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

André Danielsson

February 3, 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

Chapter 1 – Introduction 1
1.1 Background . . . 1
1.2 Problem . . . 2
1.3 Purpose . . . 2
1.4 Goal . . . 3
1.5 Method . . . 3
1.6 Delimitations . . . 4
1.7 Ethics and Sustainability . . . 4
1.8 Outline . . . 4

Chapter 2 – Background 7
2.1 Android SDK . . . 7
2.2 Dalvik Virtual Machine . . . 8
2.2.1 Dalvik Executables . . . 9
2.3 Android Runtime . . . 9
2.3.1 Installation Process . . . 9
2.4 Native Development Kit . . . 9
2.4.1 Java Native Interface . . . 9
2.4.2 Clang and LLVM . . . 9
2.5 Discrete Fourier Transform . . . 9

Chapter 3 – Methodology 11
3.1 Experiment model . . . 12
3.1.1 Hardware . . . 12
3.1.2 Benchmark Environment . . . 12
3.1.3 Time measurement . . . 12
3.1.4 Memory measurement . . . 12
3.2 Evaluation . . . 12
3.2.1 Data representation . . . 12
3.2.2 Data interpretation . . . 12
3.2.3 Statistical significance . . . 12
3.3 Fast Fourier Transform Algorithms . . . 12
3.3.1 Java libraries . . . 12
3.3.2 C++ libraries . . . 12

Chapter 4 – Experiments 13

Chapter 5 – Discussion 15

Chapter 6 – Conclusion 17

Appendix A – Example appendix 19

Glossary

Android Mobile operating system. 1

FFT *Fast Fourier Transform* – Algorithm that implements the Discrete Fourier Transform. 2

JNI *Java Native Interface* – Framework that helps Java interact with native code. 3

CHAPTER 1

Introduction

This thesis explores how significant the performance differences between bytecode and native libraries are.

1.1 Background

Android is an operating system for smartphones and as of November 2016 it is the most used (CHECK MORE SOURCES?)[1]. One reason for this is because it was designed to be run on multiple different architectures [2]. Google states that they want to ensure that manufacturers and developers have an open platform to use and therefore releases Android as Open Source software [3]. Android uses the Android kernel is based on the Linux kernel with some alterations to support the hardware on mobile devices.

Android applications are mainly written in Java to ensure portability in form of architecture independence. By using a virtual machine to run a Java app, you can use the same bytecode on multiple platforms. To ensure efficiency on low resources devices, a virtual machine called Dalvik was developed. Applications on Android have been using the Dalvik virtual machine until Android version 5 [4] in November of 2014 [5]. Since then, Dalvik has been replaced by Android Runtime. Android Runtime, ART for short, differs from Dalvik in that it uses Ahead-Of-Time (AOT) compilation. This means that it compiles during the installation of the app. Dalvik, however, exclusively uses a concept called Just-In-Time (JIT) compilation, meaning that code is compiled during runtime when needed.

To allow developers to reuse libraries written in C or C++ or just to write low level

code, a tool called Native Development Kit (NDK) was released. It was first released in June 2009 [6] and has since then gotten improvements such as new build tools, compiler versions and support for additional Application Binary Interfaces (ABI). With the NDK, the developers can choose to write parts of an app in so called *native code*. This is used when wanting to do compression, graphics and other performance heavy tasks.

1.2 Problem

To ensure that resources are spent in an efficient manner, this report has investigated whether the performance boost from having the Fast Fourier Transform (FFT) compiled by the NDK instead of by ART is significant. Multiple different implementations of FFTs will be evaluated as well as the effects of the Java Native Interface (JNI), a framework for communicating between Java code and native shared libraries. The following research questions were formed on the basis of the requirements:

Is there a significant performance difference between implementations of a Fast Fourier Transform (FFT) in native code, compiled by Clang, and Dalvik bytecode, compiled by Android Runtime, on Android?

1.3 Purpose

This thesis is a study that evaluates when and where there is a gain in writing a part of an Android application in C++. One 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). Another is to explore the topic of performance differences between Android Runtime (ART) and native code compiled with Clang/LLVM. Because ART is relatively new (Nov 2014) [5], this report would contribute with more information related to the performance of ART and how it performs compared to native code compiled by the NDK.

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.

(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

1.4 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.

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*** – Introduces the reader to the project. This chapter describes why this investigation is beneficial in its field and for whom it is useful.
- ***Chapter 2 - Background*** – Provides the reader with the necessary information to understand the content of the investigation.
- ***Chapter 3 - Methodology*** – Discusses the hardware, software and methods that are the basis of the experiment. Here, the different methods of measurement are compared and the most appropriate are chosen.
- ***Chapter 4 - Experiments*** – The result of the experiments are presented here.
- ***Chapter 5 - Discussion*** – Discussion regarding the results as well as the chosen methodology.

- ***Chapter 6 - Conclusion*** – Presents what the experiments showed and future work.

CHAPTER 2

Background

The process of developing, how Android installs the app and how it runs it is explained in this chapter. Additionally, some basic knowledge of the Discrete Fourier Transform is required when discussing differences in FFT implementations.

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[7].

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[8]. Android Runtime (ART) features Ahead-Of-Time (AOT) compilation and Just-In-Time (JIT) compilation, garbage collection and debugging support[9]. 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

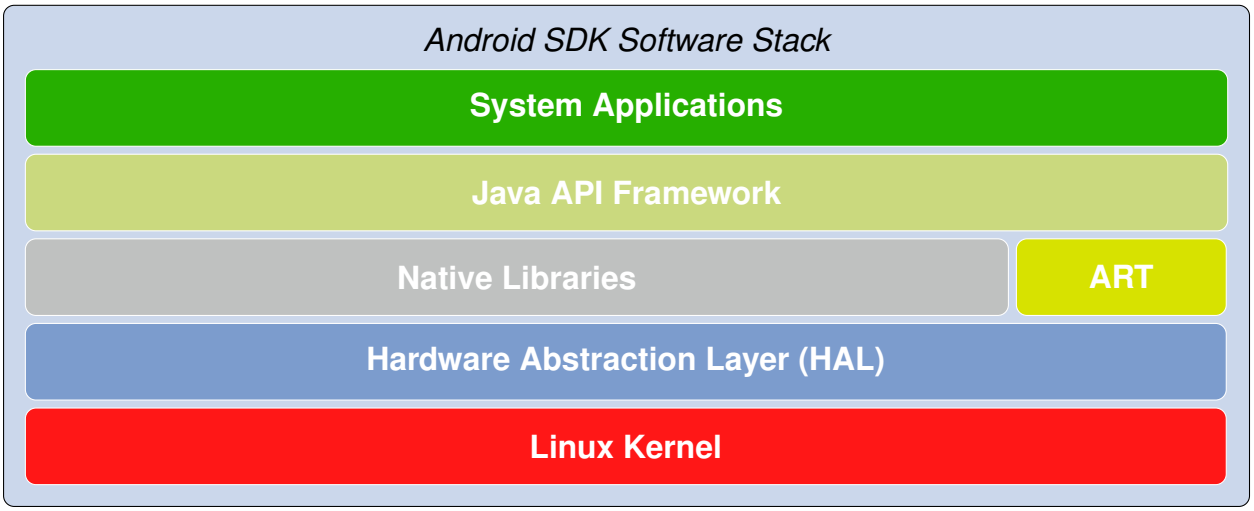


Figure 2.1: Android SDK Software Stack[9]

a third-party app to communicate with other apps. Example of usable applications are email, calendar and contacts[9].

2.2 Dalvik Virtual Machine

Compiled Java code is executed on a virtual machine called the Java Virtual Machine (JVM). The reason for this is to allow compiled code to become portable. This way, every device, independent on architecture, with a JVM installed will be able to run the same code. The Android operating system is designed to be installed on many different devices[2]. Because of the many different devices, user applications would have to be compiled for all possible platforms it should work on. For this reason, Java bytecode is a choice when wanting to distribute compiled applications.

The Dalvik Virtual Machine (DVM) is the VM initially used on Android. One difference between DVM and JVM is that the DVM uses a register-based architecture while the JVM uses a stack-based architecture. The most common virtual machine architecture is the stack-based [10, p. 158]. A stack-based architecture evaluates each expression directly on the stack and always have the last evaluated value on top of the stack. Thus, only a stack pointer is needed to find the next instruction on the stack.

Contrary to this, a register based virtual machine works more like a CPU. It uses a set of registers where it will place operands by fetching them from memory. One advantage of using register-based architecture is that fetching data between registers is faster than

fetching or storing data onto the hardware stack. The biggest disadvantage of using register-based architecture is that the compilers must be more complex than for stack-based architecture. This is because the code generators must take register management into consideration [10, p. 159-160].

The DVM is a virtual machine optimized for devices where resources are limited [11]. The main focus of the DVM is to lower memory consumption and lower the number of instructions needed to fulfil a task. Using register-based it is possible to execute more virtual machine instructions than for a stack-based architecture[12].

2.2.1 Dalvik Executables

Dalvik executables, or dex files, are the files that Dalvik bytecode is stored. They are created by converting a Java class file to the dex format. They are of a different structure than Java class files. Some differences are the header types that describes the data. One example of the differences is the string constant fields that are present in the dex-file.

2.3 Android Runtime

When an app is installed on the device, a program called **dex2oat** converts a dex-file to an executable [13].

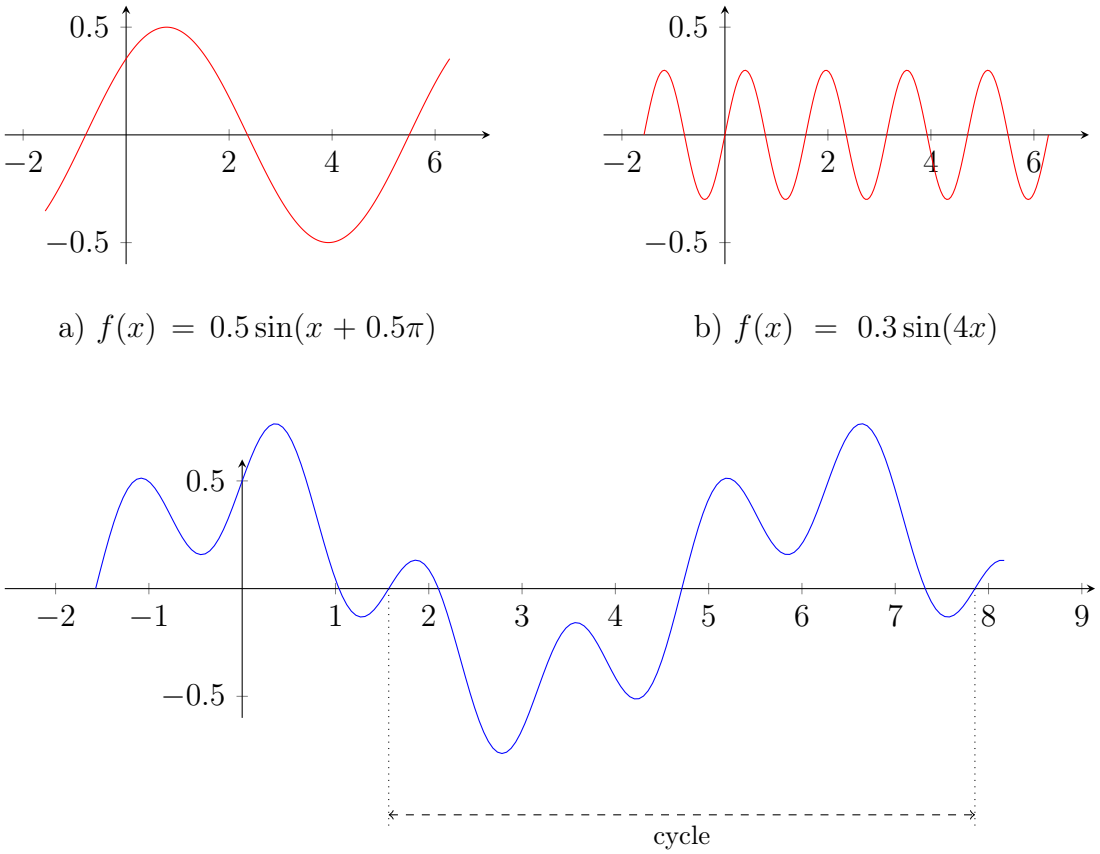
2.3.1 Installation Process

2.4 Native Development Kit

2.4.1 Java Native Interface

2.4.2 Clang and LLVM

2.5 Discrete Fourier Transform



a) $f(x) = 0.5\sin(x + 0.5\pi)$

b) $f(x) = 0.3\sin(4x)$

c) $f(x) = 0.5\sin(x + 0.5\pi) + 0.3\sin(4x)$

Figure 2.2: Sum of sinusoids

CHAPTER 3

Methodology

To ensure that the experiment is carried out correctly, many different tools for measurements was evaluated. Different implementations of the FFT are also compared to choose the ones that would typically be used in an Android project.

3.1 Experiment model

3.1.1 Hardware

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 Algorithms

3.3.1 Java libraries

JTransforms[14]

3.3.2 C++ libraries

[15]

Chapter 4

Experiments

Summarizing the chapter

CHAPTER 5

Discussion

CHAPTER 6

Conclusion

APPENDIX A

Example appendix

Bibliography

[1] IDC, "IDC: Smartphone OS Market Share 2016, 2015." http://www.idc.com/promo/smartphone-market-share/os. [Accessed: 2 February 2017].

[2] Android, "The Android Source Code." https://source.android.com/source/index.html. [Accessed: 1 February 2017].

[3] Android, "Why did we open the Android source code?." https://source.android.com/source/faqs.html. [Accessed: 2 February 2017].

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

[5] 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].

[6] C. M. Lin, J. H. Lin, C. R. Dow, and C. M. Wen, "Benchmark Dalvik and native code for Android system," *Proceedings - 2011 2nd International Conference on Innovations in Bio-Inspired Computing and Applications, IBICA 2011*, pp. 320–323, 2011.

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

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

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

[10] I. Craig, *Virtual Machines*. Springer London, 2010.

[11] D. Bornstein, "Dalvik VM internals." 2008.

[12] Y. Shi, K. Casey, M. A. Ertl, and D. Gregg, "Virtual machine showdown: Stack versus registers," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 4, no. 4, p. 2, 2008.

[13] Android, "ART and Dalvik." `http://source.android.com/devices/tech/dalvik/index.html`. [Accessed: 3 February 2017].

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

[15] 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".