill_parent"
                android:layout_height="match_parent"
                android:layout_weight="50"
                android:contentDescription="@string/dest_img" />
        </LinearLayout>

        <RelativeLayout
            android:layout_width="match_parent"
            android:layout_height="fill_parent"
            android:layout_centerHorizontal="true" >

            <TextView
                android:id="@+id/textViewDuration"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:gravity="center"
                android:text="" />
        </RelativeLayout>

</RelativeLayout>
```

La classe principale en entière n'étant pas spécifiquement intéressante, nous l'avons réduite aux parties à étudier. Après avoir récupérer l'image sous forme de Bitmap à l'aide de la fonction

BitmapFactory.decodeFile

Nous avions plusieurs possibilités : accéder directement à un tableau de bytes des valeurs de l'image ou faire appel aux fonctions d'Android. Pour convertir une image en format BRG vers RGB il est donc nécessaire de comprendre comment sont stockées les différentes couleurs sous Android.

La figure suivante montre comment sont stockés les couleurs dans les deux formats et comment passé de l'un à l'autre.



A l'aide des fonctions *setPixel* et *getPixel*, nous sommes venu échanger les valeurs de chacun des pixels de l'image. La fonction *getPixel* nous renvoi un Integer contenant la valeur RGBA du pixel courant, à l'aide des décalages indiqués dans la documentation de la classe couleur d'Android consultable à l'adresse :

http://developer.android.com/reference/android/graphics/Color.html

Nous avons pu récupérer les différentes valeurs de R et B puis les échanger à l'aide d'un *setPixel*. Il a ensuite suffit d'afficher la Bitmap obtenue dans un *ImageView*. Le code obtenu peut ainsi être observé ci-dessous :

```
Public Bitmap convertBitmap(Bitmap bm){
        int bm_width = bm.getWidth(), bm_height = bm.getHeight();

        int R, G, B,A;

        for (int y = 0; y < bm_height; y++){
            for (int x = 0; x < bm_width; x++) {
                int pixel = bm.getPixel(x, y);
                A = (pixel >> 24) & 0xFF;
                R = (pixel >> 16) & 0xFF;
                G = (pixel >> 8) & 0xFF;
                B = pixel & 0xFF;
                bm.setPixel(x, y, Color.argb(A,B,G,R));
                //R,G.B - Red, Green, Blue
                //to restore the values after RGB modification, use
                //next statement
            }}
        return bm;
    }

    public void main(){
        File sdDir = Environment.getExternalStorageDirectory();
        TextView tv = (TextView) findViewById(R.id.textViewDuration);

        String strFileName = sdDir.toString() + "/LinuxAdeneoBGR.jpg";
        StringBuilder content = new StringBuilder("External Directory:
"+strFileName.toString()+"\n");

        long startTime, duration;
```

```
        startTime = System.currentTimeMillis();
        Bitmap srcBmp = BitmapFactory.decodeFile(strFileName);
        if (srcBmp == null){
            Log.i("FAILED:", "Could not find " + strFileName);
            return;
        }
        ByteBuffer Buf =
ByteBuffer.allocate(srcBmp.getHeight()*srcBmp.getWidth()*4);
        srcBmp.copyPixelsToBuffer(Buf);
        duration = System.currentTimeMillis() - startTime;

        // Computation code to be measured
        content.append(getResources().getString(R.string.time_loading)
+Long.toString(duration) + " ms\n");
        //
        ImageView iv = (ImageView) findViewById(R.id.src_img);
        iv.setImageBitmap(srcBmp);

        startTime = System.currentTimeMillis();
        Bitmap dstBmp = srcBmp.copy(Config.ARGB_8888, true);
        dstBmp = convertBitmap(dstBmp);
        //Time
        duration = System.currentTimeMillis() - startTime;
        content.append(getResources().getString(R.string.time_converting)
+Long.toString(duration) + " ms\n");
        tv.setText(content.toString());

        ImageView iv2 = (ImageView) findViewById(R.id.dest_img);
        iv2.setImageBitmap(dstBmp);
    }
}
```

III. Implémentation Native C

Pour utiliser du code C nativement sous Android, il est nécessaire de télécharger le NDK Android. Une fois cela effectué il est nécessaire dans votre projet, si vous souhaiter utiliser le plugin du NDK Android pour Eclipse, de le télécharger (il est disponible dans les dépôts fournis lors de l'ajout du plugin du SDK) et paramétrer en spécifiant le répertoire du NDK Android dans les propriétés. Il faut ensuite spécifier pour chaque projet où vous souhaiter déployer du code C qu'il s'agit d'un projet Natif via un clic droit sur le projet, Android Tools -> add Natif Support en fournissant un nom pour la librairie qui sera créée.

Eclipse va alors créer les fichiers et dossiers nécessaire au déploiement de code natif à savoir :
Le dossier JNI, un fichier C++ vide qui contiendra votre futur code et le fichier nécessaire pour la compilation le Android.mk

Nous avons tout d'abord créé la classe Java appelant la fonction native. Le code de celle-ci peut être trouvé ci-dessous :

```
package fr.aerian.imageconverter;

public class NativePictureConverter {
    static {
        System.loadLibrary("NativePictureConverter");
    }
    public native void ConvertBGRtoRGB( byte[] pSrc, byte[] pDst, int width,
int height);
}
```

Ce code est relativement simple puisque qu'il se contente de charger la librairie déclarée, ici *NativePictureConverter*, pour ensuite déclarer le prototype de la fonction native *ConvertBGRtoRGB*.

Dans notre cas, nous souhaitions utiliser un header. N'ayant pas trouvé de méthode graphique pour le générer, nous avons utilisés la méthode fournie dans le sujet, à savoir l'utilisation, au sein du répertoire « classes » du projet, de la commande :

```
javah -jni fr.aerian.imageconverter.NativePictureConverter
```

Cette commande à générer automatiquement le header correspondant au prototype défini dans la classe Java que nous avons ensuite déplacé dans le répertoire JNI.

Le code présenté précédemment, du fait des appels aux fonctions Java n'était pas adaptable en langage C, il a donc fallu repenser la manière de faire. Il était alors nécessaire de passer par des tableaux de *Bytes*. Nous passons donc par des *BufferBytes* pour récupérer les données des *Bitmap* que nous transformons ensuite en tableaux d'entrée et de sortie de bytes.

Le premier code que nous avions développé dans la classe principale était le suivant :

```
//Byte
int bytes = srcBmp.getWidth()*srcBmp.getHeight()*4;

ByteBuffer src_buffer = ByteBuffer.allocate(bytes); //Create a new buffer
ByteBuffer dest_buffer = ByteBuffer.allocate(bytes); //Create a new buffer

srcBmp.copyPixelsToBuffer(src_buffer); //Move the byte data to the buffer

byte[] src_array = src_buffer.array();
byte[] dest_array = dest_buffer.array();

startTime = System.currentTimeMillis();
NativePictureConverter nativePictureConverter = new
NativePictureConverter();

nativePictureConverter.ConvertBGRtoRGB(src_array, dest_array,
srcBmp.getWidth(), srcBmp.getHeight());

dstBmp.copyPixelsFromBuffer(ByteBuffer.wrap(dest_array));

duration = System.currentTimeMillis() - startTime;
content.append(getResources().getString(R.string.time_converting_c)
+Long.toString(duration) + " ms\n");
```

Nous copions les données de la *Bitmap* dans le buffer d'entrée et laissions le buffer de sortie vide. Au sein du code C, nous nous déplacions au sein du tableau de byte pixel par pixel et donc de quatre byte à chaque tour de boucle. Nous venions échanger les valeurs rouges et bleu en les recopiant dans le buffer de sortie.

Les valeurs alpha et vertes étaient recopiées à l'identique dans le buffer de sortie. Cela nous faisait 4 opérations à effectuer à chaque tour de boucle.

Le code C contenu dans *NativeImageConverter* était donc tel que:

```
#include "fr_aerian_imageconverter_NativePictureConverter.h"

void JNICALL
Java_fr_aerian_imageconverter_NativePictureConverter_ConvertBGRtoRGB(
        JNIEnv* env, jobject obj, jbyteArray pSrc, jbyteArray pDst, jint
width, jint height){

    jbyte *c_src = (*env)->GetByteArrayElements(env, pSrc, 0);
    jbyte *c_dest = (*env)->GetByteArrayElements(env, pDst, 0);

    int i;
    for(i = 0 ; i < width * height * 4 ; i+=4){
            *(c_dest+i) = *(c_src+i+2);     //Swap  Blue with Red
            *(c_dest+i+1) = *(c_src+i+1);   // Green equal green
            *(c_dest+i+2) = *(c_src+i);     //Swap Red with Blue
            *(c_dest+i+3) = *(c_src+i+3);   //Alpha equal Alpha
    }
}
```

La question que nous nous sommes posée était de savoir s'il n'était pas plus rapide d'initialiser aussi le buffer de sortie aux valeurs de de celui d'entrée et donc n'avoir que les valeurs à changer comme instruction dans la boucle. Réduisant le nombre de calculs effectués au sein de la boucle, nous avons pu gagner du temps d'exécution. Nous avons aussi remarqués qu'initialiser les buffers au moment du chargement nous permettait de gagner en temps d'exécution global.

IV. Implémentation Assembleur

Pour implémenter le code en assembleur, il nous a d'abord fallu comprendre comment intégrer du code ASM au sein du projet Android. Nous avons d'abord essayé via un fichier .s passé en argument au fichier Android.mk mais sans succès.

Pour pouvoir intégrer notre code nous avons donc fait appel à la fonction *asm* au sein du code C. Le prototype de celle-ci se présentait tel que :

```
asm(String codeASM : Args entrée : Args sortie : Registres)
```

Le code ASM à implémenter était un *swap* de registre en instruction *Neon*.
Le code résultant est donc :

```
int size = (width * height)/8;
        asm (
                "mov ecx, #0\t\n"
                "loop:\t\n"
                    "cmp ecx, [%2]\t\n"
                    "ja endLoop\t\n"
                    "vld4.8 {d0, d1, d2, d3}, [%0]!\t\n"
                    "vswp d0, d2\t\n"
                    "vst4.8 {d0, d1, d2, d3}, [%1]!\t\n"
                    "jmp loop\t\n"
                    "inc ecx\t\n"
                "endLoop:\t\n"
                :"+r"(c_src),"+r"(c_dest),"+r"(size)
                :
                :"d0","d1","d2","d3"
            );
```

En analysant ce code, on observe la boucle travaillant sur chaque pixel de l'image, le saut une fois la condition compteur > size, un chargement dans les registres 32B à l'aide de l'instruction vld4.8, le swap à l'aide de l'instruction éponyme puis le store à l'aide de l'instruction vstr4.8.

Les paramètres sont fournis en lecture comme le précise le « +r » et on y accède grâce à la notation *« [%numéro de paramètre] »*.

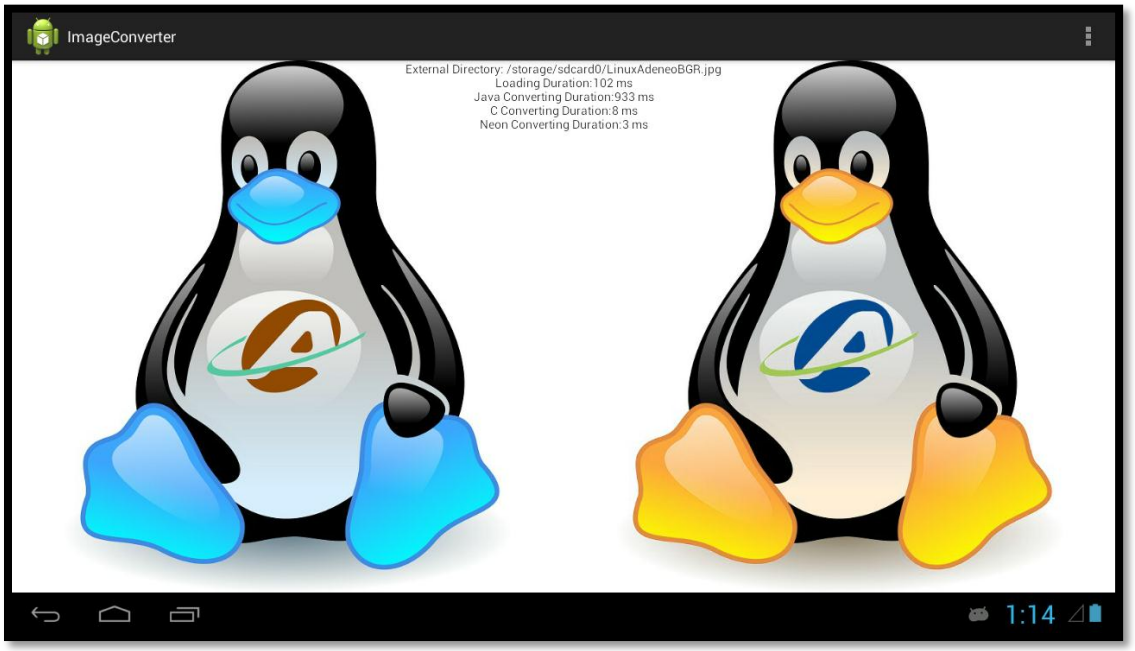
V. Comparaison des résultats

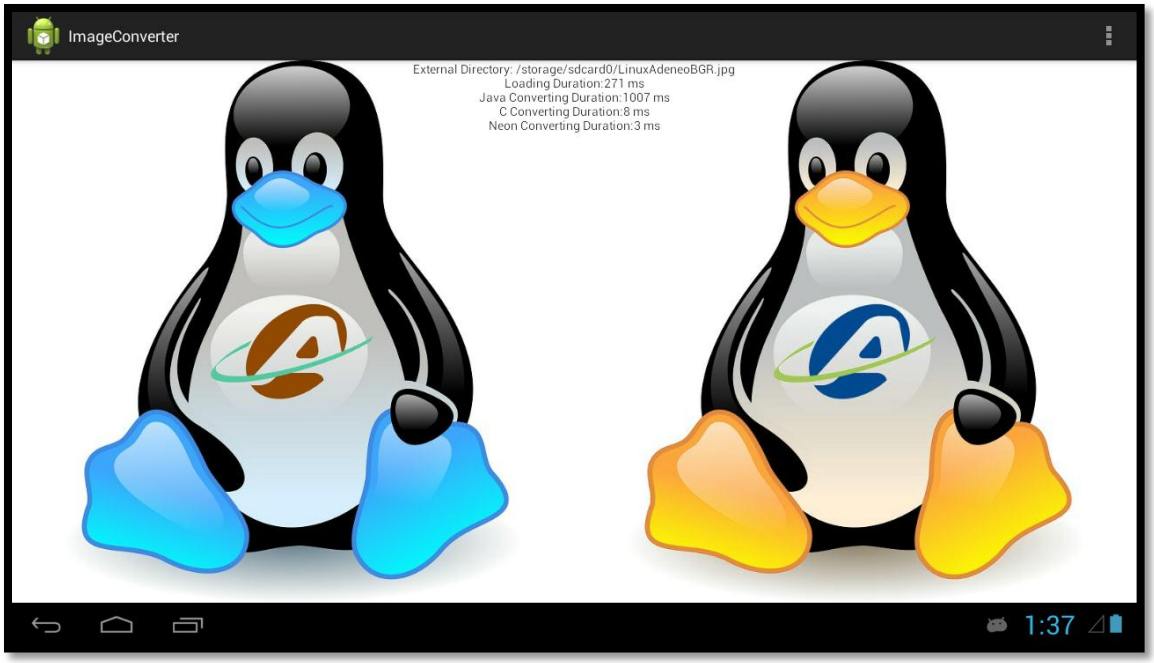
Les captures d'écran suivantes détails les différents relevés de temps obtenus lors des différentes phases d'optimisations de notre code, tout d'abord sur machine virtuelle.

On peut constater un gain de temps de facteur 4000 entre le temps d'exécution de la conversion en Java par rapport à celle en C ce qui traduit bien l'avantage considérable du développement natif sous Android dans le traitement d'image.

On observe d'autre part que le temps d'exécution globale à été réduit de 10% suite au déplacement de l'initialisation des buffers.

Puis sur la carte en elle-même avec l'ajout des résultats *ASM Neon* :





Une modification du code a été introduite pour afficher des résultats moyennés sur 10 échantillons. La moyenne des échantillons donne les résultats suivant :

- **Fonction Java** : 1007,5 ms ;
- **Fonction C :** 8,1 ms ;
- **Fonction ASM Neon :** 3,6 ms.

On constate ici un écart de temps un peu moins important entre le code Java et le code C, mais très important tout de même puisqu'on voit un écart de facteur 1000.

On observe que l'utilisation de l'assembleur *Neon* et de son traitement parallèle introduit un gain de facteur 2 par rapport au résultat en C.

Conclusion

Nous avons donc vu au cours de cet exercice comment développer et exécuter du code en C et Assembleur nativement sous Android. Cet exercice nous a permis de nous rendre compte de l'optimisation qu'il était possible d'effectuer en programmant une librairie Android directement en langage C ou Assembleur et non en Java.

Les fichiers importants du code et le fichier d'échantillon sont disponibles dans les annexes ou via l'adresse du SVN du TP.

Annexes

Fichier Résultats

Java Part (ms)	C Part (ms)	Neon part (ms)
1572	7	4
1299	7	3
849	8	3
1266	10	4
848	3	3
849	8	4
849	10	4
849	9	3
847	9	4
847	10	4

Code source

MainActivity

```
package fr.aerian.imageconverter;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.nio.ByteBuffer;

import android.app.Activity;
import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.Bitmap.Config;
import android.graphics.BitmapFactory;
import android.graphics.Color;
import android.os.Bundle;
import android.os.Environment;
import android.util.Log;
import android.view.Menu;
import android.widget.ImageView;
import android.widget.TextView;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        main();
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is
present.
        getMenuInflater().inflate(R.menu.activity_main, menu);
        return true;
    }
```

```
    public Bitmap convertBitmap(Bitmap bm){
        int bm_width = bm.getWidth(), bm_height = bm.getHeight();

        int R, G, B,A;

        for (int y = 0; y < bm_height; y++){
            for (int x = 0; x < bm_width; x++) {
                int pixel = bm.getPixel(x, y);
                A = (pixel >> 24) & 0xFF;
                R = (pixel >> 16) & 0xFF;
                G = (pixel >> 8) & 0xFF;
                B = pixel & 0xFF;
                bm.setPixel(x, y, Color.argb(A,B,G,R));
                //R,G.B - Red, Green, Blue
                //to restore the values after RGB modification, use
                //next statement
            }}
        return bm;
    }

    public void WriteSettings(Context context, String data){
         File file = new File(Environment.getExternalStorageDirectory(),
"result.txt");

         try {
            file.createNewFile();
        } catch (IOException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }
         FileWriter filewriter;
        try {
            filewriter = new FileWriter(file,true);
            filewriter.write(data);
            filewriter.close();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    public void main(){
        //Init
        long startTime, duration,initTime;
        initTime = startTime = System.currentTimeMillis();
        File sdDir = Environment.getExternalStorageDirectory();
        TextView tv = (TextView) findViewById(R.id.textViewDuration);


        String strFileName = sdDir.toString() + "/LinuxAdeneoBGR.jpg";
        StringBuilder content = new StringBuilder("External Directory:
"+strFileName.toString()+"\n");

        Bitmap srcBmp = BitmapFactory.decodeFile(strFileName);
        if (srcBmp == null){
            Log.i("FAILED:", "Could not find " + strFileName);
            try {
                throw new Exception();
            } catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
```

```
        int bytes = srcBmp.getWidth()*srcBmp.getHeight()*4;
        ByteBuffer src_buffer = ByteBuffer.allocate(bytes); //Create a new
buffer
        ByteBuffer dest_buffer = ByteBuffer.allocate(bytes); //Create a new
buffer
        srcBmp.copyPixelsToBuffer(src_buffer); //Move the byte data to the
buffer
        srcBmp.copyPixelsToBuffer(dest_buffer);
        byte[] src_array = src_buffer.array();
        byte[] dest_array = dest_buffer.array();

        //Tab
        long durationTab[] = new long[10];
        int i;

        NativePictureConverter nativePictureConverter = new
NativePictureConverter();

        duration = System.currentTimeMillis() - startTime;

        WriteSettings(this, "Neon Part\n"+String.valueOf(duration)+"ms \n");
        //Log init
        content.append(getResources().getString(R.string.time_loading)
+Long.toString(duration) + " ms\n");
        //Display init
        ImageView iv = (ImageView) findViewById(R.id.src_img);
        iv.setImageBitmap(srcBmp);

        //Java
        Bitmap dstBmp = srcBmp.copy(Config.ARGB_8888, true);
        duration = 0;
        WriteSettings(this, "Java Part\n");
        for(i = 0; i< 10; i++){
            startTime = System.currentTimeMillis();
            dstBmp = convertBitmap(dstBmp);
            durationTab[i] = System.currentTimeMillis() - startTime;
            duration += durationTab[i];
            WriteSettings(this, String.valueOf(durationTab[i])+"ms \n");
        }

        //Log Java
        content.append(getResources().getString(R.string.time_converting)
+Long.toString(duration/durationTab.length) + " ms\n");

        //C
        duration = 0;
        WriteSettings(this, "C Part\n");
        for(i = 0; i< 10; i++){
            startTime = System.currentTimeMillis();
            nativePictureConverter.ConvertBGRtoRGB(src_array, dest_array,
srcBmp.getWidth(), srcBmp.getHeight(),1);
            dstBmp.copyPixelsFromBuffer(ByteBuffer.wrap(dest_array));
            durationTab[i] = System.currentTimeMillis() - startTime;
            duration += durationTab[i];
            WriteSettings(this, String.valueOf(durationTab[i])+"ms \n");
        }

        //Log C
        content.append(getResources().getString(R.string.time_converting_c)
+Long.toString(duration/durationTab.length) + " ms\n");

        //Neon
        duration = 0;
```

```
        WriteSettings(this, "Neon Part\n");
        for(i = 0; i< 10; i++){
            startTime = System.currentTimeMillis();
            nativePictureConverter.ConvertBGRtoRGB(src_array, dest_array,
srcBmp.getWidth(), srcBmp.getHeight(),0);
            dstBmp.copyPixelsFromBuffer(ByteBuffer.wrap(dest_array));
            durationTab[i] = System.currentTimeMillis() - startTime;
            duration += durationTab[i];
            WriteSettings(this, String.valueOf(durationTab[i])+"ms \n");
        }

        //Log Neon

content.append(getResources().getString(R.string.time_converting_neon)
+Long.toString(duration/durationTab.length) + " ms\n");

        //Total Log
//      content.append(getResources().getString(R.string.time)
+Long.toString(duration+startTime-initTime) + " ms\n");

        //Setting Display & Info
        Log.i("INFO:", content.toString());
        tv.setText(content.toString());
        ImageView iv2 = (ImageView) findViewById(R.id.dest_img);
        iv2.setImageBitmap(dstBmp);
    }
}
```

ImageConverter.c

```
#include "fr_aerian_imageconverter_NativePictureConverter.h"
#include <arm_neon.h>

void JNICALL
Java_fr_aerian_imageconverter_NativePictureConverter_ConvertBGRtoRGB(
        JNIEnv* env, jobject obj, jbyteArray pSrc, jbyteArray pDst, jint
width, jint height, jint neon){

    jbyte *c_src = (*env)->GetByteArrayElements(env, pSrc, 0);
    jbyte *c_dest = (*env)->GetByteArrayElements(env, pDst, 0);
    int i = 0;
    if (neon == 0){
        int size = (width * height)/8;
        asm (
                "mov ecx, #0\t\n"
                "loop:\t\n"
                    "cmp ecx, [%2]\t\n"
                    "ja endLoop\t\n"
                    "vld4.8 {d0, d1, d2, d3}, [%0]!\t\n"
                    "vswp d0, d2\t\n"
                    "vst4.8 {d0, d1, d2, d3}, [%1]!\t\n"
                    "jmp loop\t\n"
                    "inc ecx\t\n"
                "endLoop:\t\n"
                :"+r"(c_src),"+r"(c_dest),"+r"(size)
                :
                :"d0","d1","d2","d3"
            );
    } else {
        for(i = 0 ; i < width * height * 4 ; i+=4){
            *(c_dest+i) = *(c_src+i+2);      //Swap  Blue with Red
            *(c_dest+i+2) = *(c_src+i);      //Swap Red with Blue
```

```
                }
        }
}
```