Relative Heap Efficiency of the OpenCV Framework in the Dalvik Virtual Machine and the Java Virtual Machine

Joshua Lynch[†]
Department of Computer Science and Engineering
University of California, San Diego
jblynch@ucsd.edu

Abstract
The OpenCV framework is an open source project released under the BSD license that provides libraries for image manipulation and computer vision. It has been ported to numerous platforms, though for the purpose of this report only the Java and Android platforms will be referenced. This report compares the heap allocation efficiency of the two different platforms, highlighting the potential fundamental differences in the Java Virtual Machine and the Dalvik Virtual Machine.

Introduction

With the exponential growth in mobile computing's processing power, it has become possible to port third party libraries and frameworks that were previously dependent on laptop and desktop hardware. While the capacity of mobile devices has increased by leaps and bounds, memory is still a very limiting feature. Therefore, any software designed to operate on the mobile platform must take into account the physical limitations of the devices, while still maintaining approximately the same quality in output. This optimization begins with the operating system itself, by having the Dalvik Virtual Machine (DVM) being streamlined to the point of having multiple instances running simultaneously.

The comparison of the desktop centric Java Virtual Machine (JVM) and the mobile oriented DVM is an obvious choice due to the similarities between Java and Android. Java .class bytecode files can be converted to the Dalvik Executable .dex bytecode files. The .dex bytecode files are themselves optimized for memory efficiency that goes beyond the optimizations preformed by the Java compiler.

Despite the similarities in the source code, there is a major difference in the way the two runtime environments operate. The JVM uses a stack based architecture in which each method call generates its own stack frame, whereas the DVM uses the registers themselves as its data structure, eliminating the overhead involved in pushing and popping from a stack[1]. The JVM is designed to be run on any laptop and desktop computer, and so abstracts its code from the hardware at the expense of efficiency. The DVM must work within the much more limiting constraints of the mobile hardware and so is far more chip dependent. Generally speaking, these fundamental differences in the way the two environments function makes the Android code utilize far less memory than the Java code, at the expense of versatility.

This experiment will focus on the OpenCV framework, an open source project that furnishes developers with a series of libraries designed for image manipulation and computer vision. The framework has been ported to a variety of different platforms, though with each port substantial changes must be made to the libraries' source code.

Goals

The purpose of this experiment is to compare the heap efficiency of the OpenCV Imgproc library on the Java and Android platforms to see how similarity the platforms perform after the optimizations made by the conversion from .class to .dex bytecode. In order to make the unit tests as accurate representations of the "real world," they were designed to conform to the OpenCV Foundation's official guidelines on how to load their libraries.

Methodology

Sixteen different unit tests were written using both

[†] Work done Summer 2014 at the University of California, Santa Barbara's Four Eyes Lab

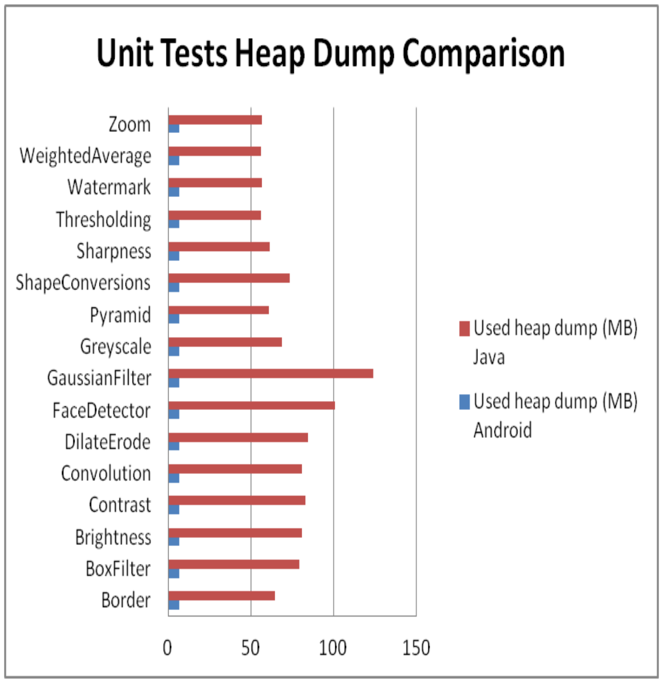
Java and Android implementations of libraries to test various methods in the Imageproc library. While some libraries such as the Java.awt library have not been directly ported and must be replaced with comparable methods from Android.graphics, most supplemental libraries are consistent across both platforms. The Imageproc library focuses on very basic image manipulation such as blurring, scaling and altering pixel brightness. The simplistic nature of its functions made the Imageproc library ideal for testing very specific aspects of the OpenCV framework. Each of the Android unit tests had two components, a viewer class and a bare-bones execution class. The viewer class was used to display the final image on the screen as proof that the unit test could successfully execute, but the overhead required to format and display the image on the screen would have tainted the heap data. The bare-bones class simply executed the unit test of the Imageproc method, thereby providing a comparable output to its Java counterpart, which simply wrote the image to disk.

The second major difference is that the Android unit tests asynchronously loaded the library from the OpenCV Manager, an app put out by the OpenCV Foundation that allows for any OpenCV dependent app to dynamically access the appropriate chip dependent version of OpenCV. This is necessary because some of the OpenCV libraries depend on C++ code compiled for specific ARM versions via Android's Native Development Kit. While this does put an additional strain on the app, it is in line with the purpose of this experiment. The OpenCV Foundation strongly recommends the use of the use of the OpenCV Manager for mobile applications, and so in a comparison of the relative effectiveness of the two platforms it is consistent to follow official best practice guidelines.

The Eclipse Memory Analyzer Tool (MAT) was the main tool used to collect and analyze the Java heap dumps. The Android heap dumps were collected with the Dalvik Debug Monitor Server (DDMS) and analyzed with the MAT. While it had no problem dealing with the Android heap dumps, the MAT consistently misrepresented the Java data by seeming to treat consecutive memory dumps as a single unit. Therefore, the reported Java heap size would grow rapidly, often doubling in size after several heap dumps. Closing the MAT was not enough to correct for this problem; Eclipse had to be completely closed and restarted before subsequent heap dumps of the same unit test would be in the ball park of one another. The MAT had none of these issues with the Android heap dumps however, and Eclipse was only closed between sessions. Each unit test was run three times.

Results

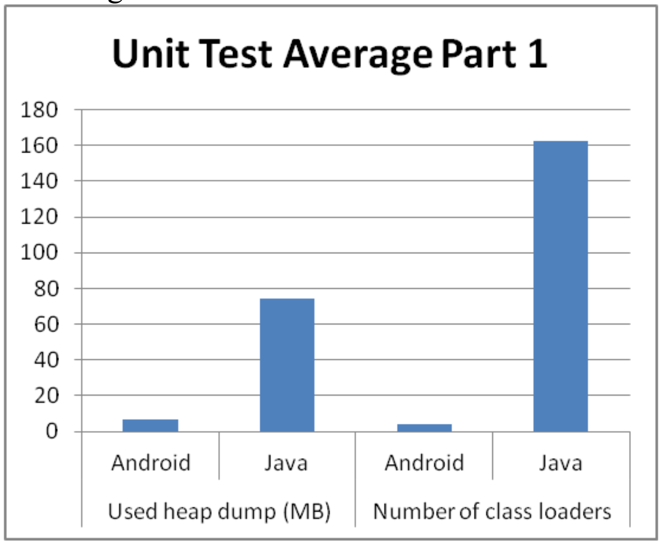
The results of the unit tests were quite striking, not only were the Android heap dumps far more uniform in size, but they were dramatically smaller that their Java counterparts. The hardware used in these tests was a Windows 64-bit laptop and a 32-bit first generation Nexus 7 running Jellybean 4.2 on the ARMv7 architecture.



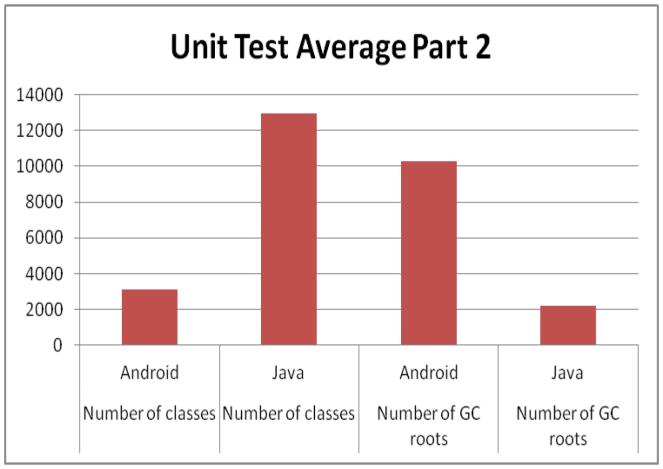
As shown by the table, the heap usage of the Java unit tests was phenomenally higher than the Android heap use. The Android tests stayed at a steady 6.7MB per run, for every single trial run across every unit test. In contrast, the Java unit tests had a low of 55.97MB and a high of 123.8MB, with fluctuations of ~10MB per trial run. Even when the Android OpenCV framework was loaded statically rather than asynchronously, the total heap usage only rose to 7.45MB.

The heap breakdown also almost

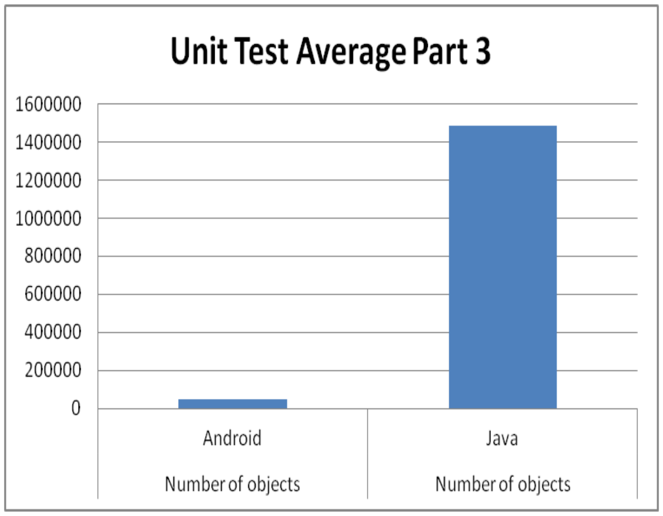
unilaterally has the Android unit tests outperforming the Java unit tests. Not only was the average Java unit test's heap allocation higher than its Android counterpart, but the number of objects instantiated by the Java unit tests was much higher.



Not only were the Java numbers higher, but they were far more erratic. The lowest average number of class loaders was 127.33 across the Thresholding unit test's three trials, with the highest average number being 205 class loaders across the three trials of GaussianFilter and FaceDetector. In contrast, every single Android unit test averaged exactly 4 class loaders across their three trials. As expected, with such a high number of class loaders in use the Java unit tests produced far more classes that their Android equivalents. Similarly there was a much higher deviation amongst the Java numbers with the number of classes ranging from ~11k to ~14.5k, while the Android tests were all at ~3.1k with a variation of about 6 classes for any given unit test.



The only category in which the Android unit tests produced more objects was Garbage Collection roots, where the usual ratio of Android to Java object creation was reversed.
The last category of heap use is the number of objects initialized. As expected, the Java unit tests initialized far more objects than the Android ones. Much like all of the other stats, the Android numbers were remarkably consistent in the number of objects created, while there are wild fluctuations in the Java numbers.



Discussion
The purpose of this experiment is to compare the memory efficiency of the OpenCV platform on its Java and Android releases. While the data seems to strongly favor the Android version of OpenCV, this may be due to environmental factors other than the work done by the OpenCV Foundation to port their libraries to Android. The platforms are different, the Java unit tests were run on a Windows 64-bit architecture while the Android

unit tests were running on a 32-bit Android Nexus 7 with Jellybean version 4.2. The VM architecture is also quite different; the DVM utilizes a register based system while the JVM uses a stack based architecture. Also, the .dex bytecode file is designed to reduce the memory used. While this helps to explain the drastic size difference between the Java and Android heaps, it does not explain why the Android heaps had far less fluctuations in size.

The OpenCV libraries were not initialized in exactly the same way across both platforms as per official recommendations; while this helps test the comparative functionalities of the platforms as a whole, it does affect the specifics of the data. This was done for two reasons; first and foremost the purpose of this experiment is to compare the two libraries' performances under “real world” conditions, which in this case involves the Java libraries being statically initialized and the Android libraries asynchronously initialized. Second, the difference itself was relatively negligible; when statically initialized the Android heap rose from 6.7MB to 7.45MB, a far cry from the lowest Java heap of 55.97MB. Finally, the unreliable nature of the MAT's heap reporting capabilities makes even the figures gathered after compensating for the MAT's quirks suspect. Platforms and tools aside, these unit tests focus exclusively on the Imageproc library and may not be representative of the OpenCV framework as a whole.

References

[1] Sinnathamby, Mark. "Stack Based vs Register Based Virtual Machine Architecture, and the Dalvik VM." *Forays in Software Development*. N.p., 15 July 2012. Web. 04 Nov. 2014.