Performance Comparison of Dalvik and ART on Different Android-Based Mobile Devices

Ahmad Fikri, Alfan Presekal, Ruki Harwahyu, Riri Fitri Sari
Computer Engineering, Departement of Electrical Engineering
University of Indonesia, Depok, Indonesia
rirri@ui.ac.id

Abstract—This work mainly compares Android Runtime (ART) and Dalvik in the Android-based app. To compare we create native-based app and test in four version of Android: 4.1 (Jelly Bean), 4.2 (Jelly Bean), 4.4 (Kit Kat), 6.0 (Marshmallow) and 7.0 (Nougat). The test conducted by running three different algorithms which are running on the native-based platform. The algorithms are Quick Sort, Float Calculation, and Prime Number. The experiment show that native C is better when used in the ART with 59% faster performance compared to Dalvik.

Keywords—Dalvik, ART, Native, Android, Performance, Algorithm, Difference.

I. INTRODUCTION

First Android smart phone was launched in October 2008. Originally Android was a software created by Andy Rubin, Rich Miner, Nick Sears and Chris White in 2003. At the beginning Android was created as a software to run on digital camera, then it starts going to the direction Android as a mobile phone operating system. Android competing with other mobile operating system such as Symbian and Microsoft Windows Mobile. Google acquisition to Android in 2005 made significance milestone toward popularity of Android in recent years.

Android has become the largest operating system worldwide. In 2014 Android has dominated 78% of global market [1]. Currently 22 companies have been using Android as the operating system for their devices. Android devices which classified as low-end devices in common has same problems. They are supported with small capacity of RAM and limited processor speed. To solve those problem, it required to improve the specification of hardware and increase the cost of the component.

Dalvik was the original mechanism to run application on Android. Recently ART was developed and expected to give better performance compared to Dalvik. This work aims to compare performance of ART and Dalvik.

II. BACKGROUND

A. Runtime Application

Runtime is a set of instruction used to translated application code into code which can be understood by the machine. In the runtime process virtual machine will be involved. The early version of Android using Dalvik as its virtual machine. All the access to the Android machine provided by Dalvik virtual machine.

Android Runtime (ART) which recently developed, was introduced in October 2013. ART implemented as default runtime in the Android since version of Jelly Bean (4.2). Since Android Kit Kat (4.4) ART has not been become the default runtime for Android. ART can be selected in the developer mode of Android operating system. In November 2014 Android Lollipop (5.0) using ART as the default runtime. Since those version of Android to recent version, ART used as the default runtime [2].

B. Characteristic Dalvik and ART

Dalvik:
- Implement Just-In-Time (JIT) [3] [5] approach, which results in the least amount of memory space used but the application waiting time is longer.
- The cache does its own additions over time, so boot time is faster.
- Works better for low internal memory, as more space is available.
- Time generated stable, is a virtual machine suitable for developers.
ART:
- Using the *Ahead-Of-Time* (AOT) approach, which compiles applications when they are installed, results in faster waiting times and less memory on the processor used [4].
- A cache is created during initial boot, but reviving the device takes longer.
- Eating more internal memory since ART does optimization on APK.
- An experimental and relatively new runtime, not all developers yet.

> Figure 2. JIT vs AOT [11]

### III. EXPERIMENT DESIGN

To perform test, this work design Native-based Application using Android Studio which running several algorithms. The application will be run on Android with following requirements:

a. Device run on Android OS version 5 or above  
b. Test devices do not require same specification.  
c. Test focused on performance comparison between Dalvik and ART [6].  
d. Android API version 15 up to 25 required to run HelloJni application.

To compare the performance there are three algorithms to be tested, they are: prime number calculation, floating point calculation, and recursion (quick sort).

a. Prime number calculation

The purpose of Prime Number Calculation testing is to implement an algorithm that performs calculations with integers, thus showing the possibility of performance comparisons between Native and Java implementations on the basic operations of integers. The calculation of numbers starts at zero and ends when the maximum number has been reached [7].

```java
public long primeNumber(int max) {
    int i, j;
    int nr = 0;
    long starts = System.currentTimeMillis();
    for (i = 2; nr < max; i++) {
        for (j = 2; j <= i / 2; j++) {
            if ((i % j) == 0) break;
        }
        if (j > i / 2) nr++;
    }
    long ends = System.currentTimeMillis();
    return (ends - starts);
}
```

Figure 3. Prime Calculation Number Algorithm

Figure 3 explains that the calculation of prime numbers this time using the method of Eratosthenes, Although Eratosthenes method is not the exact same approach to check whether there is a prime number, but this can give us a list of prime numbers easily. To delete a non-prime number, it starts with 2 and the code will remove each item from the second divisible list [7]. Then the code will check the rest of the list, as shown in the Figure 4 below.

>`Figure 4. Eratosthenes Method [7]`

b. Floating point

A floating point as shows in Figure 5 is a standard type of calculation, typically used for non-integer representation. Floating point calculations are usually performed by hardware units called Floating Point Units (FPUs), but some processors lack the unit and eventually use the software. Software-based calculations are usually much slower than calculations using hardware.

The purpose of the implementation is to perform basic operations (such as addition, subtraction, multiplication and division) on floating point and measure possible performance differences.

```java
private long floatCalculation(int max) {
    long start = System.nanoTime();
    double temp = 1.5;
    double fl = 6.691842;
```
// multiplication
for (int i = 0; i < max; fl++, i++) {
    temp = temp * fl;
}

// division
for (int i = 0; i < max; fl++, i++) {
    temp = temp / fl;
}

// addition
for (int i = 0; i < max; fl++, i++) {
    temp = temp + fl;
}

// substraction
for (int i = 0; i < max; fl++, i++) {
    temp = temp - fl;
}

return System.nanoTime() - start;

private void exchange(int i, int j) {
    int temp = numbers[i];
    numbers[i] = numbers[j];
    numbers[j] = temp;
}

Figure 5: Float Calculation Code

c. Recursion
Recursion is a method of finding solutions to a problem in solving the problem itself. Recursion produces many method calls, this algorithm is one of interesting calculations to be used as a benchmark. The algorithm is also interesting because recursion is common and is used in general computer science.

Quick sort is a general calculation, in which Quick sort is a recursive algorithm that performs quick calculations. Then, the algorithm recursively calls itself with both a smaller list of elements and a larger list of elements until the list contains no more than one element. When all base cases have been reached, then the algorithm is complete.

public long sort(int[] values) {
    long startTime = System.nanoTime();
    if (values == null || values.length == 0) return 0;
    this.numbers = values;
    quicksort(0, values.length - 1);
    long endTime = System.nanoTime();
    return endTime - startTime;
}

private void quicksort(int low, int high) {
    int i = low, j = high;
    int pivot = numbers[low + (high - low) / 2];
    while (i <= j) {
        while (numbers[i] < pivot) {
            i++;
        }
        while (numbers[j] > pivot) {
            j--;
        }
        if (i <= j) {
            exchange(i, j);
            i++;
            j--;
        }
    }
}

Figure 6: Quick Sort Code

IV. RESULT AND ANALYSIS

This chapter explain data and analysis from ART and Dalvik Comparison in Java Native Interface (JNI) based applications. Experiment conducted by running HelloJNI applications several times on four different android versions, Dalvik Jelly Bean (4.1), Dalvik Jelly Bean (4.2), Dalvik KitKat (4.4), ART KitKat (4.4), ART Marshmallow (6.0) and ART Nougat (7.0).

a. Prime Number
Figure 7 bellow shows that Dalvik on Android version 4.2 (Jelly Bean) and 4.4 (KitKat) has better compilation time compared to ART version 4.4 (KitKat) and 6.0 (Marshmellow), this indicates that in version 4.2 (Jelly Bean) and 4.4 (KitKat) Dalvik compiles Java code as good as than ART.

b. Floating Point
Figure 8 shows that Dalvik on Android with version 4.1 (Jelly Bean), 4.2 (Jelly Bean) and 4.4 (KitKat) has a significant comparison to the compilation of the ART runtime. Android with ART version 4.4 (KitKat) and Android version 7.0 (Nougat) has the fastest time, it proves that ART is improving in terms of performance compared to Dalvik.

Figure 7: Prime number experiment result

Based on Figure 7 above it can be concluded that Android with version 4.1 (Jelly Bean) runs relatively slower time, and for Android version 4.4 (KitKat) running on Dalvik has a smaller time compared to ART version 4.4 (KitKat) this proves that on Android 4.4 (KitKat) or hybrid version, experiments performed on this version work well on the Prime Number Java algorithm and its time almost matches other ART runtime.

b. Floating Point
Figure 8 shows that Dalvik on Android with version 4.1 (Jelly Bean), 4.2 (Jelly Bean) and 4.4 (KitKat) has a significant comparison to the compilation of the ART runtime. Android with ART version 4.4 (KitKat) and Android version 7.0 (Nougat) has the fastest time, it proves that ART is improving in terms of performance compared to Dalvik.
Based on Figure 8 above Dalvik on Android 4.1 (Jelly Bean) and 4.2 (Jelly Bean) has the greatest compilation time compared to the others. It can be concluded that Dalvik runtime running on Android version 4.1 (Jelly Bean) and 4.2 (Jelly Bean) has a significant time comparison with other versions, hence it can be concluded that the performance of ART in this experiment is better compared to Dalvik.

c. Recursion (Quick Sort)

Figure 9 shows that the comparison of Dalvik on Android 4.1 (Jelly Bean), and 4.4 (KitKat), has a relatively slower time compared to other Android versions, whereas in experimental data there is considerable distance in certain experiments, this indicates ART more stable than Dalvik. We can see that Android with Dalvik runtime version 4.1 (Jelly Bean) and version 4.4 (KitKat), has the highest time of all, while Android with ART runtime version 6.0 (Marshmallow), and version 4.4 (KitKat) has the fastest time, it denotes that ART is better in its compilation time.

Based on the Figure 10, Android 4.1 (Jelly Bean) has the slowest time compared to Android version 7.0 (Nougat). We can infer that Dalvik on Android with version 4.1 (Jelly Bean) runs slower in running native algorithm, while Android device Android version 7.0 (Nougat) has the fastest time. Based on the data, we can conclude that ART is better running with native C.

V. CONCLUSION

From data and experiment comparison of C and Java in native interface Java based application can be concluded that:

1. ART works better than Dalvik in terms of performance and implementation.
2. Most ART devices compile relatively fast C native algorithms that indicate the use of C native increases compile time at ART runtime.
3. In the experiments carried out device with Android version 4.1 (Jelly Bean), received relatively high time results in terms of compilation time across all algorithms, indicating that Dalvik is significantly behind ART. ART has been improved and can perform the calculation process better.
4. Native C less works well on Android version 4.1 (Jelly Bean) while working with good performance on Android version 6.0 (Marshmallow).

REFERENCES


