

Review: Comparative Study of Android Runtime Vs Android Native Development Kit Vs Renderscript

Harneet Kaur

Assistant Professor

School of Computer Science and Engineering
Lovely Professional University,
Phagwara, Punjab, India

Amogh Agarwal

Research Scholar

School of Computer Science and Engineering
Lovely Professional University,
Phagwara, Punjab, India

Abstract: *The Android ecosystem contains mainly three entitlement platform for the execution appropriate for different purposes. Android claims are mainly written in Java language, the computationally intense part can be flew up by forcing us to use native language or by applying the different approach of architecture originate in Graphics Processing Unit(GPUs). So in my comparison of this thesis measure the performance of Google's GPU acceleration framework and benefits of switching Java to RenderScript. This comparison paper shows the benefit of advanced image processing on larger images, the benefits are enormous enough of using RenderScript usage instead of java for upcoming smartphones. However, if the image processing is conducted on very small images(eg logos) it contains few calculations, moving to a RenderScript or Native language so it is not worth the added development static and time complexity. RenderScript is the best choice if the Graphics Processing Unit(GPU) vendors provide an optimized implementation of the image processing task and if there is no such implementation then we can use any java or RenderScript according to your choice. If we required the full precision in the floating-point arithmetic, then C++ or Java implementation is suggested. If there is any possibility to achieve the desired effect without compliance with RenderScript provides better run time performance as compared to floating Point Arithmetic standards.*

Keywords: *Android ecosystem, Graphics Processing Unit (GPU), Java, RenderScript, Floating-point arithmetic*

INTRODUCTION:

The first version of the mobile operating system Android was released in fall 2008. It is, as of January 2017, the most widely used smartphone operating system [1]. Android phones are using all over the world, with varying network and device quality. It has important for smartphone application developers to be able to develop a high-quality application that great on low-specs devices all around the countries.

Android application developers can choose to write the business logic of an application in a native language (a source language that is directly compiled to machine code) or Java, where Google recommends the use of Java [2]. However, when conducting computationally intensive tasks it can be advantageous to use native languages, as it is generally faster than Java [2], to not impede the user experience. The developer can utilize the graphics processing unit to increase the computing to utilize the full abilities of the device to the full potential. This shows that the device using a graphics processor to offload compute-intensive portions of code to use the

graphics processing unit, and leaving the remainder of the code to run in central processing unit this allows the device to advance the speed and accuracy of the smartphone. Code written for GPU doesn't have to be customized for every different type of GPU, but it can be compiled for a higher-level language. Due to this acceleration of GPU is more readily available for developers today as compared to any other language. As today's technology, we see more and more augmented and virtual reality products; it is of utmost importance to keep the experience as smooth as possible.

BACKGROUND

This session contains background information that is required to understand the rest of the paper.

Interpreted and Native languages: The difference between native and interpreted language is that the native language gets compiled to instruction that can be interpreted by the processor. An overview of their compilation processes and differences are presented in this section.

Java: Java is a general-purpose interpreted, object-oriented, class-based language developed Oracle. It is a platform-independent application which means it is written once, run anywhere(WORA). Java gets compiled to bytecode to run on Java Virtual Machine(JVM) regardless of the underlying computer architecture. The JVM takes care of translating the bytecode to instructions that the host CPU can understand. Figure2.1 below shows the process compiling Java source code to its correspondingbytecode that will be processed by the Java Virtual Machine

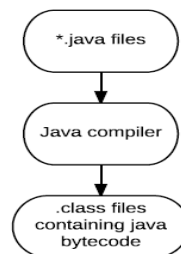


Figure 2.1:Steps in compilation to Java bytecode

C++: C++ is a general-purpose programming language as an extension of the C withy class developed by Bjarne Stroustrup. It is a native language, and skip the translation step of Java Virtual Machine, as it can directly be compiled to native processor instruction. This shows that it does not architecture-independent and code has to be compiled for each architecture as it supposed to run on. The native-language source files are compiled to assemble code by the compiler. Then the code generated by the compiler is then assembled into Object-code as shown in the figure below.

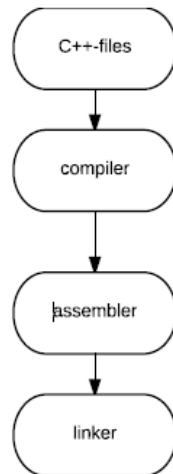


Figure 2.2: Steps in compiling native language source files to an executable

Android: Android is an operating system developed by Google, designed primarily for use on mobile smartphones and tablets. It is based on the Linux kernel. The Android operating system has also been customized to run on smartphones, TVs, and cars. It is the most widely used mobile operating system with a market share of 88% [3]. Linux Kernel at the root of the Android architecture we also have middleware and native libraries (e.g. OpenGL, Webkit) above it lies the application framework. For the development of the Android application, we use the Application Programming Interface (APIs).

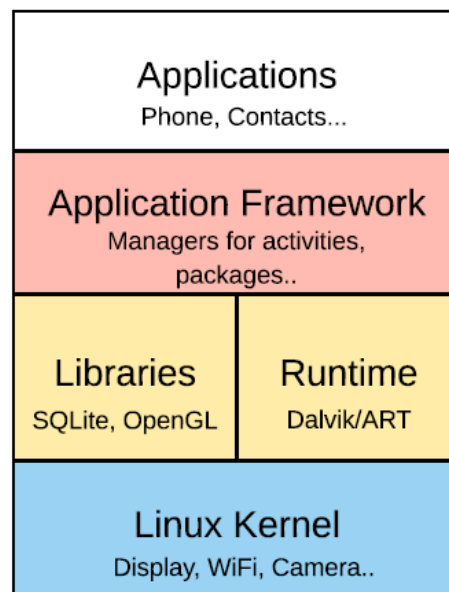


Figure 3.1: The Android software stack

As shown in the figure the applications that are found on Play Store are written in the topmost layer in the Application layer. The runtime, for running the application on modern smartphone lies in between the application framework and native libraries.

Dalvik: Dalvik is a discontinued virtual machine (VM) on Android device for running Android version 4.4v and earlier. It was written by Dan Bornstein it is completely different from Oracle's standard Java Virtual Machine (JVM). The Dalvik VM will help mobile devices with limited storage space and memory. It uses a register-based architecture, as opposed to the Java Virtual Machine (JVM) stack-based architecture and therefore requires fewer VM instructions. The uncompressed .dex-file used by the Dalvik Virtual Machine is a little smaller than as compare to Java Archive, which makes it suitable for running with limited storage space and memory on an android smartphone. It also has drawbacks of using a Virtual Machine with a JIT-compiler such as Dalvik as comparing to using the native language. The time constraints of running them incur time constrain while lowering the degree of possible optimization as compared to Static compilers. Dalvik has had trace-based just-in-time (JIT) compilation since the release of Android 2.2. The JIT compilation allows Dalvik to compile frequently executed code ("traces") to native code. Even though Dalvik interprets the remaining bytecode, this dynamic compilation provided significant performance improvements [4].

Android Runtime: Android Runtime (ART) is Dalvik's successor and is the standard runtime for Android application and certain system service. Android runtime (ART) was created specifically for the Android operating system and help in the run and optimized the application to run on the devices with a limited amount of storage space and memory. It also has numbers of features to improve the performance of the application in the android system

Android Native Development Kit: The Android Native Development Kit (NDK) is a set of tools allowing developers to write parts of their Android application in native languages such as C++ or C. It is used to achieve to run computationally or low latencies intense code. Additionally, it enables reusing previously written C++ or C code.

The NDK is used to compile C/C++ code into a native library and package it into the application package. Java code can then use the Java Native Interface (JNI) to call functions in the native library. It is worth noting that crossing this Java-Native boundary might incur performance degradation, as compared to calling a Java method [5]. Native code is platform-specific, so for the native code to work on all devices the code must be compiled for every supported device architecture e.g. x86, ARM.

RenderScript: RenderScript (RS) is an android ecosystem framework for running computationally intensive task with high performance using heterogeneous hardware and its primarily oriented for use with data-parallel computation. It allows developers to increase the performance and compatibility of their application the cost of making and writing more complex (lower-level) code which will work across the multi-core GPUs and CPUs available on a smartphone. We use GPU for the compute-intensive function which beneficial is that its architecture differs from that of CPU as it consists of cores optimized for sequential processing,

on the other hand, GPU has parallel architecture so that we can optimize multiple tasks simultaneously.

It provides the developer has three primary tools

1. Simple 3D rendering API
2. Compute Unified Device Architecture (CUDA)
3. C99

The performance gain, compared to Java, it gained from executing native code on the device. As compared to RanderScript, Android Native Development Kit(NDK) is cross-platform. The runtime of the device decides on computation should be run on the GPU or CPU.

Blurring: A pre-defined bitmap is blurred and shown using different blurring algorithms. The algorithm runs sequentially using AsyncTasks a class in the Android framework used for processing a background thread. Each algorithm takes two parameters

1. An input bitmaps
2. An output bitmap

After processing the bitmap is invalidated and is redrawn by the OS. The processing is done by maximizing the numbers of threads possible in the device in the case where it is possible to change. Then the final thread waits for every thread to complete before rendering the final image on the smartphone, it uses the join-method present in the language. The system garbage collector is manually requested to run to not pollute the run times of the algorithms after this it runs the algorithm. Note that this does not pledge that the garbage collector is raised, but it is visible in the logs when it is running. The logs were checked manually to make sure during running the process no garbage collector invokes. For some instances, the developer cannot specify the kernel when we are using RenderScript e.g. Gaussian Burning. The difference between the two images is calculated pix per pixel.

$$||x_1|| = \sum_{i=1}^n x_i$$

The Blue, Green, and Red(R, G, B) values of each pixel are summed and the sum is compared between the images. This is called Manhattan Norm. If the resulting image differs by 10% then also they are acceptable.

Gaussian Filter: Gaussian Kernel is calculated as follows:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

For increasing the speed of image processing, one can use a 1D filter and apply it again, both vertically and horizontally. This indicates that if we use 1D vector on computing and applied horizontally to row then the resulting image will use the vertical pass for all columns for the given image. The Gaussian Kennel used was pre-calculated using an online service[12].

The Gaussian filter has eight different implementations tested

- Single-threaded Java
- Multi-threaded Java
- Single-threaded C++
- Multi-threaded C++
- C++ using OpenCV
- RenderScript
- Relaxed RenderScript
- RenderScriptIntrinsics

The C++ and Java implementations were based on the reference C++ implementation found in the Android system source Code2.

Box filter: Applying a box filter can be done using the same strategy as applying the guessing filter using one horizontal pass and one vertical pass. Recall that a box filter is identical to a Gaussian filter.

The box filter had seven different implementations:

- Multi-threaded Java
- Single-threaded C++
- Multi-threaded C++
- C++, using OpenCV
- Relaxed RenderScript
- RenderScript

Google does not provide an intrinsic RenderScript box filter function and it Google does not provide an intrinsic Renserscriptvox filter

Dalvik vs ART

Konradsson [6] compared the performance of the Dalvik VM and the then newly introduced Android Runtime. The author compared run time, memory usage and application size, using well established benchmarking frameworks.

For solving a dense 2000_2000 linear equation system yielded an average improvement of 13.35% when we use Android Runtime(ART) over Dalvik when we compare numbers of floating-point operations per second. In case most of the cases of Dalvik overperform Android Runtime(ART)by performing 0.6%-0.8% more floating-point operations per second. The test was done on Android version 4.4.2v and 5.1.1, meaning the ART version has any control on JIT-compiler in case of Dalvik was able to optimize the code for architecture on the device. The author furthermore measure the RAM usage for the following application

1. Drive
2. Gmail
3. WhatsApp
4. Netflix
5. Dropbox
6. Skype

It is not clear what actions the user performs, but the average RAM usage was 45% higher on Dalvik.

Java and C++ Benchmarks: Reinholtz [7] claimed that the run-time performance of the Java programming language will likely surpass that of C++. The author based this claim on the fact that the dynamic compilation gives the Java compiler access to runtime information that is not available to the C++ compiler. The author claims that this is bound to occur since the market for embedded systems will be driven to extend battery life and that a more performant language would be desirable. Hundt [8] conducted a comparison of the programming languages Java, C++, Go and Scala. The intent was to compare loop recognition in the languages mentioned. The implementations all used idiomatic container classes but did not attempt to exploit specific language or run time features.

The outcome of the result

1. C++ implementation contains 25.6% less line of code than Java version.
2. Java version uses 6 times less Virtual Memory as compared to C++ version.
3. C++ version is 5.8 times faster than the Java version.

The author said that even though the benchmark itself was compact and simple, it utilized many languages features such as higher-level data structures (maps, lists), some well-known algorithms (eg Depth First Search, Breadth-First Search), which means that the comparison shows could be applicable in other situations as well. Following the reasoning of Reinholtz [9], this gap between the C++ and Java performance will decrease as the JIT compilers used in the Java Virtual Machine are improved. Gherardi, Brugali, and Comotti [9] showed a smaller difference in run time performance than Hundt [8]. Gherardi, Brugali, and Comotti [10] implemented algorithms processing sensor data often used in robotics.

RESULTS

Measuring Runtime Performance: There are a couple of ways to measure run time in Java. We can also use wall-clock to measure `System.currentTimeMillis()`, which is not reliable seeing as it can be altered at a seemingly random time by the OS. The elapsed CPU time is measured using the Android OS system call `SystemClock.elapsedRealtimeNanos()` as recommended by Google [11].

Image processing test: The run time of the algorithms can be changed depending on several factors. Among others, the Just-in-time (JIT) compiler present in the OS will optimize the code as it was running. To diminish the effect on the run time several warmups round are done before the run time is measured. The device should run on flight mode so that there was no interference while testing the algorithms. In the case of a blurring test, we use up to 8-10 warmup rounds so that we can get the best result.

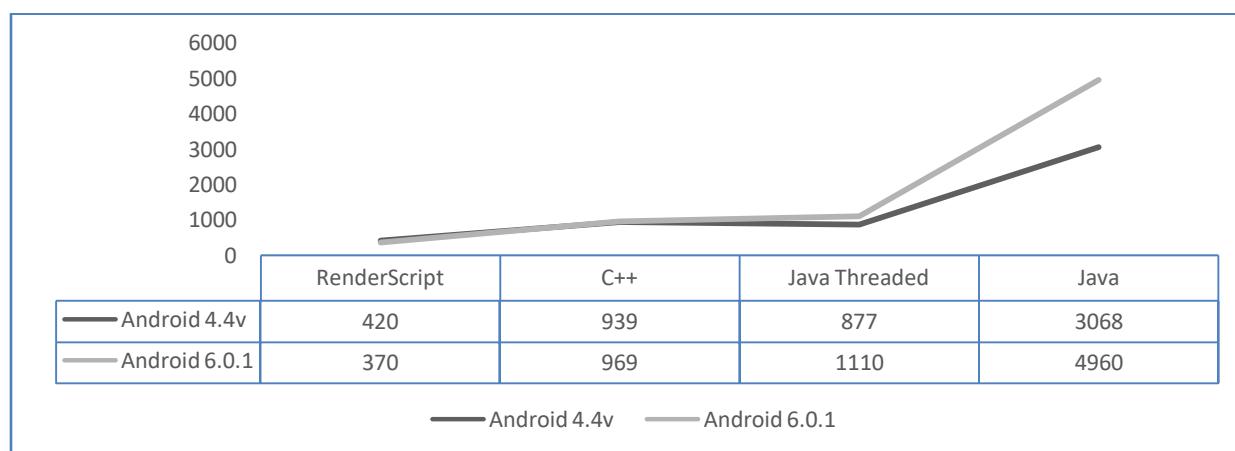
Blurring: Blurring algorithms were applied to three images of different sizes, Ranging from 100_67 p up to 1920_1080p. The algorithms were warmup by 8-10 times before calculating the result. The run time of the algorithms are presented below

Gaussian filter: The Table 1 shows the runtime of the Gaussian filter implementation on a 1920_1080p image which will run on both Android 4.4v and 6.0.1v. It could be noted that the different OS versions were used on different Smartphones while keeping the RAM and the storage space the same.

Algorithm	Android 4.4v	Android 6.0.1
Java	3068ms	4960ms
Java Threaded	877ms	1110ms
C++	939ms	969ms
RenderScript	420ms	370ms

Table 1: Run times for applying Gaussian filter to a 1920 -1080 image

This table clearly shows that the implementation of RenderScript will upsurge the implementation by a significant margin. The Java Threaded was the second-fastest implementation on Android version 4.4v and slows at android version 6.0.1v.

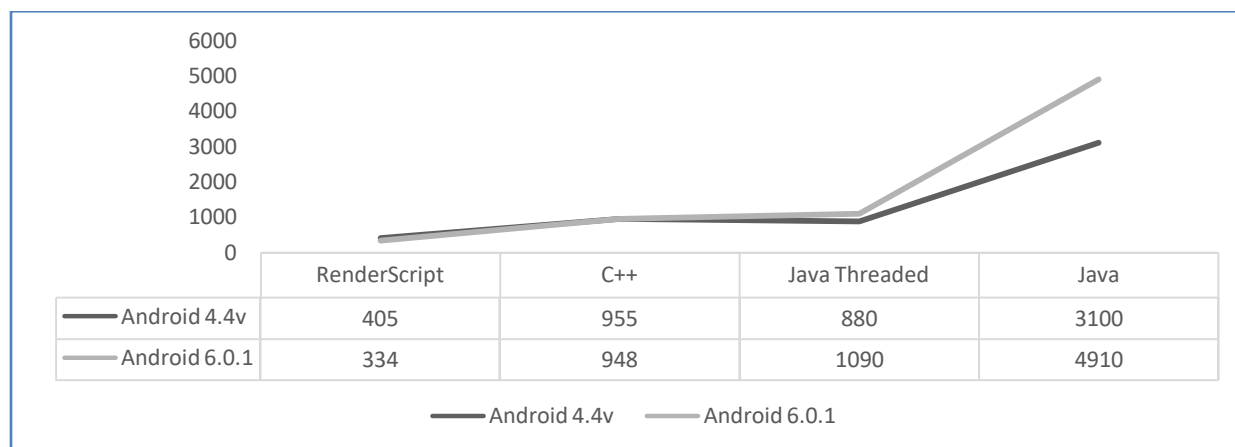


Box filter: The Table 2 shows the runtime of the Gaussian filter implementation on a 1920_1080p image which will run on both Android 4.4v and 6.0.1v. It could be noted that the different OS versions were used on different Smartphones while keeping the RAM and the storage space the same.

Algorithm	Android 4.4v	Android 6.0.1
Java	3100ms	4910ms
Java Threaded	880ms	1090ms
C++	955ms	948ms
RenderScript	405ms	334ms

Table2: Run times for applying box filter to a 1920-1080 image

This table clearly shows that the implementation of RenderScript will upsurge the implementation by a significant amount, on the other hand, Java implementation is the slowest for both the android version 4.4v and 6.0.1v.



CONCLUSION

The performance will increase in run time warrant usage of C++ or GPU acceleration frameworks over Java when writing image processing algorithms on Android smartphones. All the test showed that are implementations of C++ was showing better performance as compare to Java implementation. On the other hand, RenderScript implementations were significantly faster than Java and C++ with a greater margin. It shows that in case of bigger and large image RenderScript implementations were far superior as compare to Java but it didn't perform better in the average case. As such as Java cannot be considered a viable option while conducting advanced image processing on a larger image which will give it a greater edge for the current generation smartphone. However, the difference in the run time performance between C++ and Java is minor when the calculation is very simple or the image is very small. RenderScript implements with full floating point precision didn't show the best performance than their C++ counterpart. If compliance with Floating-point Arithmetic's is required then C++ is recommended. If there is no hard and strict requirement on floating point precision then RenderScript performance is the best performance however our result did not show a statisificant difference between the average score point. When the algorithm was applied on large image then RanderScript implementation was the best.

REFERENCES

- [1] "IDC. Smartphone OS Market Share" 2016 Q3. <http://www.idc.com/promo/smartphone-market-share/os>. Accessed on 2019-10-25
- [2] Getting Started with the NDK. <https://developer.android.com/ndk/guides/index.html>. Accessed on 2017-03-20.
- [3] Ananya Bhattacharya. Android just hit a record 88% market share of all smartphones. <https://source.android.com/devices/tech/dalvik/>. Accessed on 2017-02-07.

- [4] Bill Buzbee Ben Cheng. A JIT Compiler for Android's Dalvik VM. <http://www.android-app-developer.co.uk/android-app-development-docs/android-jit-compiler-androids-dalvik-vm.pdf>. Accessed on 2017-02-07.
- [5] N A Halli, H Charles, J FMehaut. "Performance comparison between Java and JNI for optimal implementation of computational micro-kernels", arXiv preprint arXiv, 1412.6765 (2014)
- [6] Tobias Konradsson. ART and Dalvik performance compared. 2015.
- [7] KirkReinholtz. "Java will be faster than C++". In: ACM Sigplan Notices 35.2 (2000), pp. 25–28.
- [8] Robert Hundt. "Loop recognition in C++/Java/Go/Scala". In: Proceedings of Scala Days 2011 (2011), p. 38.
- [9] BT Series," Recommendation ITU-R BT.601-5: Studio Encoding Parameters of DigitalTelevision for Standard 4:3 and wide-screen 16:9 Aspect Ratios", International Telecommunication Union,2011
- [10] L Gherardi, D Brugali, and D Comotti "A java vs. c++performance evaluation: a 3d modeling benchmark", InternationalConference on Simulation, Modeling, and Programming for AutonomousRobots, Springer. 2012, pp. 161–172.