PARALLDROID

Towards a unified heterogeneous development model in Android™

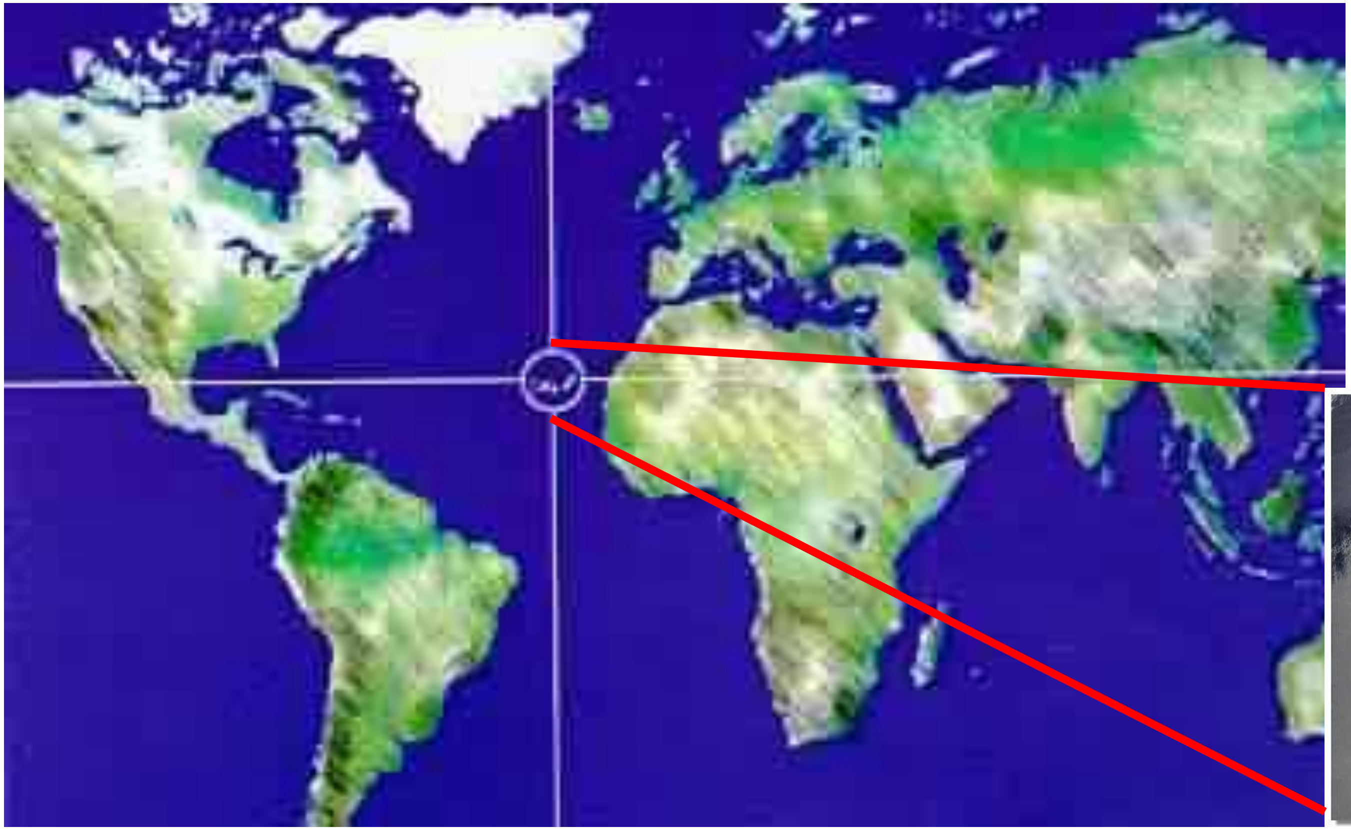
Alejandro Acosta
aacostad@ull.es

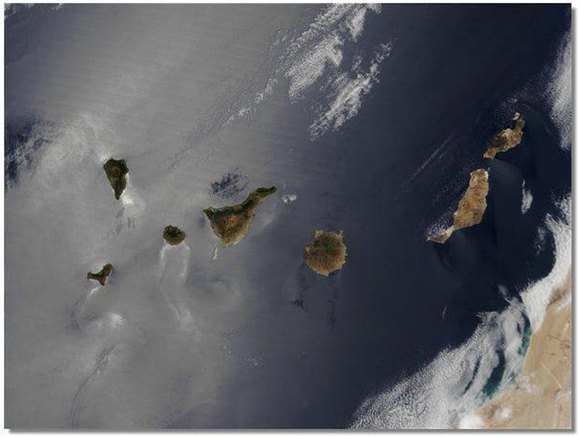
Francisco Almeida
falmeida@ull.es

Universidad de La Laguna

High Performance Computing Group

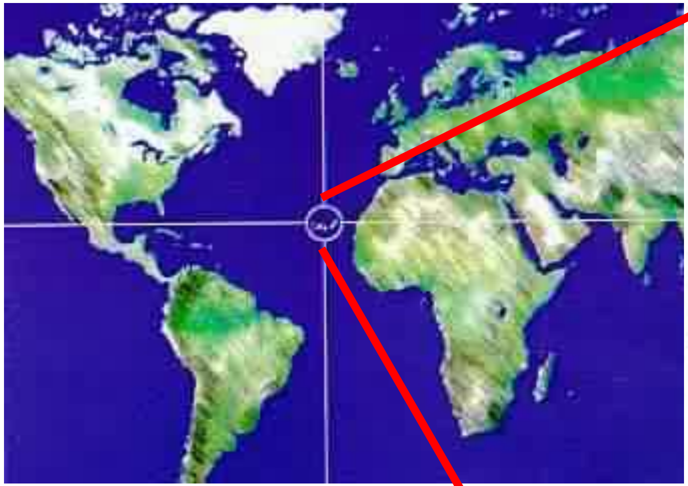
Where we come from





Who we are

High Performance Computing Group



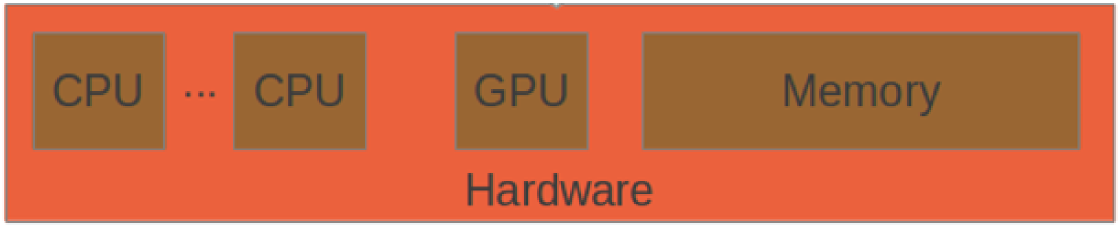
Outline

- Motivation
- Android Programming Models
- Paralldroid
  - Directives
- Extending to Classes
  - Directives
- Computational Results
- Future

MOTIVATION

The Goal: To ease the Parallelization in Mobile Devices

- Multicore + GPU



The Context: Android Devices

- Sequential programmers (no knowledge about parallelism)
- Applications demanding more computational capabilities
  - Image/Video Processing
  - Augmented Reality
  - …
- Different sequential and parallel programming models

- Difficulties - Programmability Wall
  - Developing new efficient code is a difficult task
  - Adapting existent code to new emergent architectures is also difficult

The Context: Android Devices

- Sequential programmers (no knowledge about parallelism)
- Applications demanding more computational capabilities
  - Image/Video Processing
  - Augmented Reality
  - …
- Different sequential and parallel programming models

- Difficulties - Programmability Wall
  - Developing new efficient code is a difficult task
  - Adapting existent code to new emergent architectures is also difficult

- The same scenario than in traditional scientific applications???

The Hypothesis:
To Apply the Known Methodologies

- Scientific Context:
  - Many tools developed to ease the programmer task

  - Standards (based in compiler directives) designed to simplify parallel programming
    - OpenMP: Shared memory systems
    - OpenACC: Accelerator systems

The Hypothesis:
To Apply the Known Methodologies

- Scientific Context:
  - Many tools developed to easy the programmer task
  - Standards (based in compiler directives) designed to simplify parallel programming
    - OpenMP: Shared memory systems
    - OpenACC: Accelerator systems
- To extend these ideas to the Android programming models under a unified framework

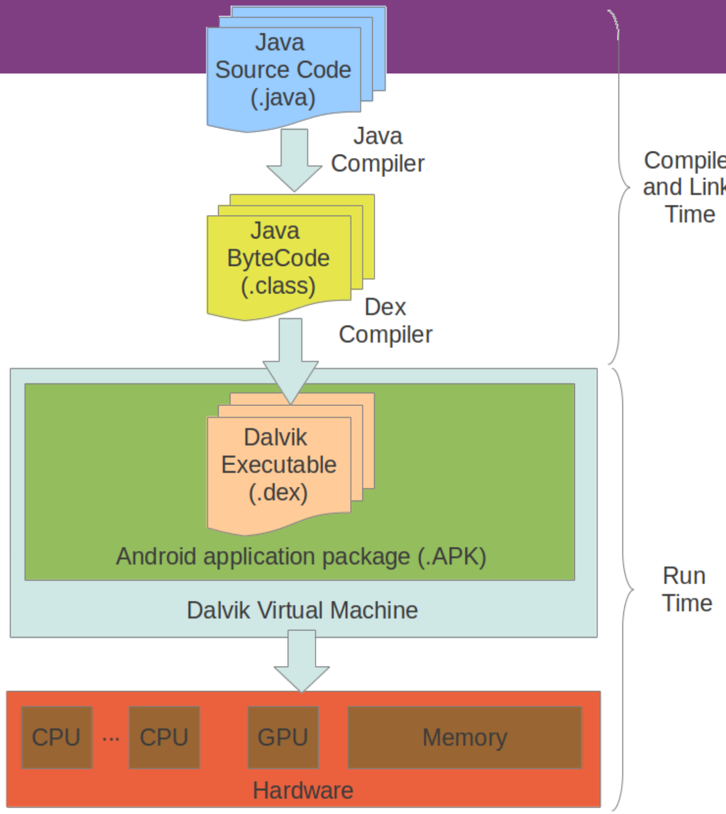
THE ANDROID PROGRAMMING MODELS

Android Programming Models

- Java (Dalvik)
- Native C
- Renderscript

Java

- Object Oriented
- Well-known.
- Rapid learning curve.
- Large developers community.
- Main programming language for Android.



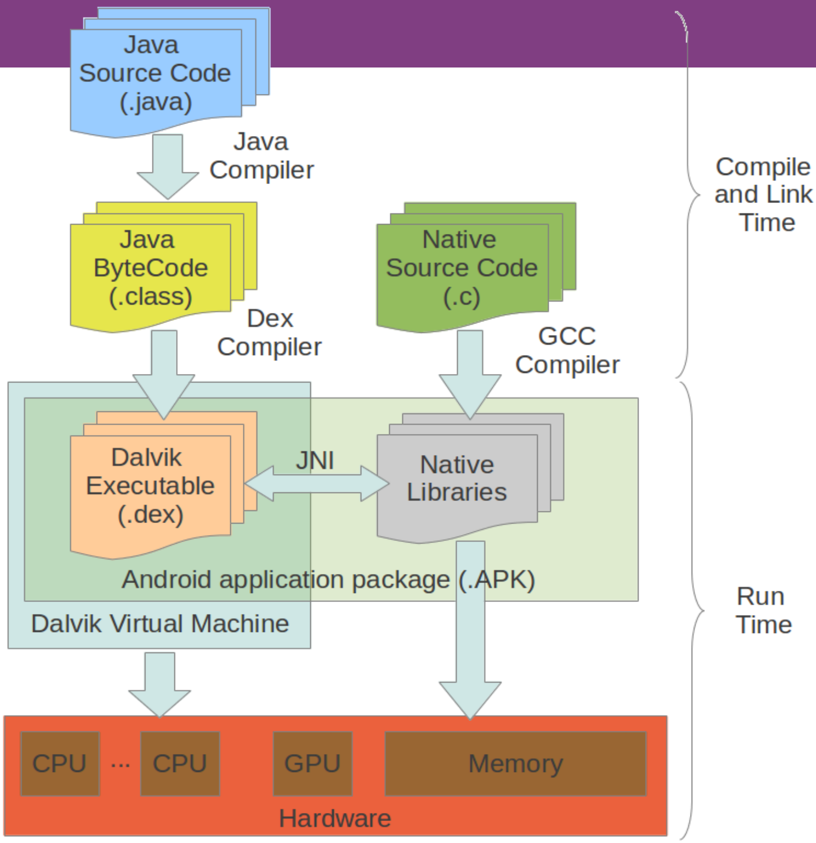
Java code

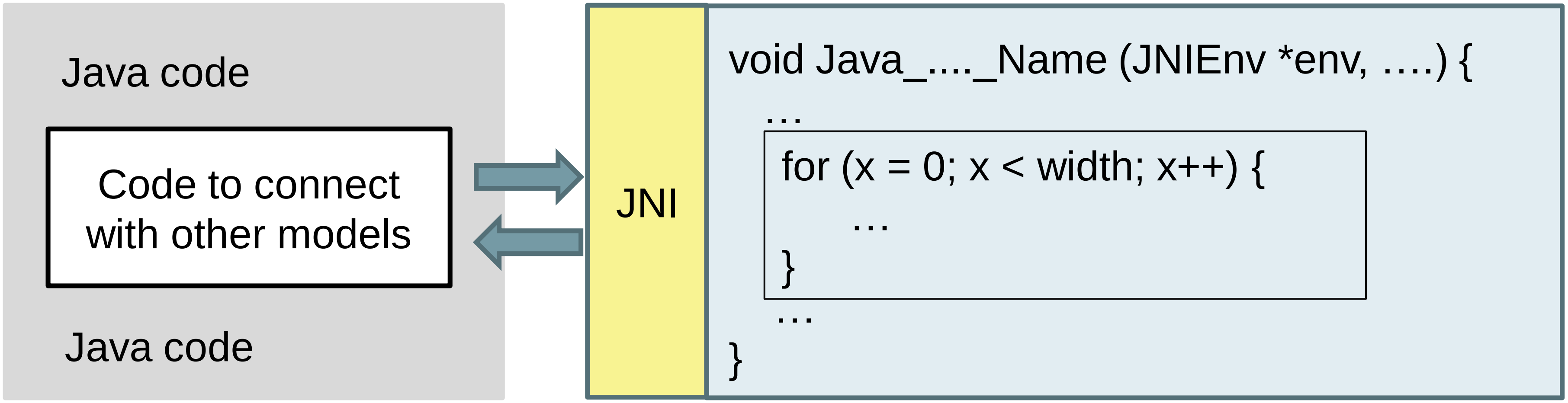
```
for (x = 0; x < width; x++) {
    for (y = 0; y < height; y++) {
        …
    }
}
```

Java code

Native C

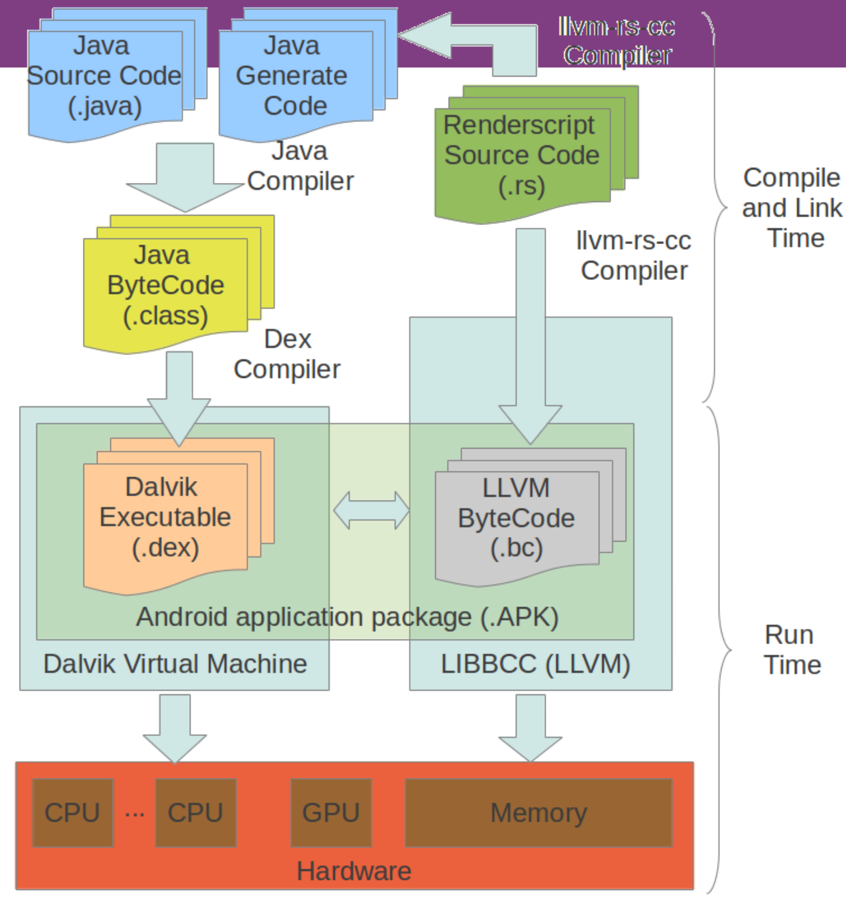
- Well-known.
- Large developers community.
- Used to implement some sections of the application.
- Use JNI.
- Interfaces to access the Java Objects
- Compatibility with C libraries
- OpenCL (Blocked on Nexus with Android 4.3)





Renderscript

- High performance.
- Similar to C with some extension.
- Support for parallel executions.
- Support for GPU executions.
- Some Java Objects are ported to the Renderscript layer



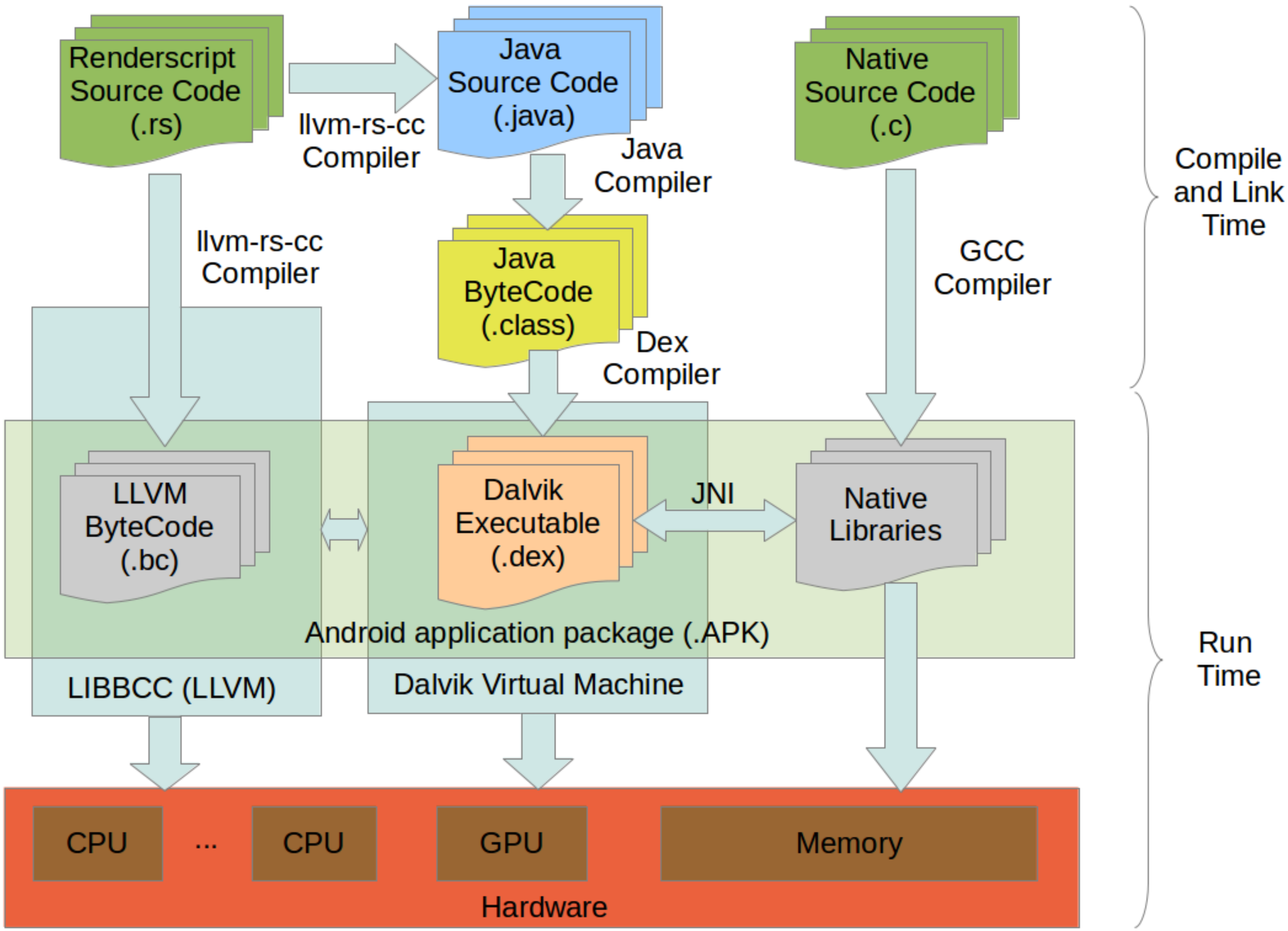
Java code

Renderscript Context
Allocate Memory
Call to the Renderscript funcion

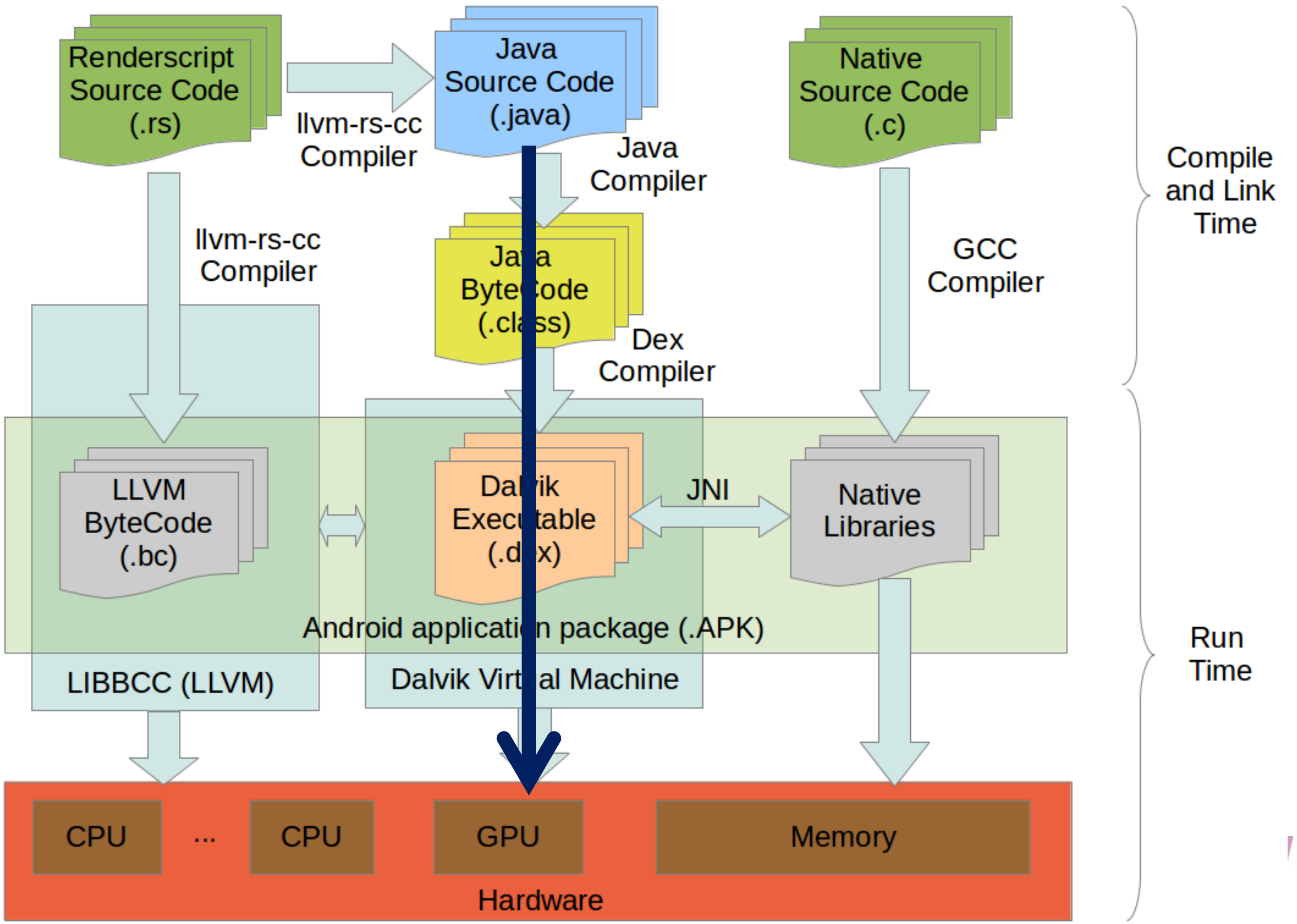
Java code

```
void root(const uchar4 *v_in, uchar4 *v_out) {
    …
    for (int  x = 0; x < width; x++) {
        …
    }
    …
}
```

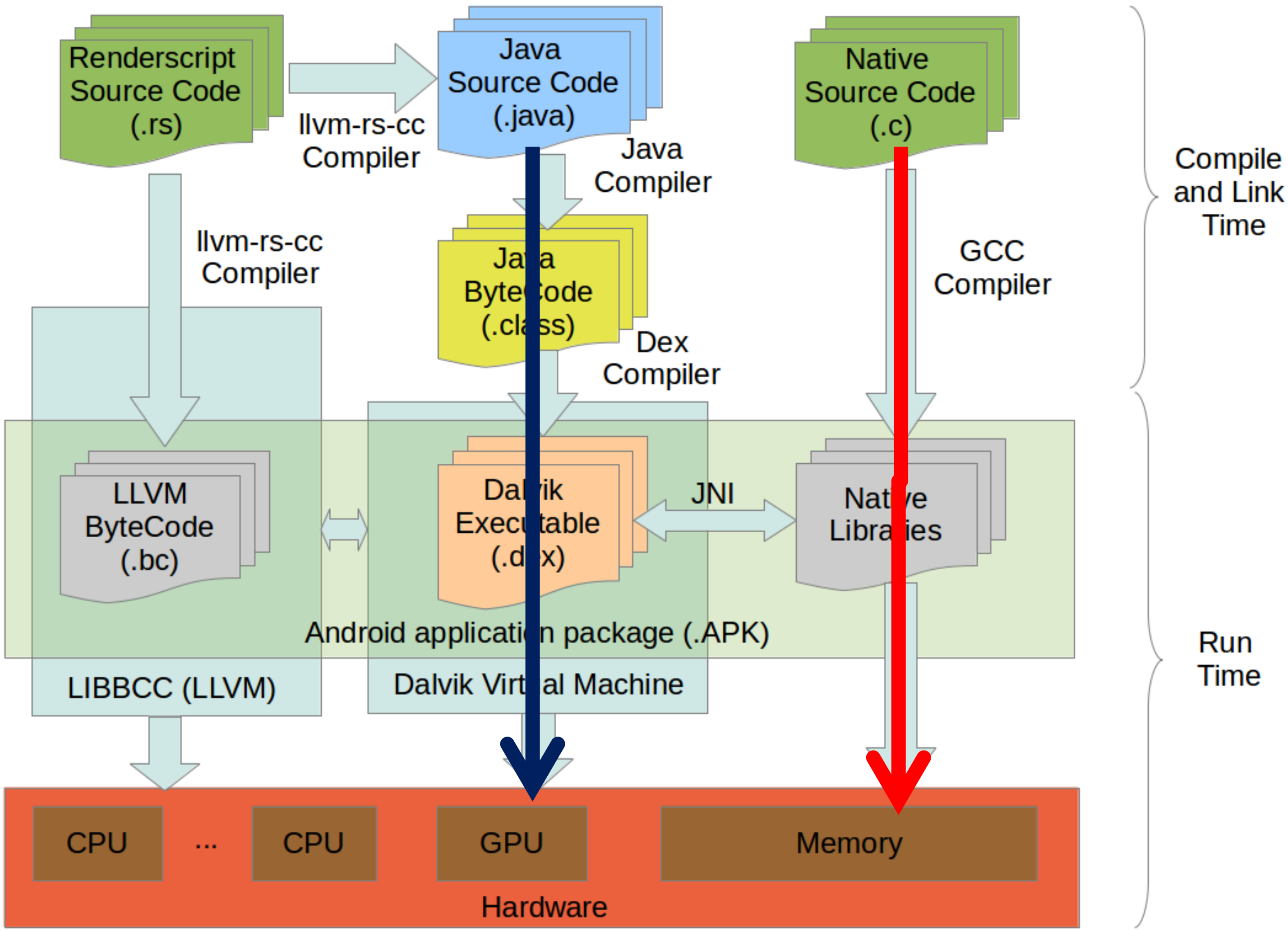
Compilation Process



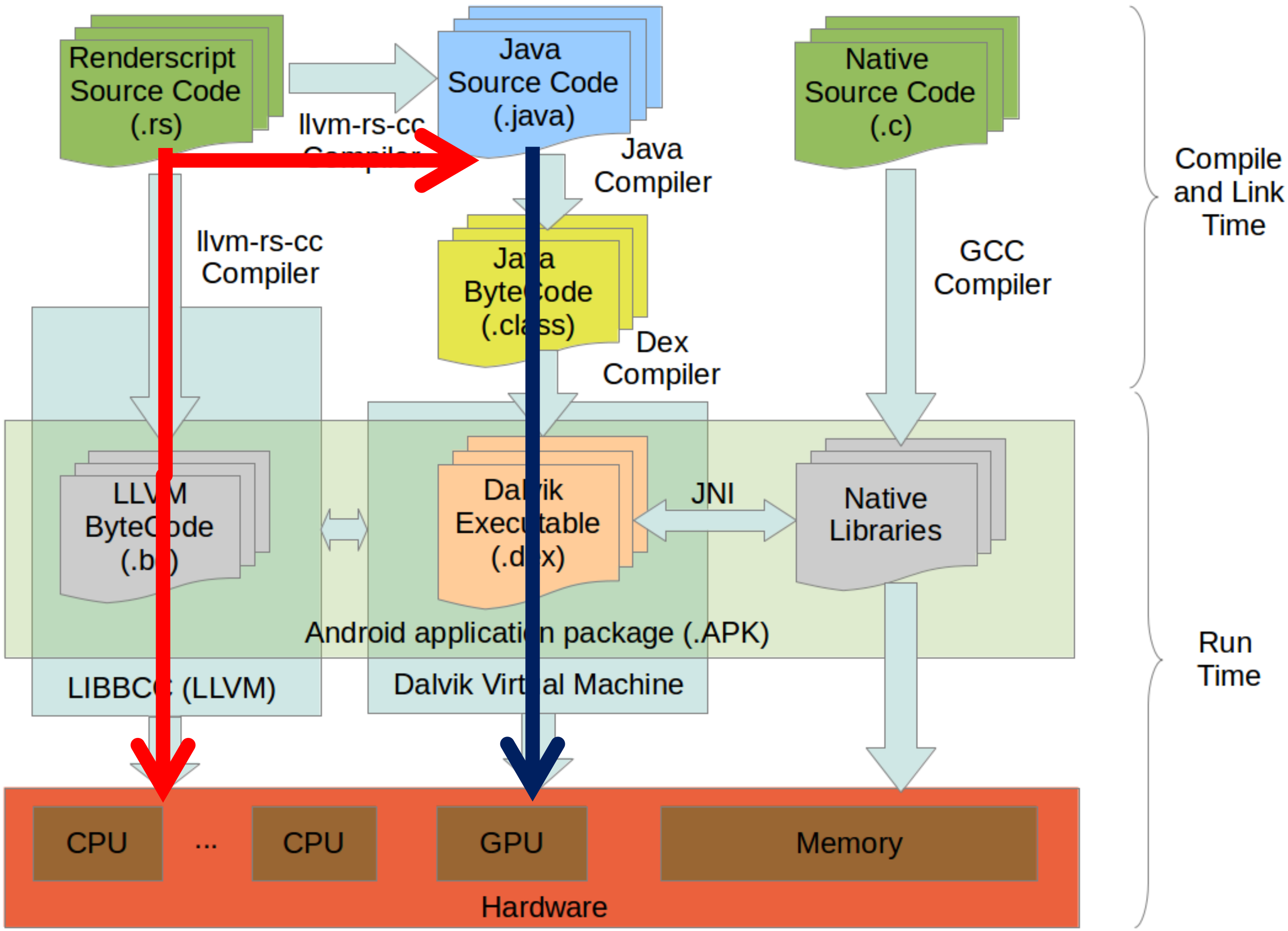
Compilation Process



Compilation Process



Compilation Process



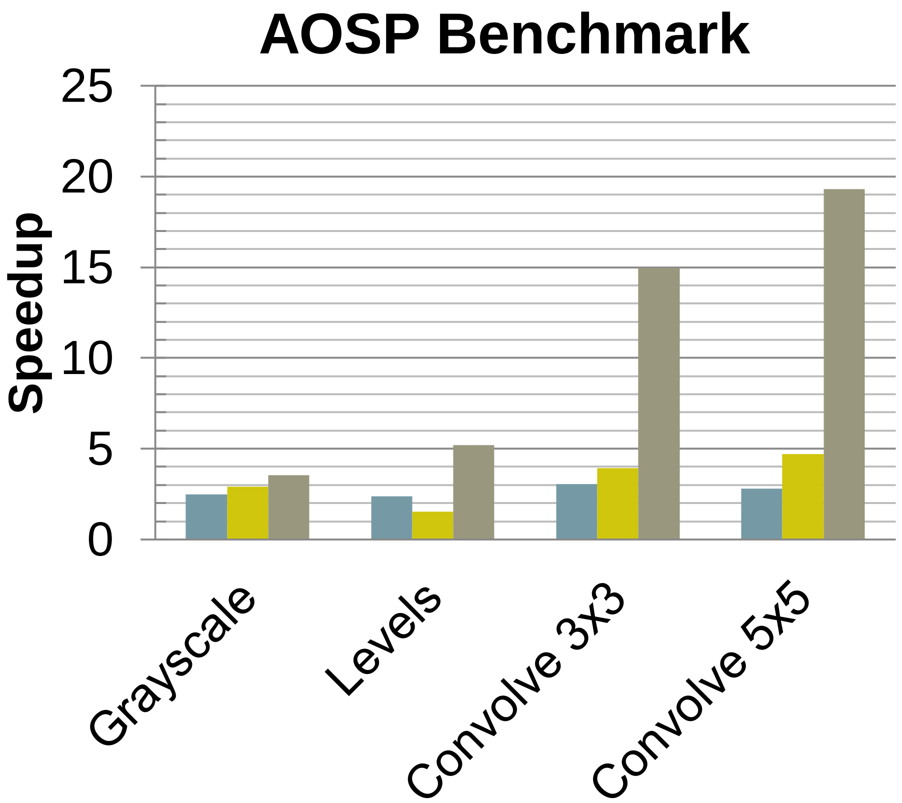
The Test: Nexus 7

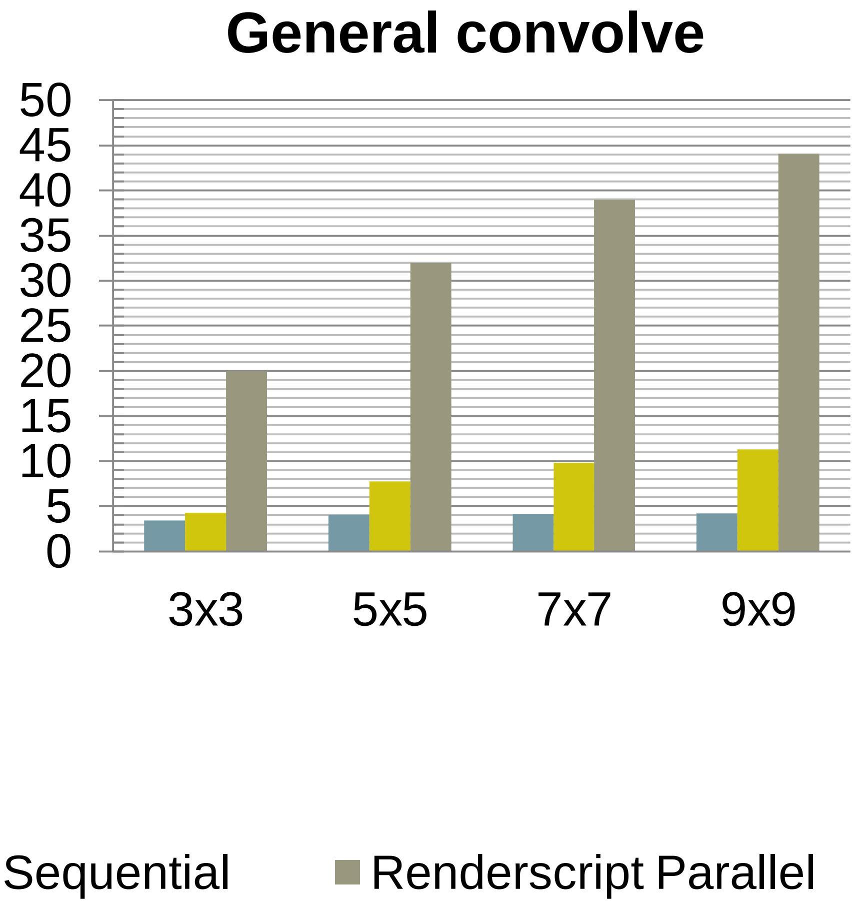
- Renderscript ImageProcessing benchmark
  (AOSP: frameworks/base/tests/RenderScriptTests/ImageProcessing)
  - Grayscale
  - Convolve 3x3
  - Convolve 5x5
  - Levels
- General Convolve
  - 3x3
  - 5x5
  - 7x7
  - 9x9

Java (Dalvik)
Native C
Renderscript

Gray scale

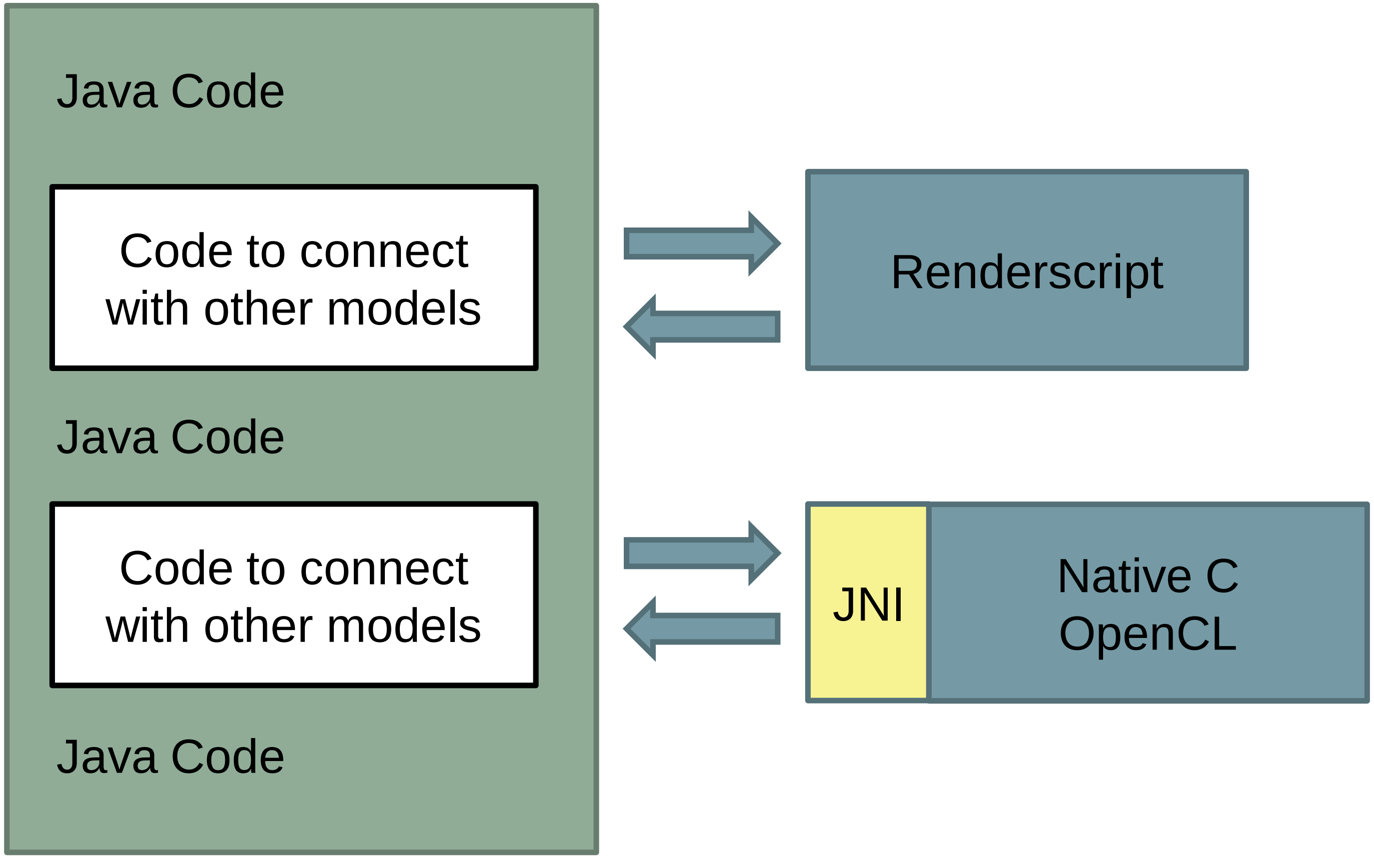
1600x1067





Programming models	CPU 1	CPU 2	CPU 3	CPU 4
Java	100%	Offline	Offline	Offline
Native C	100%	Offline	Offline	Offline
Renderscript sequential	100%	Offline	Offline	Offline
Renderscript Parallel	100%	100%	100%	100%

Implementation

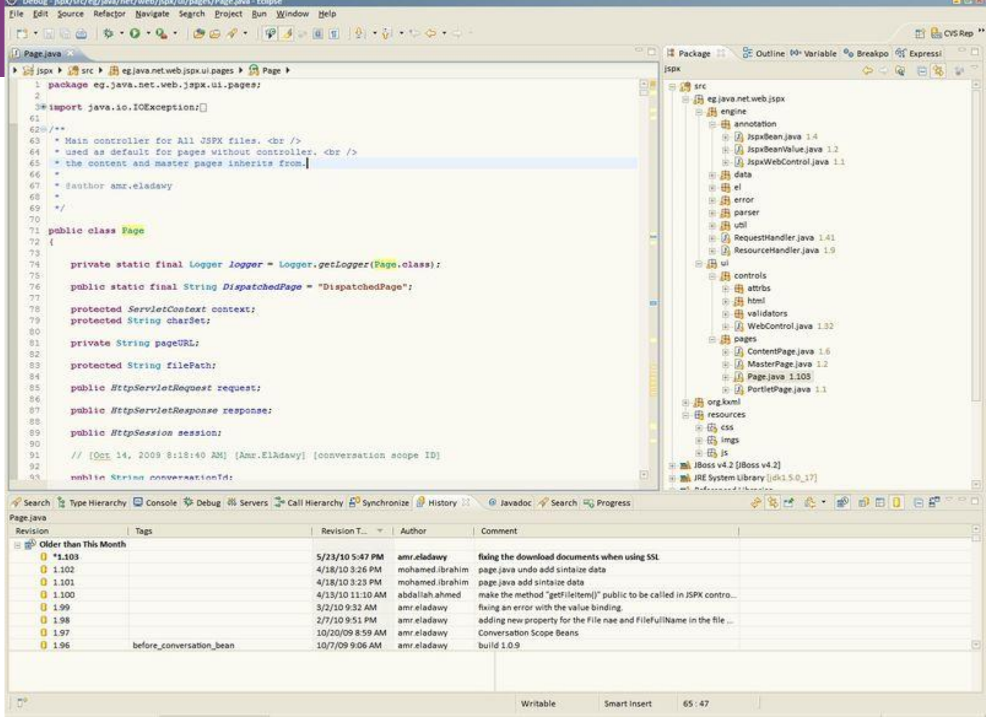


PARALLDROID

Towards a unified heterogeneous development model in Android. HeteroPar 2013

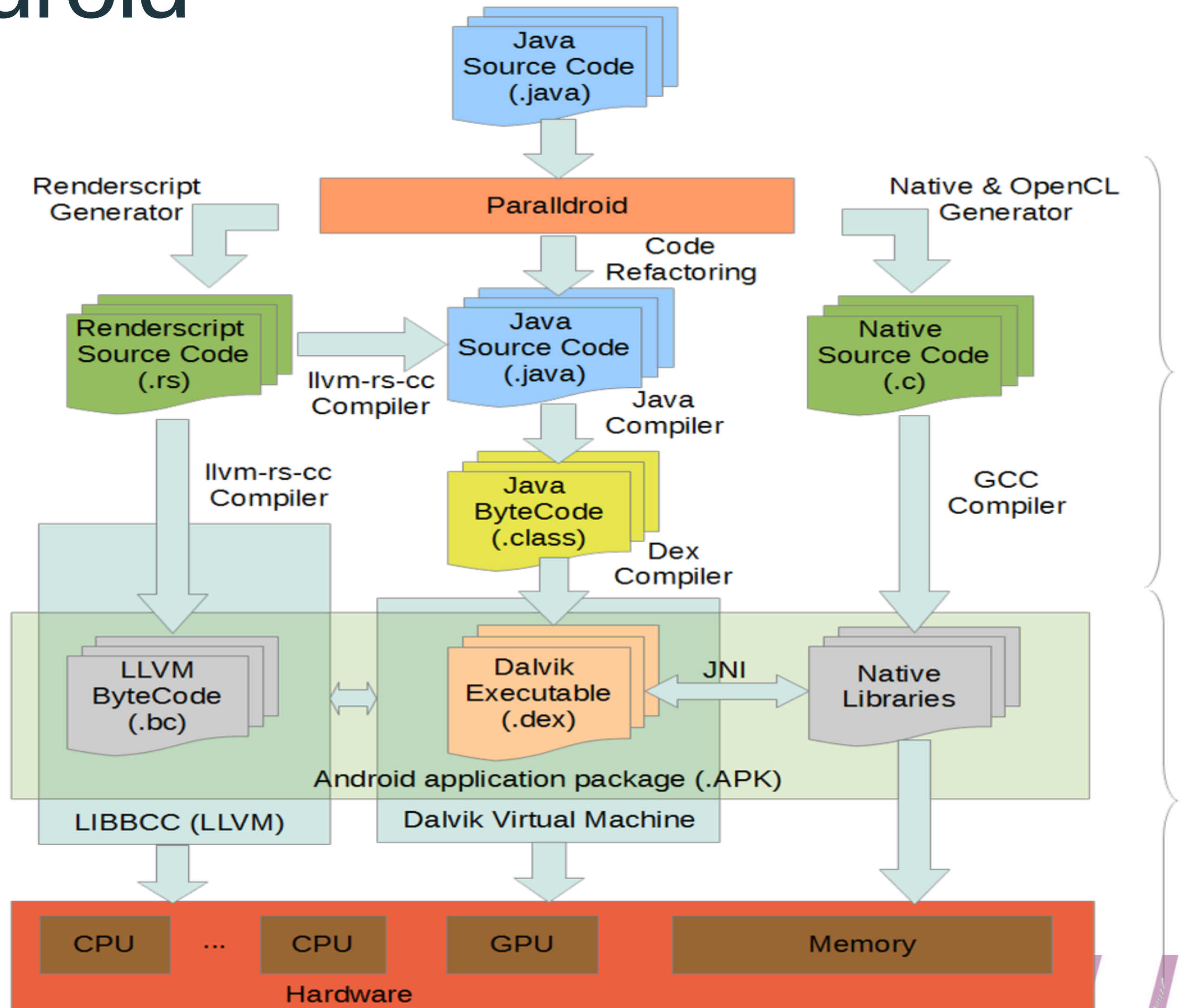
Paralldroid: A Framework for Parallelism in Android. LEAP 2013. (Low Energy Application Parallelism)

Paralldroid



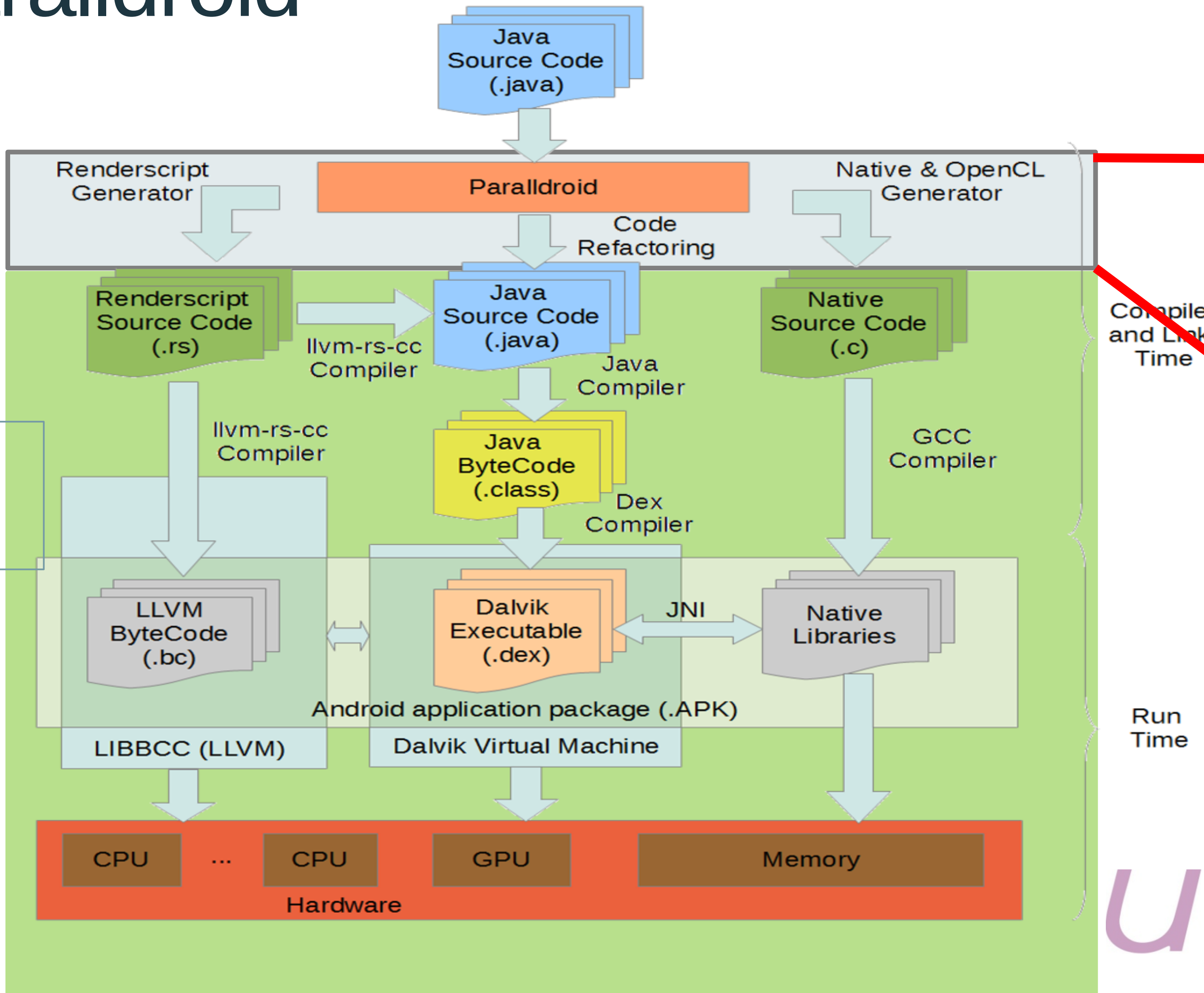
- Framework based on Java.
- Source to Source translator.
- Eclipse plugin.
- Annotated Java code.
- OpenMP 4.0 extension
- Advantages:
  - Increased use of the parallel devices by non-expert users.
  - Rapid inclusion of emerging technology into their systems.
  - Delivery of new applications due to the rapid development time.
  - Unify the different programming models of Android.
- Disadvantages:
  - Less performance compared with an adhoc version (at a low effort).
  - Eclipse dependency.

Paralldroid



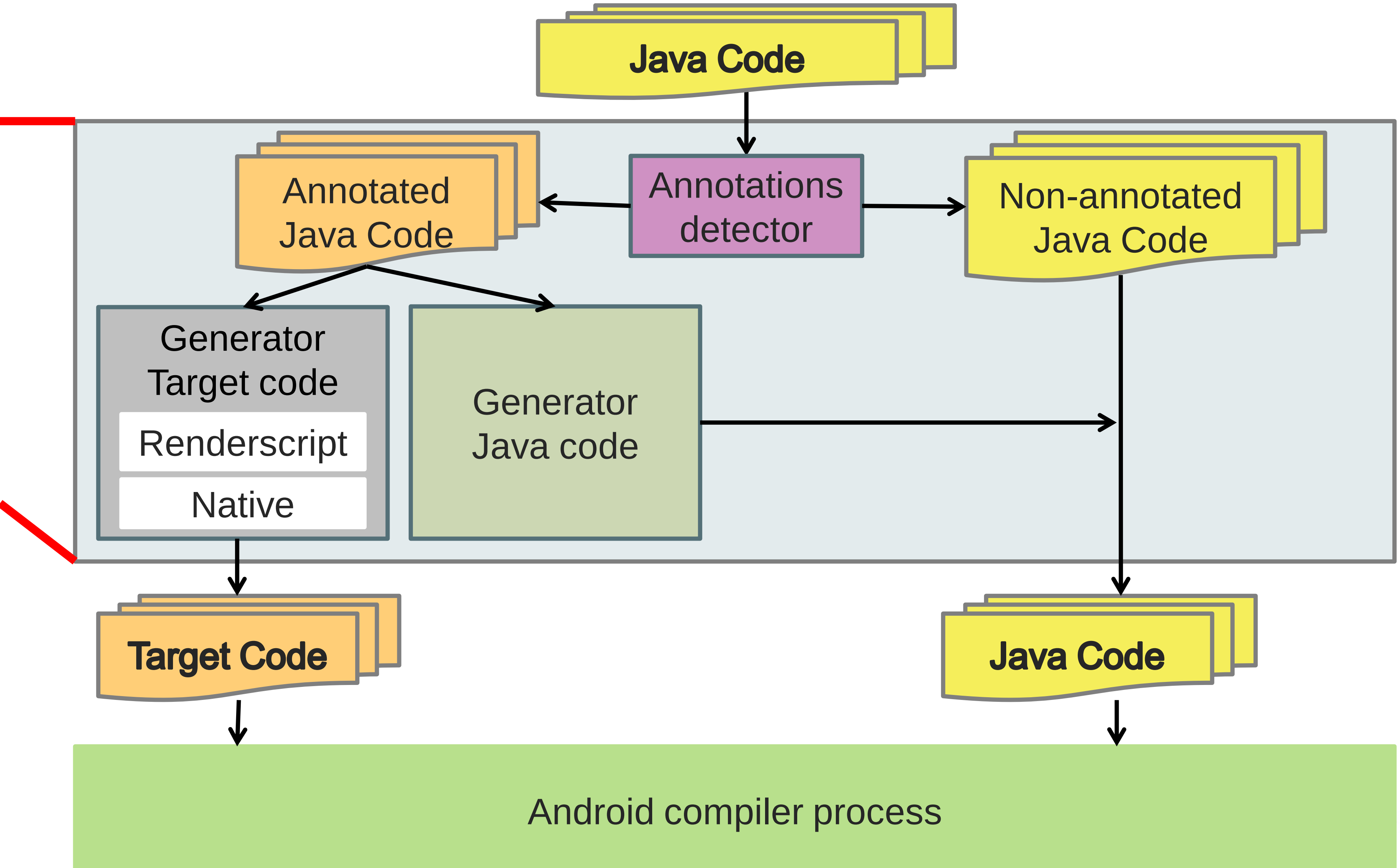
Android Compilation Process

Paralldroid

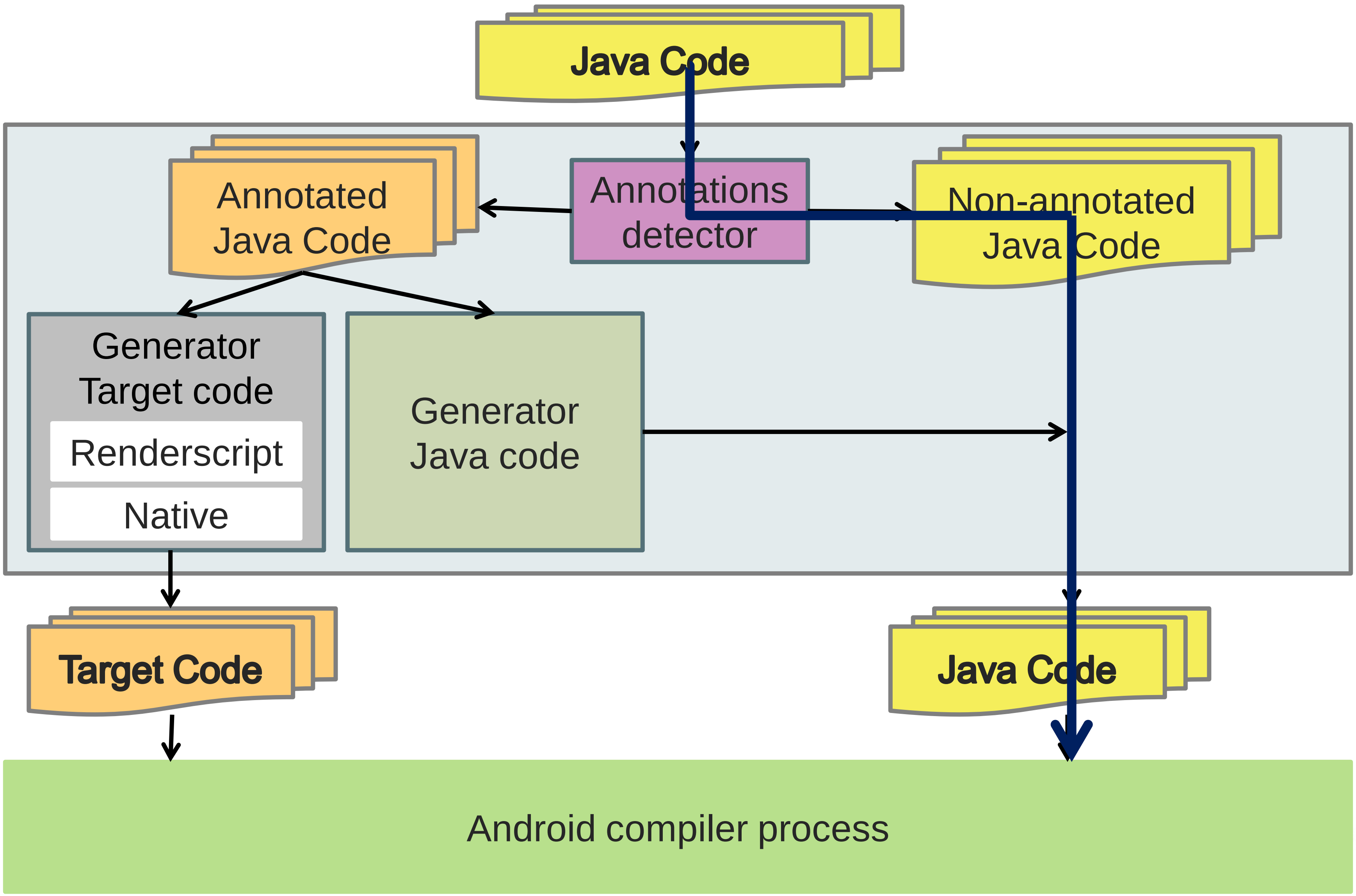


Android Compilation Process

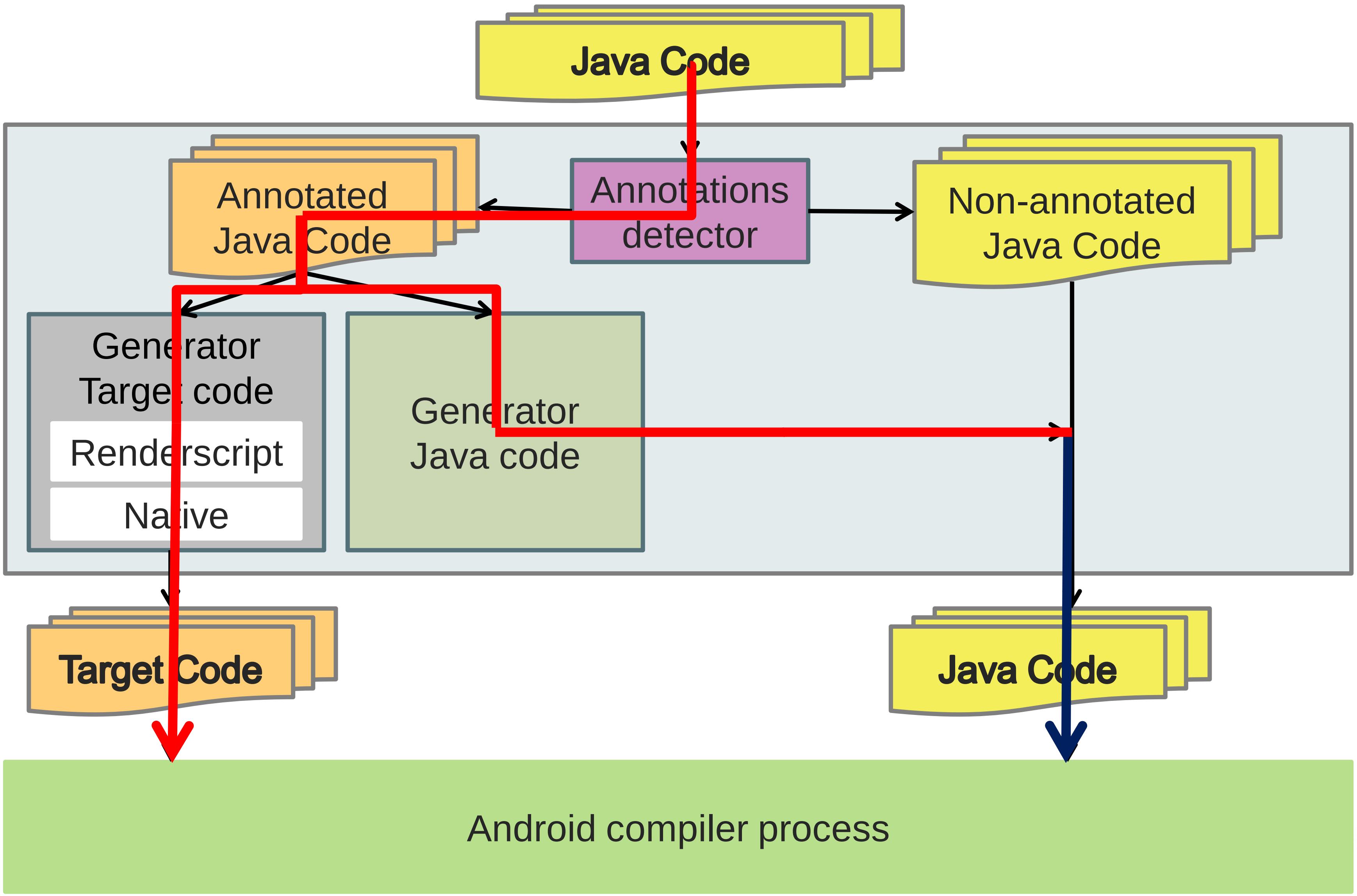
Paralldroid compiler process



Non-annotated Java code



Annotated Java code

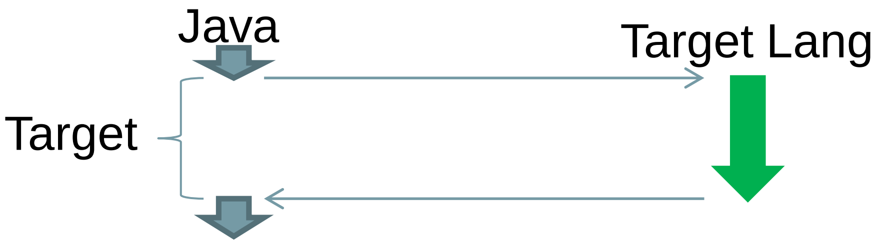


Directive: Target data

- Create a data environment.
- Mapping data to the target context.
- Clauses
  - Lang
    - Extension to the OpenMP standard,
    - Target language (Renderscript, Native or OpenCL)
  - Map
    - Maps a variable from the current Java context to the target data context.
    - Map types: Alloc, To, From, ToFrom (default).

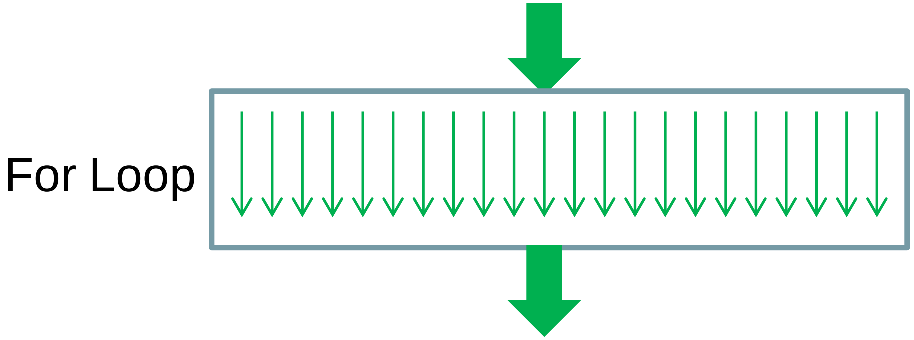
Directive: Target

- Create a data environment.
- Mapping data to the target context.
- Execute the code associate to the directive
- Clauses
  - Lang
    - Extension to the OpenMP standard,
    - Target language (Renderscript, Native or OpenCL)
  - Map
    - Maps a variable from the current Java context to the target data context.
    - Map types: Alloc, To, From, ToFrom (default).



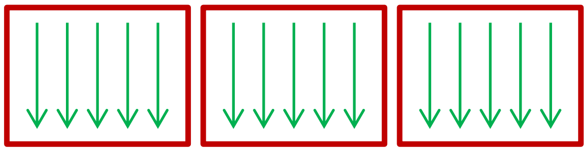
Directive: Parallel for

- Used in the context of a target directive
- Distributing the load of the for loop between the threads available
- Clauses
  - Private
  - Firstprivate
  - Shared
  - Colapse
  - Rsvector
    - Extension to the OpenMP standard.
    - Input and output vectors used in Renderscript.



Directive: Teams

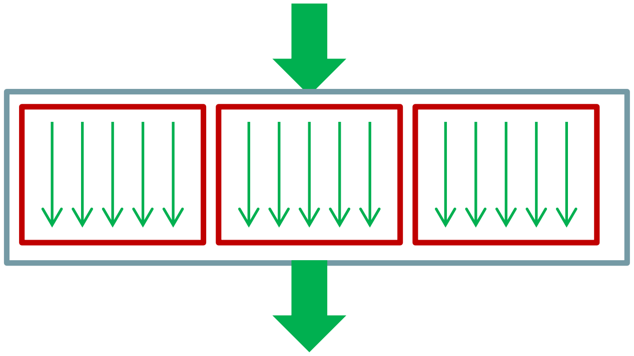
- Used in the context of a target directive
- Create teams or groups of threads.
- Clauses
  - Num_teams
  - Num_thread
  - Private
  - Firstprivate
  - Shared.



Directive: Distribute

- Used in the context of a teams directive
- Distributing the load of the for loop between the teams available
- Clauses
  - Private
  - Firstprivate
  - Colapse

For Loop



Grayscale Java implementation

```
public void grayscale() {
   int pixel, sum, x;
   int [] scrPxs = new int[width*height];
   int [] outPxs = new int[width*height];
   bitmapIn.getPixels(scrPxs, 0, width, 0, 0, width, height);

   for(x = 0; x < width*height; x++) {
      pixel = scrPxs[x];
      sum = (int)(((pixel) & 0xff) * 0.299f);
      sum += (int)(((pixel >> 8 ) & 0xff) * 0.587f);
      sum += (int)(((pixel >> 16) & 0xff) * 0.114f);
      outPxs[x] = (sum) + (sum << 8) + (sum << 16) + (scrPxs[x] & 0xff000000);
   }

   bitmapOut.setPixels(outPxs, 0, width, 0, 0, width, height);
}
```

Grayscale Java implementation

```
public void grayscale() {
   int pixel, sum, x;
   int [] scrPxs = new int[width*height];
   int [] outPxs = new int[width*height];
   bitmapIn.getPixels(scrPxs, 0, width, 0, 0, width, height);

   for(x = 0; x < width*height; x++) {
      pixel = scrPxs[x];
      sum = (int)(((pixel) & 0xff) * 0.299f);
      sum += (int)(((pixel >> 8 ) & 0xff) * 0.587f);
      sum += (int)(((pixel >> 16) & 0xff) * 0.114f);
      outPxs[x] = (sum) + (sum << 8) + (sum << 16) + (scrPxs[x] & 0xff000000);
   }

   bitmapOut.setPixels(outPxs, 0, width, 0, 0, width, height);
}
```

Native

```
public void grayscale() {
   int pixel, sum, x;
   int [] scrPxs = new int[width*height];
   int [] outPxs = new int[width*height];
   bitmapIn.getPixels(scrPxs, 0, width, 0, 0, width, height);

   // pragma paralldroid target lang(native) map(alloc:x,pixel,sum)
   for(x = 0; x < width*height; x++) {
      pixel = scrPxs[x];
      sum = (int)(((pixel) & 0xff) * 0.299f);
      sum += (int)(((pixel >> 8 ) & 0xff) * 0.587f);
      sum += (int)(((pixel >> 16) & 0xff) * 0.114f);
      outPxs[x] = (sum) + (sum << 8) + (sum << 16) + (scrPxs[x] & 0xff000000);
   }

   bitmapOut.setPixels(outPxs, 0, width, 0, 0, width, height);
}
```

Renderscript

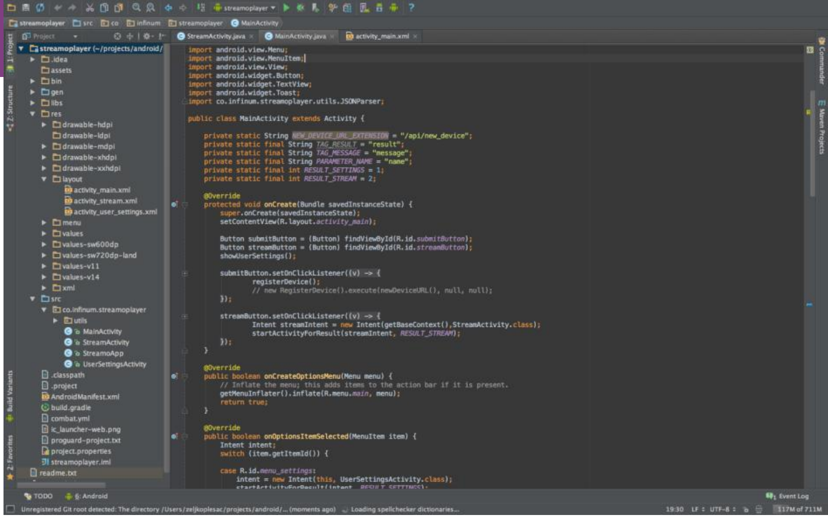
```
public void grayscale() {
   int pixel, sum, x;
   int [] scrPxs = new int[width*height];
   int [] outPxs = new int[width*height];
   bitmapIn.getPixels(scrPxs, 0, width, 0, 0, width, height);

   // pragma paralldroid target lang(rs) map(to:scrPxs,width,height) map(from:outPxs)
   // pragma paralldroid parallel for private(x,pixel,sum) rsvector(scrPxs,outPxs)
   for(x = 0; x < width*height; x++) {
      pixel = scrPxs[x];
      sum = (int)(((pixel) & 0xff) * 0.299f);
      sum += (int)(((pixel >> 8 ) & 0xff) * 0.587f);
      sum += (int)(((pixel >> 16) & 0xff) * 0.114f);
      outPxs[x] = (sum) + (sum << 8) + (sum << 16) + (scrPxs[x] & 0xff000000);
   }

   bitmapOut.setPixels(outPxs, 0, width, 0, 0, width, height);
}
```

ONE STEP FORWARD: EXTENDING TO CLASSES

Paralldroid



Android Studio

- New implementation.
- Extension of OpenJDK.
- Use Java annotations. (@annotations)
- Based on OpenMP directives.
- Apply OpenMP directives into definitions of classes.
- Advantages:
  - All advantages of previous implementation.
  - No Eclipse dependency.
  - More easy to understand for Java programmers.
- Disadvantages:
  - Less performance compared with an adhoc version.
  - Currently no support for statement annotations.

Grayscale Java implementation

```java
public class GrayScale {
    private final float gMonoMult[] = {0.299f, 0.587f, 0.114f};
    private int width;
    private int height;

    public GrayScale (int width, int height) {
        this.width = width; this.height = height;
    }

    public void runTest(int scrPxs[], int outPxs[]) {
        int x;  int acc;
        for(x = 0; x < width*height; x++) {
            acc = (int)(((scrPxs[x]) & 0xff) * gMonoMult[0]);
            acc += (int)(((scrPxs[x] >> 8 ) & 0xff) * gMonoMult[1]);
            acc += (int)(((scrPxs[x] >> 16) & 0xff) * gMonoMult[2]);
            outPxs[x] = (acc) + (acc << 8) + (acc << 16) + (scrPxs[x] << 24);
        }
    }
}
```

Directive: Target data

```
@Target(RENDERSCRIPT)
public class GrayScale {
    private final float gMonoMult[] = …;

    @Map(TO)
    private int width;

    @Map(TO)
    private int height;

    public GrayScale (Activity act, int width, int height) {
        …
    }

    public void runTest(int scrPxs[], int outPxs[]) {
        …
    }
}
```

- Constructor method initializes the target context, allocates memory and copy initial values.
- Generate getter or setter methods.
- Generate finalize method that free memory and destroy the target context.

Directive: Target

```
@Target(RENDERSCRIPT)
public class GrayScale {
    private final float gMonoMult[] = …;

    @Map(TO)
    private int width;

    @Map(TO)
    private int height;

    public GrayScale (Activity act, int width, int height) {
        …
    }

    public void runTest(@Map(TO) int scrPxs[], @Map(FROM) int outPxs[]) {
        …
    }
}
```

- Allocate memory and copy values to target context.
- Execute method.
- Copy values from target context and free memory.

Directive: Parallel

```
@Target(RENDERSCRIPT)
public class GrayScale {
    private final float gMonoMult[] = …;
    @Map(TO)
    private int width;
    @Map(TO)
    private int height;
    public GrayScale (int width, int height) {
        …
    }
    @Parallel
    public void runTest(@Input int scrPxs[], @Output int outPxs[], @index int x, …) {
        int x;  int acc;
        acc = (int)(((scrPxs[x]) & 0xff) * gMonoMult[0]);
        acc += (int)(((scrPxs[x] >> 8 ) & 0xff) * gMonoMult[1]);
        acc += (int)(((scrPxs[x] >> 16) & 0xff) * gMonoMult[2]);
        outPxs[x] = (acc) + (acc << 8) + (acc << 16) + (scrPxs[x] << 24);
    }
}
```

- Execute method in parallel.
- Function is executed many times as elements contain the input or output vectors.

Directive: Declare

- Fields or methods that are declared only in the target context.

```
@Target(RENDERSCRIPT)
public class GrayScale {
    @Declare
    private final float gMonoMult[] = …;
    @Map(TO)
    private int width;
    @Map(TO)
    private int height;
    public GrayScale (int width, int height) { … }

    @Parallel
    public void runTest(@Input int scrPxs[], @Output int outPxs[], @index int x, …) {
        int x;  int acc;
        acc = (int)(((scrPxs[x]) & 0xff) * gMonoMult[0]);
        acc += (int)(((scrPxs[x] >> 8 ) & 0xff) * gMonoMult[1]);
        acc += (int)(((scrPxs[x] >> 16) & 0xff) * gMonoMult[2]);
        outPxs[x] = (acc) + (acc << 8) + (acc << 16) + (scrPxs[x] << 24);
    }
}
```

COMPUTATIONAL RESULTS

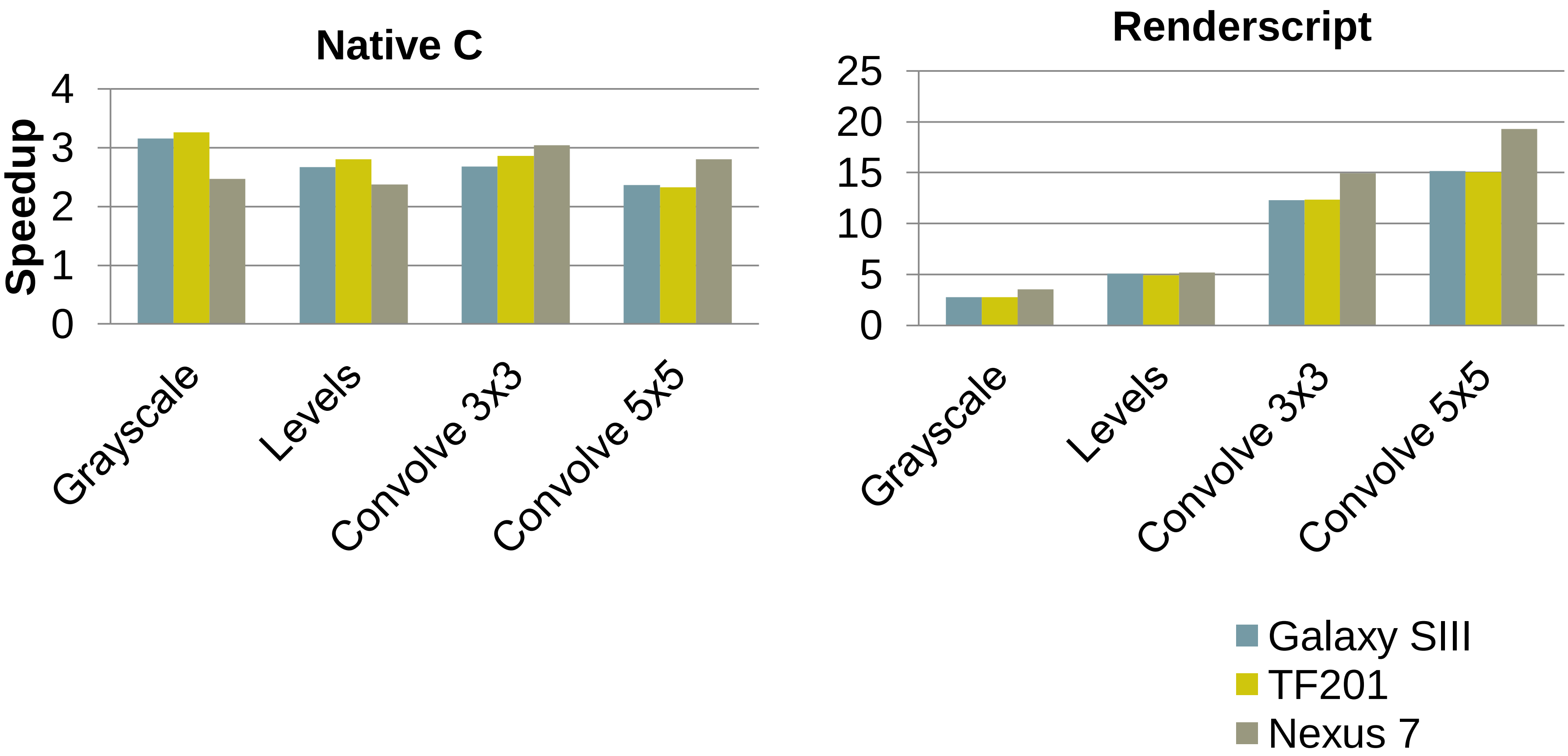
Computational Result

- Samsung Galaxy SIII
  - Exynos 4 – 4412 (1.4GHz, Quad-core)
  - GPU ARM Mali-400/MP4
  - 1 GB RAM
  - Android 4.1

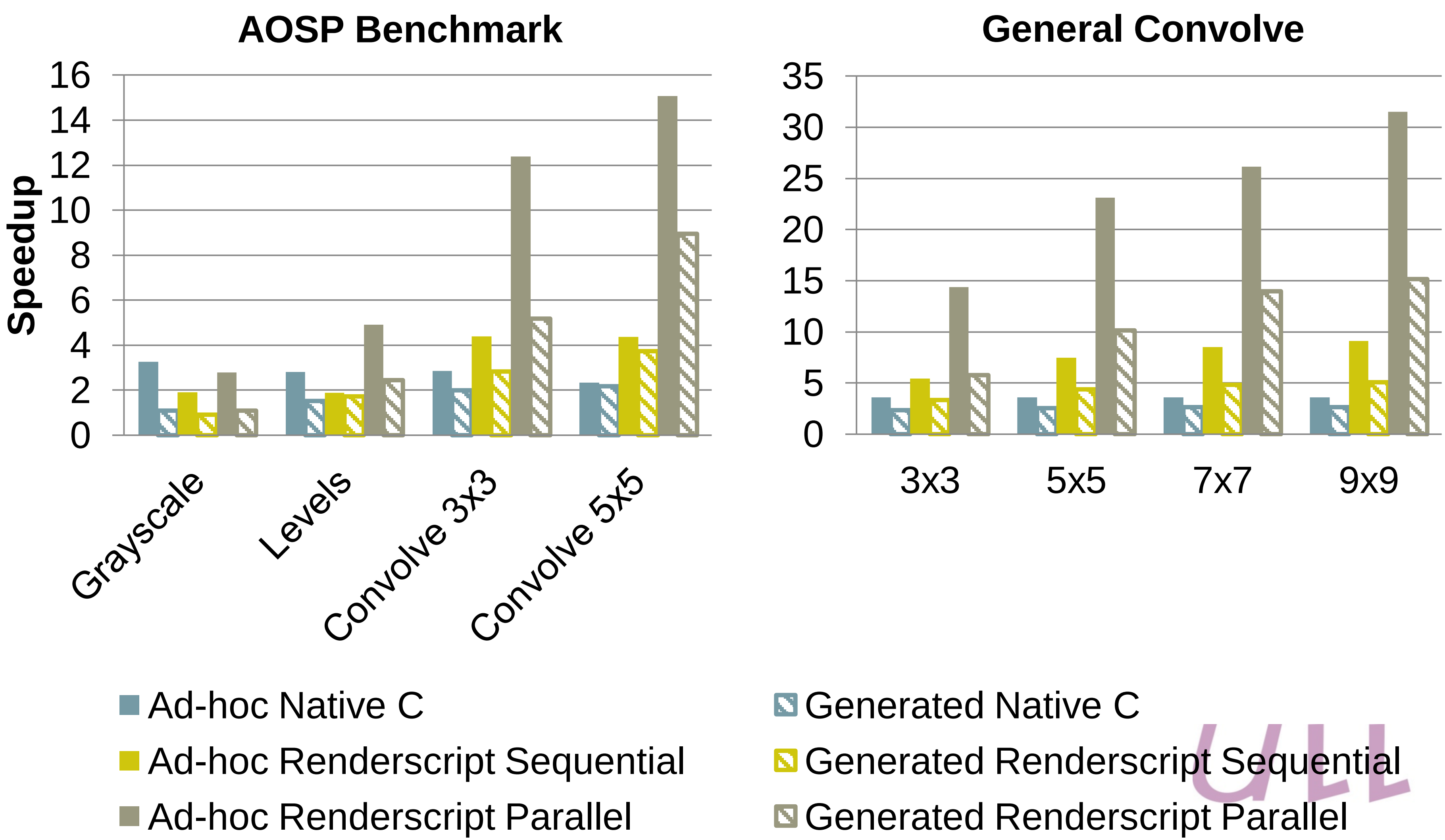
- Asus Transformer Prime TF201
  - NVIDIA Tegra 3 (1.4GHz, Quad-core)
  - GPU NVIDIA ULP GeForce.
  - 1GB RAM
  - Android 4.1

- Asus Nexus 7
  - Qualcomm Snapdragon S4 Pro, (1.5GHz, Quad-core)
  - GPU Adreno 320.
  - 2GB RAM
  - Android 4.3

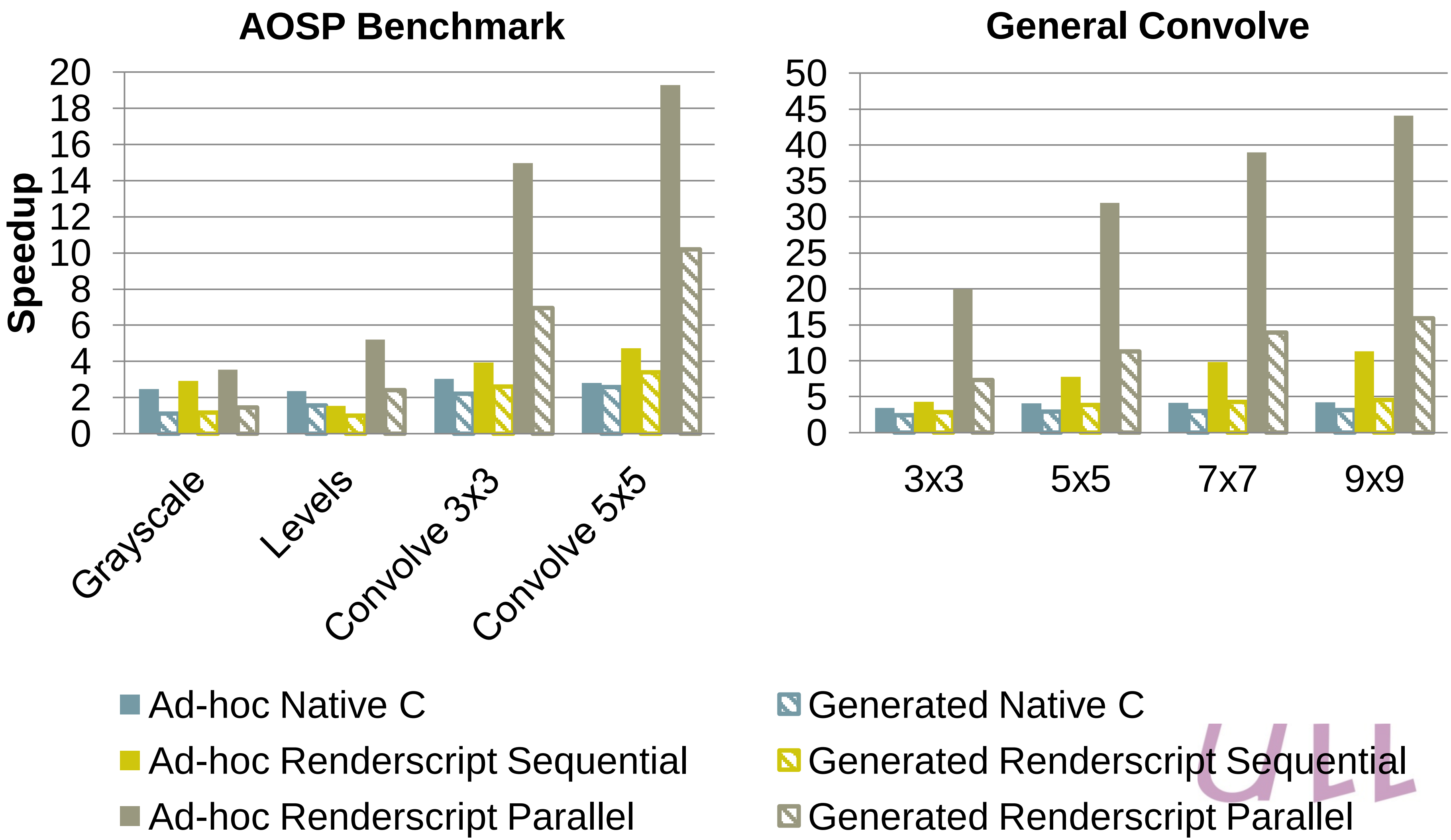
Device comparison



Speedup: Galaxy SIII – Asus TF201



Speedup: Nexus 7



Timsort

- Hybrid sorting algorithm.
  - Insertion sort.
  - Merge sort.
- Used to sort arrays.
  - Python since version 2.3.
  - Java SE 7.
  - Android.
  - GNU Octave.
- Performance
  - Best case O(n).
  - Average case O(n log n).
  - Worst case O(n log n).

Java TimSort

```
public class TimSort {
    public void sort(float[] a) {
        ...
        BinarySort binarySort = new BinarySort();
        binarySort.sort(a, loArray, runlenArray, startArray, 0);
        ...
        MergeSort merge = new MergeSort();
        merge.sort(a, loArray, runlenArray);
    }
}
```

Renderscript Binary Sort

```
@Target(RENDERSCRIPT)
public class BinarySort {

    @Parallel
    public void sort(float[] a, @Input int loArray[], @Map(TO) int runlenArray[],
                                        @Map(TO) int startArray[], @Index int i) {
        ...
        binarySort(a, lo, lo + runLen, lo + start);
    }

    @Declare
    private void binarySort(float[] a, int lo, int hi, int start) {
        ...
    }
}
```

Native Merge

```
@Target(NATIVE)
public class MergeSort {

    public void sort(float[] a, @Map(TO) int loArray[], @Map(TO) int runlenArray[]) {
        ...
        mergeCollapse();
        ...
    }

    @Declare
    private void mergeCollapse() {
        ...
    }
}
```

Timsort

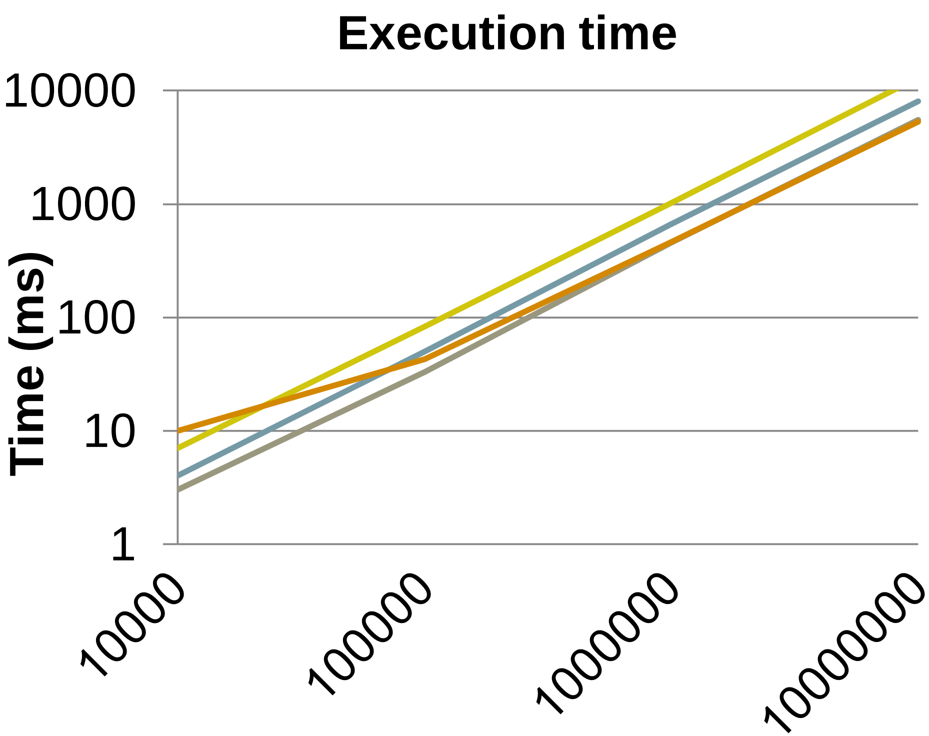
- Hybrid sorting algorithm.
  - Insertion sort.
  - Merge sort.
- Used to sort arrays.
  - Python since version 2.3.
  - Java SE 7.
  - Android.
  - GNU Octave.
- Performance
  - Best case O(n).
  - Average case O(n log n).
  - Worst case O(n log n).

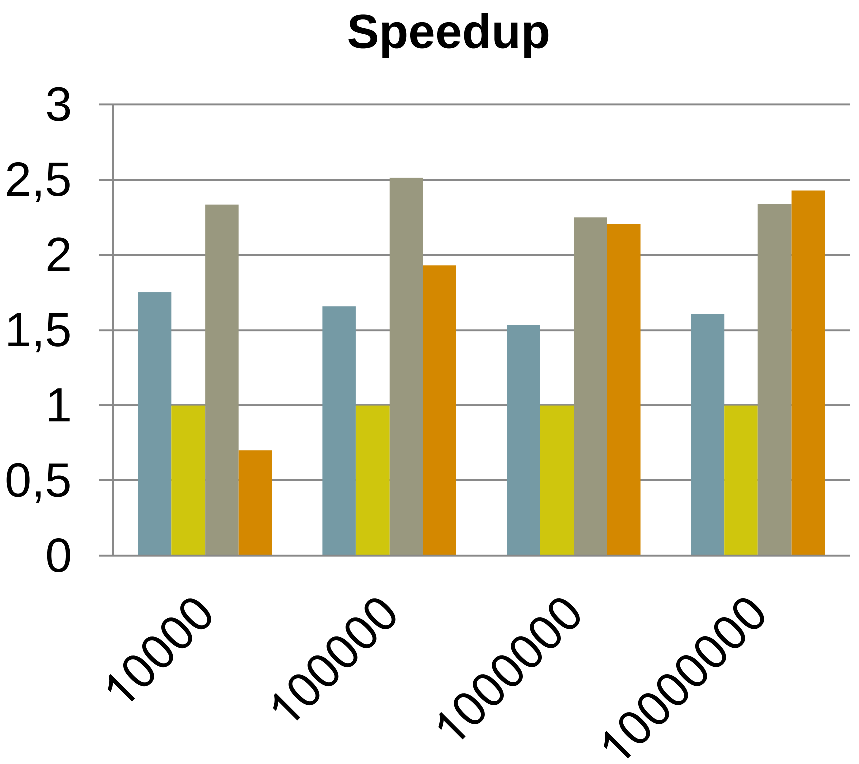
Java
Generated Native C
Generated Renderscript Sequential

Dual Pivot Quicksort

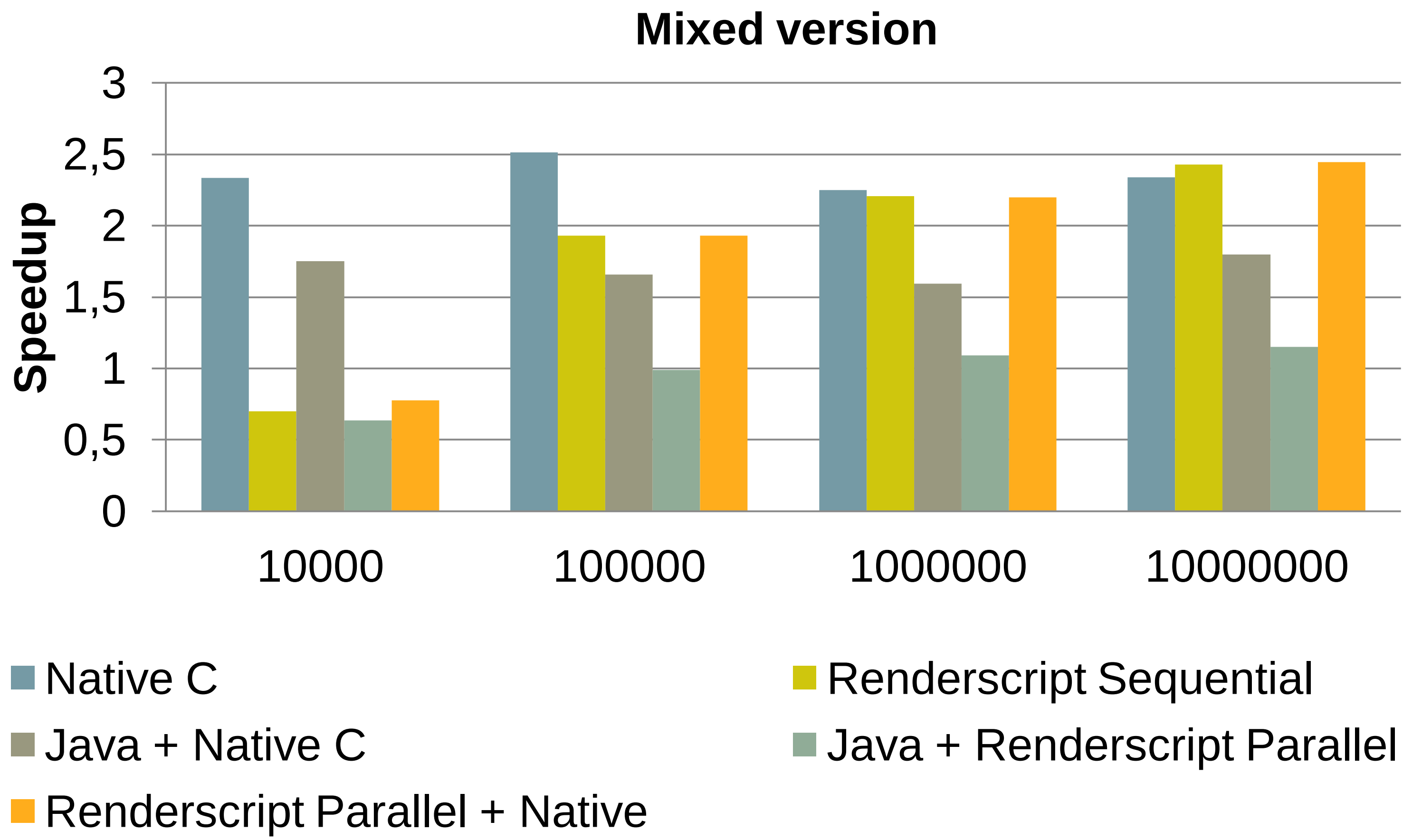
Insertion sort	Merge sort
Java	Native C
RS Parallel	Java
RS Parallel	Native C

Timsort


Execution time
Time (ms)
10000
1000
100
10
1
10000
100000
1000000
10000000


Speedup
3
2,5
2
1,5
1
0,5
0
10000
100000
1000000
10000000
Java Quicksort
Java
Native C
Renderscript Sequential

Timsort



Mixed version
Speedup
3
2,5
2
1,5
1
0,5
0
10000
100000
1000000
10000000
Native C
Renderscript Sequential
Java + Native C
Java + Renderscript Parallel
Renderscript Parallel + Native

Conclusion

- The methodology used has been validated on scientific environments.
- We proved that this methodology can be also applied to not scientific environments.
- The tool presented makes easier the development of efficient applications in Android.
- We get efficient code at a low development cost.
- The ad-hoc versions get higher performance but their implementations are more complex.

Future work

- Adding support for statement annotations.
- Adding new directives and clauses.
- To optimize memory efficiency (supported objects).
- To optimize compute efficiency (vector operation).
- To generate parallel C code.
- To generate parallel Java code.

THANKS

Alejandro Acosta
aacostad@ull.es

Francisco Almeida
falmeida@ull.es

Universidad de La Laguna

High Performance Computing Group

FEDER-TIN2011-24598