Comparing Android Runtime with native code: Fast Fourier Transform on Android

André Danielsson

January 30, 2017

KTH Royal Institute of Technology

Master's Thesis in Computer Science

KTH Supervisor: Erik Isaksson

Bontouch Supervisor: Erik Westenius

Examiner: Olle Bälter

Abstract

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Sammanfattning

Svenska Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Preface

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

André Danielsson

Contents

Glossary

Android Mobile operating system. 1

Clang Compiler used by the NDK. 2

FFT *Fast Fourier Transform* – Algorithm that implements the Discrete Fourier Transform. 1

JNI *Java Native Interface* – Framework that helps Java interact with native code. 2

NDK *Native Development Kit* – used to write android applications in C or C++. 2

CHAPTER 1

Introduction

Summary of the chapter

1.1 Background

Android applications are written in Java to ensure portability in form of architecture independence. Applications have been run on the Dalvik virtual machine until Android version 5[1] in November of 2014[2]. Since then, Dalvik has been replaced by Android Runtime. Android Runtime, ART for short, differs from Dalvik in that it uses the Ahead-Of-Time (AOT) concept when it compiles during installation of the app. Dalvik exclusively uses a concept called Just-In-Time (JIT) compilation which means that code is compiled to native code during runtime, when needed.

1.2 Problem

To ensure that resources are spent in an efficient manner, this report has investigated whether the performance boost from writing the Fast Fourier Transform (FFT) in C++ instead of Java is significant. The following research questions were formed on the basis of the requirements:

Is there a performance difference in the implementation of an FFT in Java compared to native C++ code on an Android platform using ART?

1.3 Purpose

This report is a study that evaluates when and where there is a gain in writing a part of an Android application in C++. The purpose of this report is to educate the reader about the process of porting parts of an app to native code using the Native Development Kit (NDK) and explore the topic of performance differences between Android Runtime (ART) and native code compiled with Clang. Because ART is relatively new (Nov 2014)[2], this report would contribute with more data related to the performance between ART and native.

The result of the report can be used to value the decision of implementing a given algorithm or other solutions in native code instead of Java. The FFT is frequently used for signal processing when you want to analyse a signal in the frequency domain. It is therefore valuable to know how efficient an implementation in native code is, depending on the size of the data.

1.4 Goal

The goal of this report is to examine the efficiency of ART and how it compares to natively written code using the NDK in combination with the Java Native Interface (JNI). This report presents a study that investigates the relevance of using the NDK to produce efficient code. Further, the cost to pass through the JNI will also be a factor when analysing the code. A discussion about to what extent the simplicity of the code outweighs the performance of the code is also present. For people who are interested to know about the impacts of implementing algorithms in C++ for Android, this report might be of some use.

1.5 Method

The method used to find the relevant literature and previous studies was to search through databases using boolean expressions. By specifying synonyms and required keywords, more literature could be found. Figure 1.1 contains the expression that was used to filter out relevant articles. For each article found, the liability was assessed by looking at the amount of times it has been referenced (for articles) and if it has been through a peer-review.

(NDK OR JNI) AND
Android AND
(benchmark* OR efficien*) AND
(Java OR C OR C++) AND
(Dalvik OR Runtime OR ART)

Figure 1.1: Expression used to filter out relevant articles

1.6 Delimitations

This report does only cover a performance evaluation of the FFT algorithm and does not go into detail on other related algorithms. The decision of choosing the FFT was due to it being a common algorithm to use for signal analysis. This report will not investigate the performance differences for FFT in parallel due to the complexity of the Linux kernel used on Android. This would require more knowledge outside the scope of this report and would result in a more broad subject.

1.7 Ethics and Sustainability

An ethical aspect of this report is ...

Environmental sustainability is fulfilled in this report because there is an aspect of battery usage in different implementations of algorithms. The less number of instructions an algorithm require, the faster will the CPU lower its frequency, saving power. This will also have an influence on the user experience and can therefore have an impact on the society aspect of sustainability. If this report is used as a basis on a decision that have an economical impact, this report would fulfil the economical sustainability goal.

1.8 Outline

- **Chapter 1 - Introduction –**
- **Chapter 2 - Background –**
- **Chapter 3 - Methodology –**
- **Chapter 4 - Experiments –**
- **Chapter 5 - Discussion –**
- **Chapter 6 - Conclusion –**

Chapter 2

Background

Summarizing the chapter

2.1 Android SDK

To allow developers to build Android apps, Google developed a Standard Development Kit (SDK) to facilitate the process of writing Android applications. The Android SDK software stack is described in Figure 2.1. The Linux kernel is at the base of the stack, handling the core functionality of the device. Detecting hardware interaction, process scheduling and memory allocation are examples of services provided by the kernel. The Hardware Abstraction Layer (HAL) is an abstraction layer above the device drivers. This allows the developer to interact with hardware independent on the type of device[3].

The native libraries are low level libraries, written in C or C++, that handle functionality such as the Secure Sockets Layer (SSL) and Open GL[4]. Android Runtime (ART) features Ahead-Of-Time (AOT) compilation and Just-In-Time (JIT) compilation, garbage collection and debugging support[5]. This is where the Java code is being run and because of the debugging and garbage collection support, it is also beneficial for the developer to write applications against this layer.

The Java API Framework is the Java library you use when controlling the Android UI. It is the reusable code for managing activities, implementing data structures and designing the application. The System Application layer represents the functionality that allows

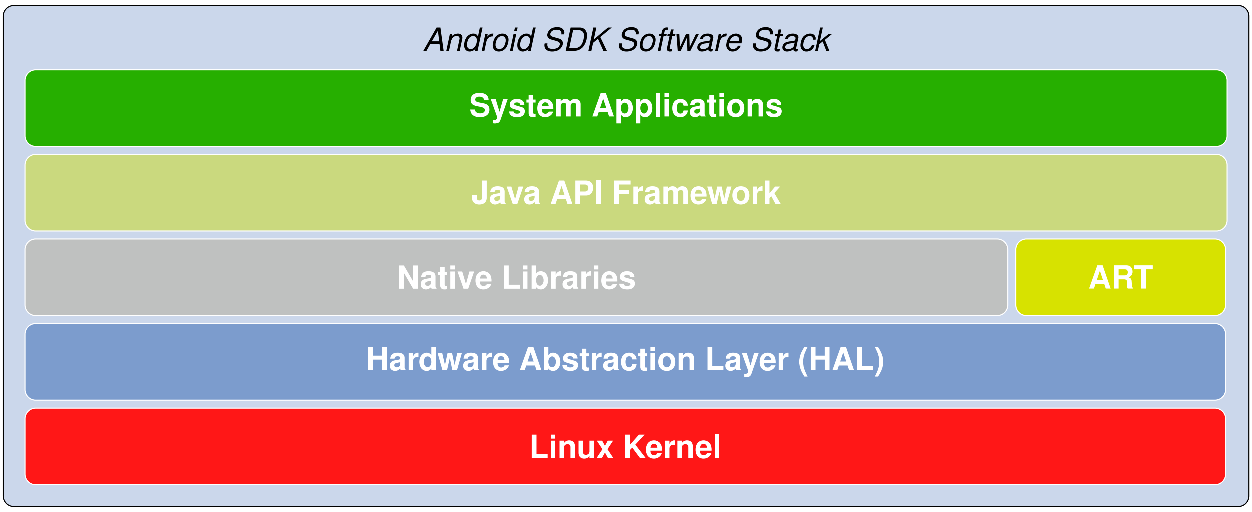

Figure 2.1: Android SDK Software Stack[5]

a third-party app to communicate with other apps. Example of usable applications are email, calendar and contacts[5].

2.2 Dalvik Virtual Machine

2.2.1 Dalvik Executables

2.3 Android Runtime

2.3.1 Installation Process

Applications for the Android operating system are mostly written in Java for easier portability between [6, p. 33]

2.4 Native Development Kit

2.4.1 Java Native Interface

2.4.2 Clang and LLVM

2.5 Discrete Fourier Transform

2.6 Gaussian Elimination

2.7 A* Pathfinding

CHAPTER 3

Methodology

Summarizing the chapter

3.1 Experiment model

3.1.1 Hardware/Testbed

3.1.2 Benchmark Environment

3.1.3 Time measurement

3.1.4 Memory measurement

3.2 Evaluation

3.2.1 Data representation

3.2.2 Data interpretation

3.2.3 Statistical significance

3.3 Fast Fourier Transform Algorithm

3.3.1 Java libraries

JTransforms[7]

3.3.2 C++ libraries

[8]

3.4 Gaussian elimination

3.4.1 Java libraries

3.4.2 C++ libraries

3.5 A*

3.5.1 Java libraries

3.5.2 C++ libraries

Chapter 4

Experiments

Summarizing the chapter

Chapter 5

Discussion

CHAPTER 6

Conclusion

Appendix A

Example appendix

Bibliography

[1] Google, "Android 5.0 Behavior Changes – Android Runtime (ART)." `https://developer.android.com/about/versions/android-5.0-changes.html`. [Accessed: 24 January 2017].

[2] Google, "android-5.0.0_r1 - platform/build - Git at Google." `https://android.googlesource.com/platform/build/+/android-5.0.0\_r2`. [Accessed: 24 January 2017].

[3] Google, "Android Interfaces and Architecture - Hardware Abstraction Layer (HAL)." `https://source.android.com/devices/index.html`. [Accessed: 30 January 2017].

[4] S. Komatineni and D. MacLean, *Pro Android 4*. Apressus Series, Apress, 2012.

[5] Google, "Platform Architecture." `https://developer.android.com/guide/platform/index.html`. [Accessed: 30 January 2017].

[6] G. Nolan, *Decompiling Android*. SpringerLink : Bücher, Apress, 2012.

[7] P. Wendykier, "JTransforms - Benchmark." `https://sites.google.com/site/piotrwendykier/software/jtransforms`. [Accessed: 30 January 2017].

[8] M. Frigo and S. G. Johnson, "The design and implementation of FFTW3," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 216–231, 2005. Special issue on "Program Generation, Optimization, and Platform Adaptation".