2011 Second International Conference on Innovations in Bio-inspired Computing and Applications

Benchmark Dalvik and Native Code for Android System

Cheng-Min Lin
Department of Computer and Communication Engineering, Nan Kai University of Technology, Caotun Township, Nantou County, Taiwan (R.O.C.)
e-mail: lcm@nkut.edu.tw

Jyh-Horng Lin
Department of Electrical Engineering, Nan Kai University of Technology, Caotun Township, Nantou County, Taiwan (R.O.C.)
e-mail:jhlin@nkut.edu.tw

Chyi-Ren Dow
Department of Information Engineering and Computer Science, Feng-Chia University TaiChung City, Taiwan (R.O.C.)
e-mail: crdow@fcu.edu.tw

Chang-Ming Wen
Graduate Institute of Information Engineering and Computer Science, Feng-Chia University, TaiChung City, Taiwan (R.O.C.)
e-mail: m9740977@fcu.edu.tw

***Abstract*—Google's Android Native Development Kit (NDK) is a toolset that lets you embed components to use of native code in your Android applications. It makes possible for developers to easily compile in C/C++ for the Android development platform. Generally, developer does not concern how effective between native code and Dalvik Java code that will causes poor performance of Android. There are some researches discussed about benchmark Java and C/C++. But they do not consider the issues of Dalvik and native code for Android programming or evaluate them in real Android device. In this work, we use a more complete approach to benchmark Dalvik java code and Native code on real Android device. We conducted 12 test programs to analyze the performance and found that native code faster than Dalvik Java code for about 34.2%.**

Keywords-Component; Android; NDK; Benchmark; Native code ; Dalvik Java code;

I. INTRODUCTION

Google Android is an operating system which includes middleware, key applications, and software stack for mobile devices [6]. The first Google's Android phone T-Mobile G1 was announced at Guastavino's in New York City unveiled September 23 2008. Over the past years, the market of Google Android continues growing rapidly. Today, many devices install the Android system as the operating system.

To developing Android program, the Google Inc provides two kits for the Android development platform:

- The Android Software Development Kit (SDK) provides necessary tools and libraries for programmers to begin developing applications that run on Android-powered devices [3]. The first Android SDK (m3-rc20a) was released on 12 December 2007. Now, the latest version is SDK Tools, Revision 12 was released on July 2011. This version supports Android 2.3.4 and 3.2 Platform developments.
- The Android Native Development Kit (NDK) is a toolset that lets programmers to embed components that make use of native code in Android applications [7]. The first Android NDK (Android 1.5 NDK, Release 1) was released on June 2009. Now, the latest version of Android NDK, Revision 6 was released on July 2011. This version supports ARM and X86 platforms.

Generally, developer does not concern how effective between native code and Dalvik Java code that will causes poor performance of Android.

In additional, there are some researches [9-11] discussed about benchmark Java and C/C++. But they do not consider the issues of Dalvik and native code for Android programming or evaluate them in real Android device [1]. In this work, we use a more complete approach to benchmark Dalvik java code and Native code on real Android device. We refer to the benchmark programs of predecessors [12] and port them to the Android platform.

The reset of this paper is organized as follows. In Section 2, we discuss related works. While in Section 3, we describe the contents of the comparison benchmarks. Section 4 demonstrates the experimental results. Lastly, we conclude this paper in Section 5.

II. RELATED WORKS

A few related studies indicating performance analysis have been made. Keith's web [12] evaluates 14 kinds of algorithms in Java and C + + performance comparison. The work shows that Java has better performance than C++. But the performance analysis for native code and Dalvik Java code in Android platform was not evaluated. However, we refer the author's programs and port them to the Android platform for comparing native code and Dalvik Java code.

L. Batyuk et al. [1] benchmarks Native code on Android emulator and Linux x86 PC. The work shows that native applications can be up to 30 times as fast as an identical

978-0-7695-4606-3/11 $26.00 © 2011 IEEE
DOI 10.1109/IBICA.2011.85

algorithm running in Dalvik VM and java applications can speed-up to 10 times if utilizing JNI. But there are too few experimental methods evaluated and experiments were down on an X86 PC simulator which was not a real Android device.

S. Lee and J.W. Jeon [2] proposed five kinds of the same algorithms to evaluate the performance on the Android platform. Although that the work has mentioned the Java Native Interface (JNI) communication delay and implementation in the Android platform. But the authors proposed four methods did not completely compare performance between C and Java.

III. IMPLEMENTATION

A. Android NDK and JNI

Applications use the JNI can incorporate native code written in programming languages such as C and C++, as well as code written in the Java programming language [13]. So that we compare native code and Dalvik Java code through JNI methods.

The Android NDK is a companion tool for the Android SDK that lets programmers builds performance-critical portions of the apps in native code [7].

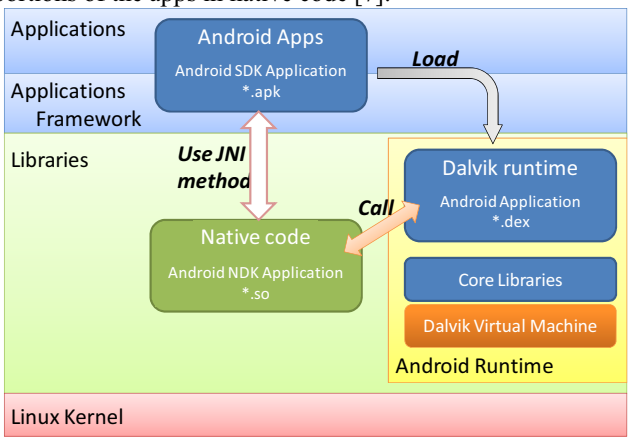

Figure 1. Android NDK Overview.

Figure 1 shows the Android NDK overview. JNI is an interface that Android uses it to access codes that implemented in other language like C or C++. The Android NDK is an optional extension for the Android SDK that supports developers to use of C or C++ code in Android applications.

B. The JNI OnLoad() function implementation

Using C++ language in NDK will cause "No JNI_OnLoad found in ..." error and then the program crash. We took some time to resolve from the JNI Examples for Android [3] to find a solution. We implement "JNI_OnLoad()" function in the native code and register the native functions and create an array of "JNINativeMethod" structures which contains function names, signatures and pointers to the implementing functions.

C. Method

In this work, we conducted the following eight kinds of test methods. Table I illustrates the programming category and test program name:

TABLE I. PROGRAMMING CATEGORIES

Method	*Test Program Name*
Numeric calculation with recursion	Ackermann, Fibonacci
Library facilities	Hash, Hash2
Usermade data structure	Heapsort, Matrix
Polymorphism	Mthcall, Objinst
Nested loops	Nestedloops
Random number generation	Random
Sieve of Eratosthenes	Sieve
String operations	Strcat

- Numeric calculation with recursion: This recursive test is used Ackermann and Fibonacci; they test the performance of making recursive function calls and also stack memory management speed. Ackermann equation was shown in (1) and Fibonacci equation was shown in (2).

$$A(m,n) = \begin{cases} n+1 & \text{if } m = 0 \\ A(m-1,1) & \text{if } m > 0 \text{ and } n = 0 \\ A(m-1, A(m,n-1)) & \text{if } m > 0 \text{ and } n > 0 \end{cases} \quad (1)$$

$$F(n) = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ F(n-1) + F(n-2) & n > 1 \end{cases} \quad (2)$$

- Library facilities: Using Hash and Hash2 test the performance of the library implementation of hash maps. On Java, the standard library container Hash Map is used, while C++ use gcc non-standard hash_map container.
- Usermade data structure: This test uses Heapsort and Matrix and do not rely on the library that rather implements their own data structures and performance operations.
- Polymorphism: This test uses Methcall and Objinst test to see how long it takes to make a call on a pointer on which the program needs to resolve the actual embed type runtime.
- Nested loops: This test consists of six nested loops, each of which runs N rounds.
- Random number generation: This part of the test uses pure numeric calculation with just a few variables and no data structures.
- Sieve of Eratosthenes: This is a simple algorithm for finding prime numbers efficiently. An array of Boolean values is used, with each element representing whether its index value can be prime or not.
- String operations: The strcat test adding a string to the end of another overhead.

We rewrite above test programs with Java and C++ language and port them to the Android platform. Java uses the SDK to compile and using the NDK compile C++ program.

IV. EXPERIMENTAL RESULTS

A. Testing environment

We use Android phone as the testing device. Table II shows the specification of the equipment and Table III is our development environment.

TABLE II. HTC DESIRE A8181 SPECIFICATION

CPU	Qualcomm 8250, 1GHz
RAM	576MB
ROM	512MB
Android Version	2.2 Froyo

TABLE III. DEVELOPMENT ENVIRONMENT

Operating System	Microsoft Windown XP SP3
IDE	Eclipse 3.5.2
SUN Java Version	1.6.0_26
Android SDK	Revision 12 .
Android NDK	Revision 6.
GCC/G++ Version	Cygwin / 4.3.4
Android Emulator	Android 2.1-update1 / API 7

Typically, Default heap size of Android phone is 24MB. To avoid out memory error caused by the program, we adjust the heap size to 64MB.

Figure 2 is one of the program screenshot that it shows the performance of the Nestedloop test program.

Figure 2. Mestedloop program screenshot.

B. Results

Figure 4 shows the experimental results. The x-axis of the graph represents input parameter. The y-axis of the graph represents execution time in milliseconds. The blue line of the graph represents Dalvik Java code and red line of the graph represents native code.

According to the results, most experiment using the native code is faster than using the same algorithm running on Dalvik Java code. Throughout of the recursive experiment Ackermann in Dalvik Java code performance is very bad. When we set the input parameter to 6, it produces a "StackOverflowError" error and then the program crash. Another result from the entire experiment, we found that Hash, Heapsort and Random testing of Dalvik Java code was faster than native code. In the native code, Nested loops has best performance.

In the separate analysis, we set input parameter to let the entire program's execution time to 3 seconds. Figure 3 shows the results of all evaluation results.

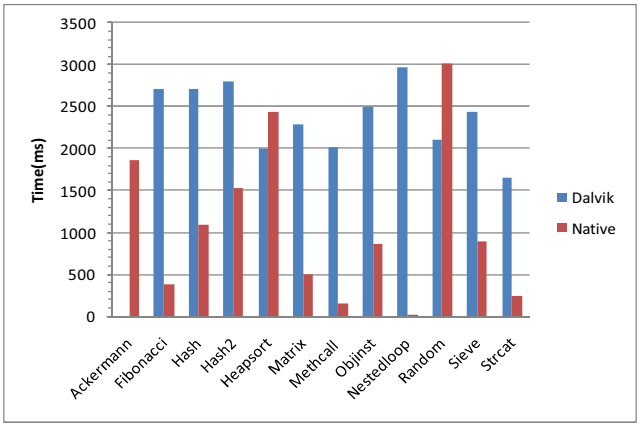

Figure 3. The program's execution time is about 3 seconds.

Table 4 shows the detail experimental results of Figure 3. We found that native code faster than Dalvik Java code about 34.20%.

TABLE IV. DALVIK JAVA CODE WITH NATIVE CODE COMPARED

Test program	***Input***	***Dalvik***	***Native***	***Compare***
Ackermann	14	Program Crash	202065	- -
Fibonacci	35	540439	72648	86.56%
Hash	290000	11873	30679	-158.39%
Hash2	10000	234268	147226	37.15%
Heapsort	1000000	4378	5543	-26.61%
Matrix	100000	115306	24703	78.58%
Methcall	1E+09	411370	30276	92.64%
Objinst	1E+08	487266	172997	64.50%
Nestedloop	35	35406	7	99.98%
Random	1E+07	5126	7521	-46.72%
Sieve	100000	61035	22244	63.56%
Strcat	1400000	1657	250	84.91%
Average				34.20%

a. Dalvik and Native execution time in milliseconds

V. CONCLUSIONS AND FUTUREWORK

In this work, we benchmark Dalvik java code and native code with Android phone HTC Desire. We rewrite 12 programs and port them to the Android platform for comparing native code and Dalvik Java code.

We evaluate different algorithms in Dalvik Java code and native code. According to the results, we found that there are three tests that the Dalvik Java code faster than native code. Also, we found that a one item that Dalvik Java code performance is very bad. In addition, the overall performance assessment of native code is faster than Dalvik Java code about 34.2%. This will help software designers to create efficient Android applications.

In our future work, we will continue analysis of dual-core mobile CPU performance and battery power consumption comparison in Dalvik Java code and native code that help software designers to create efficient Android applications.

ACKNOWLEDGMENT

The authors would like to thank the National Science Council of the Republic of China, Taiwan for financially supporting this research under Contract No. NSC 100-2221-E-252-011.

REFERENCES

[1] L. Batyuk, A.D. Schmidt, H.G. Schmidt, A Camtepe and S Albayrak, "Developing and Benchmarking Native Linux Applications on Android" Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, Volume 7, pp. 381-392, 2009.

[2] S. Lee and J.W. Jeon, "Evaluating Performance of Android Platform Using Native C for Embedded Systems," 2010 International Conference on Control Automation and Systems (ICCAS), , pp. 1160 – 1163, 2010.

[3] J. Smakov, "JNI Examples for Android," JNIExample.nw, 2009.

[4] Android Developers,"Download the Android NDK," Available : http://developer.android.com/sdk/ndk/index.html.

[5] Android Developers, "SDK Archives," Available: http://developer.android.com/sdk/older_releases.html.

[6] Android Developers, "What is Android?," Available: http://developer.android.com/guide/basics/what-is-android.html

[7] Android Developers, "What is the NDK?," Available: http://developer.android.com/sdk/ndk/overview.html.

[8] Google code, "Google Projects for Android," Available: http://code.google.com/intl/en/android/Android

[9] Przemysław Bruski, "The Java (not really) Faster than C++ Benchmark," Available: http://bruscy.republika.pl/pages/przemek/java_not_really_faster_than_cpp.html.

[10] Christian Felde, "C++ vs Java performance; It's a tie!," Available: http://blog.cfelde.com/2010/06/c-vs-java-performance.

[11] Lasse Kärkkäinen, "Yet Another Java vs. C++ Shootout,," Available: http://zi.fi/shootout.

[12] Keith, "The Java is Faster than C++ and C++ Sucks Unbiased Benchmark," Available: http://keithlea.com/javabench.

[13] S. Liang, "The Java™ Native Interface Programmer's Guide and Specification," Sun Microsystems, Inc 1999.

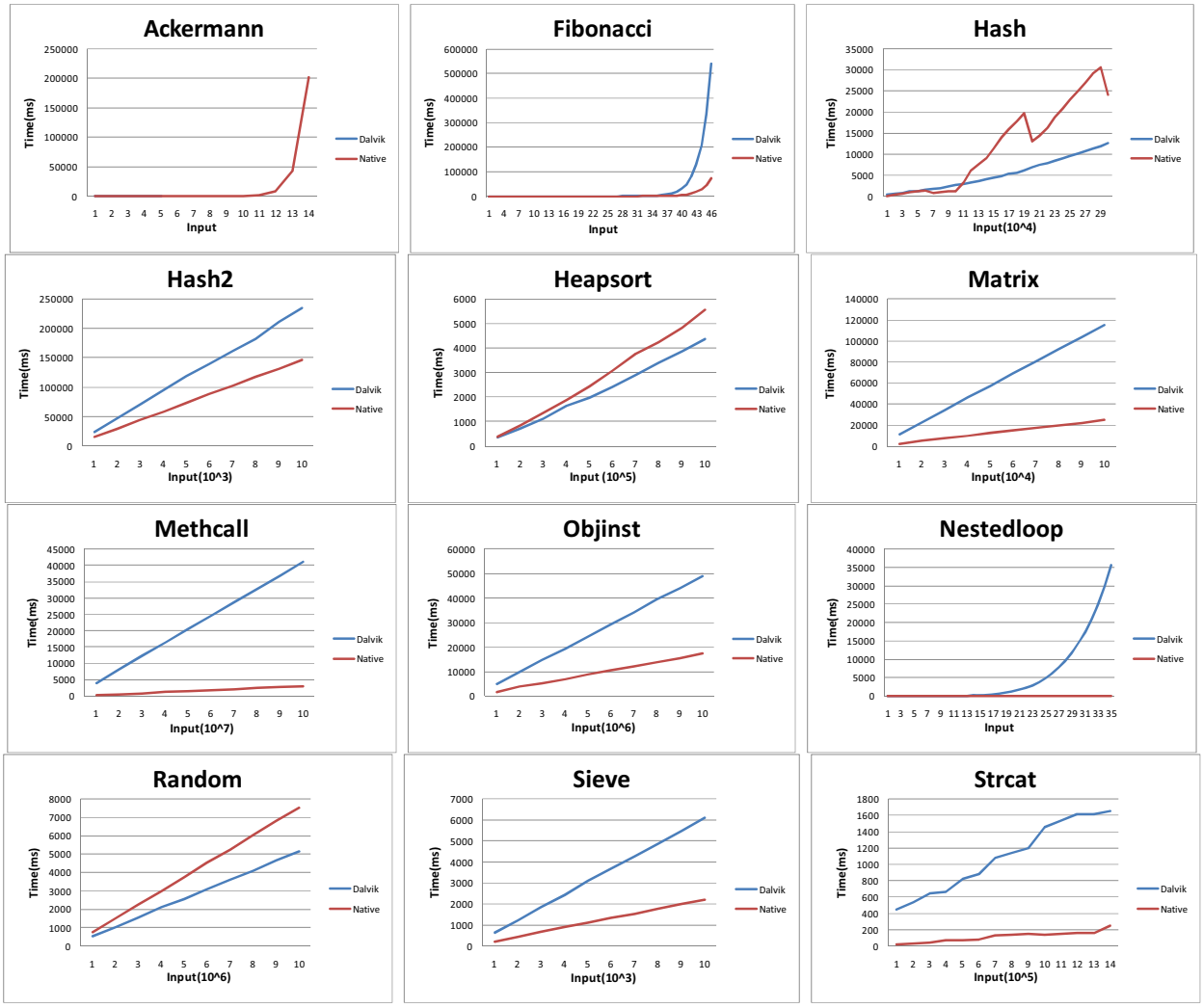

Figure 4. Experimental results