DE GRUYTER OPEN

ICTE Journal

2017, **6**(3): 5–13
DOI: 10.1515/ijicte-2017-0011

NATIVE DEVELOPMENT KIT AND SOFTWARE DEVELOPMENT KIT COMPARISON FOR ANDROID APPLICATIONS

Arturo Mascorro[1], Francisco Mesa[2], Jose Alvarez[1] & Laura Cruz[3]

[1]*Polytechnic University of Victoria, 87138 Ciudad Victoria, Tamaulipas, Mexico*
{mascorrocienfuegos@gmail.com}

[2]*Autonomous University of Baja California, 21500 Tijuana, Baja California, Mexico*
{fmesa@uabc.edu.mx}

[3]*Technological University of Guanajuato North, 37800 Dolores Hidalgo, Guanajuato, México*
{laura_cruz_vazquez@hotmail.com}

ABSTRACT

A computational cost comparative study through both Java and C applications was developed. Computational routines consist of a matrix multiplications, the discrete cosine transform and the bubble-sorting algorithm. Memory and Runtime for each application were measure. It was determined that the runtime of matrix multiplication in Java was within the limits of 200 and 300 milliseconds, as opposed to the application developed in C, which shown to be stable with an execution period less than 20 milliseconds. In the ordering algorithm with the bubble method, it was observe that the Java language show be very slow compared to C. In addition, the memory usage was lower in most of the applications, showing a minimum difference. Applications were tested in both, a mobile LG- E510f and a Laptop Toshiba Satellite. The study allowed to report the profit generated in both runtime and memory consumption when performing a native implementation in C and Java.

KEYWORDS

Android, NDK, C Language, JNI.

1 INTRODUCTION

The Android operating system is widely used in several mobile devices such as Smart Phones and Tablets, probably because have a good performance, in addition to being very light.

Both, C language and Java are the programming languages that have managed to hoard the great and most of the programmers around the world. C-language is a strongly typed mid-level language, but with many low-level features. It has the typical structures of the high level languages but, at the same time, it has constructions of the language that allow a control to very low level (Kernighan 1998). The Android software architecture can be subdivided into five layers: core, low-level tools, native libraries, layer macros and runtime (Kyu 2011).

Java technology is characterized by being a multi-platform programming language, interpreted by a virtual machine. In Java distributions is not only the compiler, but also a virtual machine that allows you to run programs on any platform. Java is a high-level object-oriented programming language. The language itself takes much of C++, COBOL, and Visual Basic syntax, but it has a simpler object model and eliminates low-level tools which often induce many errors, such as direct manipulation of pointers or memory. The memory is managed by a garbage collector (Tauro 2012).

Companies that manufacture mobile devices have adopted this operating system to use it in their devices. Currently there are a large number of applications, most of which are developed in Java, probably due to the benefits it presents. For example, its cross-platform feature is very useful for developing applications for mobile devices; but since it is an interpreted language, it tends to be slower. This is a limitation for devices that do not have good resources.

There is the possibility of developing applications using a lower level Java language, which is the C language; this, to facilitate a greater control on the resources of the device, since new applications could be made with a better performance, reusing or creating libraries in language C. Working with C-language on Android increases the complexity and workload for the developer (Descamps-Vila 2011). Although it is feasible to reuse C libraries, sometimes modifications have to be made depending on the architecture with which it is being worked, and it would have to be compiled or recompiled for the different devices to which the application is directed.

Native Android Development Kit (NDK) is a set of tools that allows integrating components that make use of native code in your Android applications (Ki-Cheol 2011). Android applications run on Dalvik's virtual machine. The NDK allows to implement parts of your applications using native code, such as C and C ++ languages. This can provide benefits to certain classes of applications, in the form of existing code reuse and in some cases higher speed.

NDK is a great tool for the Android Software Development Kit (SDK) to combine the power of native x86 codes with the graphical interface of an Android container application (Cheng-Min 2011). While the tool can be used to gain performance benefits in some applications, a certain caution should be taken. For example, if the native library consists of an online assembler in the C code, the code can not only be mounted and worked "as is" in the two different architectures, and some modifications will be necessary (Palmieri 2012). This brings some benefits such as the re-use of code already implemented, or the possibility of developing applications that need to be more aware of memory management; such as a video-call.

The NDK consists of four parts; the first one includes the set of tools and compiled files that allows to generate the code in C or C ++; secondly, how to include native libraries in an apk; thirdly refers to the native headers or headers supported; and end part is the documentation, examples and tutorials. Although it is possible to develop in this way, there is a part of every application that must be written in Java. And to connect everything, a Java Native Interface (JNI) is used, which is designed to handle situations in which they need to combine Java In applications with native code. As a two-way interface, the JNI can support two types of native code, such as native libraries and native applications (Liang 1999).

While the Java programming language is of the secure type, native languages such as C or C ++ are not. As a result, should use special care when writing applications that use the native Java interface. A native method of bad behavior can corrupt the entire application. For this reason, Java applications are subject to security controls before invoking functions. As a general application rule, native methods are usually defined as few classes as possible. This implies a cleaner isolation between the native code and the rest of the application.

The objective of this article is to perform a series of comparisons between different routines executed in C-language and Java. Applications were tested on a LG-E510f mobile device and a Toshiba Satellite Laptop. Routines consisted of a 100 X 100 matrices multiplication, the discrete cosine transform, and the bubble sorting algorithm, with which the execution time and the memory usage in the application are measured.

2. MATERIAL AND METHODS

The methodology used to evaluate the development of mobile applications consist of comparing two programming languages, running different applications in each of them, using Android NDK and SDK. The development of the work was based on the execution of the different applications, such as the multiplication of matrices or the ordering of bubble, as well as in the behavior of programming languages "C" and "Java".

C is a fast and compiled language, but with slow interpreted; while Java is a half compiled language, with half interpreter. That is, Java needs a previous compilation to be able to run in its interpreter Java Virtual Machine (JVM). A LG branded model E510f (LG Electronics, Inc., Seoul, South Korea), with an Android operating system, version 2.3 Gingerbread, with Qualcomm MSM7227T 800MHz processor, Adreno 200 GPU and 512 MB of memory RAM, for the development of the tests, were used. It included a Toshiba Satellite A205-SP5822 Laptop (Toshiba America Information Systems Inc., China) with 2 GB in RAM and a Pentium Dual-Core processor at 1.86GHz.

Multiplication of 100-percent matrices filled with double-type random numbers was the first test performed. It's area which is commonly used in digital image processing. Program was developed in both Java and C languages. Table 1 shows an example of how operations were performed in both languages. The second application consisted in performing the discrete cosine transform of a 16 x 16 matrix. These matrices were loaded with random numbers of double type, commonly used in the compression of data and images (Lam 1998 and Strang 1999).

The discrete cosine transform (DCT) is an invertible linear function, often used in signal and image processing, especially for loss data compression. The action of the DCT can be described in terms of a transformation of matrix A of type:

$$Y = AXA^T \tag{1}$$

where X is a sample matrix, Y is a coefficient matrix and A is a transformation of an N × N matrix. A-elements are represented by:

$$Y_{xy} = C_i \frac{(2j+1)i\pi}{2N} \tag{2}$$

with C_i represented by equation 3:

$$C_i = \sqrt{\frac{1}{N}}(i = 0), \quad C_i = \sqrt{\frac{2}{N}}(i > 0), \tag{3}$$

Calculating formula for discrete transform of the two-dimensional cosine can be visualized in the equation:

$$Y_{xy} = C_x C_y \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} X_{ij} \cdot \cos\frac{(2j+1)y\pi}{2N} \cdot \cos\frac{(2i+1)x\pi}{2N} \tag{4}$$

which is complemented from the data of the recorded matrix in Table 1. Discrete cosine transformation is shown in Table 2, and presents the same dimensions as the matrix of Table 1. It can be observed that the major values are in the triangular upper-left part of the matrix.

Finally, the ordering algorithm was obtained by the bubble method with a 1000 test data randomly filled with numbers from 0 to 99; this to visualize the time in which a certain amount of data in each language is ordered. For development with the NDK, program was generated with the language in Java (Activity, Service), then the folder where the files were created in language C and a file called Android.mk. Applications measure runtime and usage of your routine memories in the "C" and "Java" languages both on the mobile device and on a PC.

3. RESULTS

The first test consisted in multiplying two 100 X 100 matrices, which were filled at random with double type data. Performing 32 routines in both applications, it was observed that the C language in this device proved to be faster to perform the multiplication of matrices.

Making a Box Diagram (Figure 1) it can be observed that the execution time of the multiplication of matrices in Java was within the limits of 200 and 300 milliseconds, as opposed to the application developed in C, which remains stable with an execution period below 20 milliseconds.

31	23	60	93	49	57	57	35
31	21	74	73	55	52	45	44
25	16	82	65	61	38	50	51
26	38	91	61	62	28	49	50
17	85	70	76	50	34	37	43
29	90	68	53	43	25	30	49
116	184	108	25	52	26	38	36
151	144	182	129	44	23	41	39

Table 1: Test matrix for DTC calculation

The second test was worked with the discrete cosine transform. The results obtained in Java showed an average of 247 milliseconds, while in C its average was of 6 milliseconds (Figure 2). In the bubble method ordering algorithm it can be observed that the Java language is extremely slow to perform the ordering, in comparison with the C language. Figure 3 shows the behavior of the execution time of the two routines. In the routine programmed in Java the average runtime is 12,128 seconds and the average runtime in C is 2.9 milliseconds. The results obtained show a comparison regarding the use of the Memory in the applications.

466.25	100.9063	-40.3393	-91.1315	-25.25	2.6791	-3.3152	16.6482
-88.4065	-146.066	-67.5059	26.1243	33.412	44.3008	24.6658	-0.4861
69.5604	73.4218	17.6909	8.1087	-1.2285	31.3229	12.3824	-7.9303
-44.0839	-28.5187	4.3866	18.4638	-11.9507	-32.162	-31.1853	-32.1898
27.5	12.2552	-25.4491	-19.8833	9.5	35.2306	3.2352	-10.1438
-2.2299	6.377	26.7637	18.3775	-13.4471	-20.9802	-10.1116	19.2261
-10.4122	-22.5187	-20.8676	-15.9436	17.2859	25.742	9.5591	-4.747
8.0193	11.5534	7.5823	9.4544	-9.8801	-12.0629	-13.257	-7.4175

Table 2: Transformed Matrix

Figures 4, 5 and 6 shows that the use of memory in the bubble ordering mobile device for C language is less than that used in the Java programming language. Although the difference is minimum 6.94%, this behavior shows be very similar in the case of the discrete cosine transform, except in the shared memory, since this is used more in the C language.

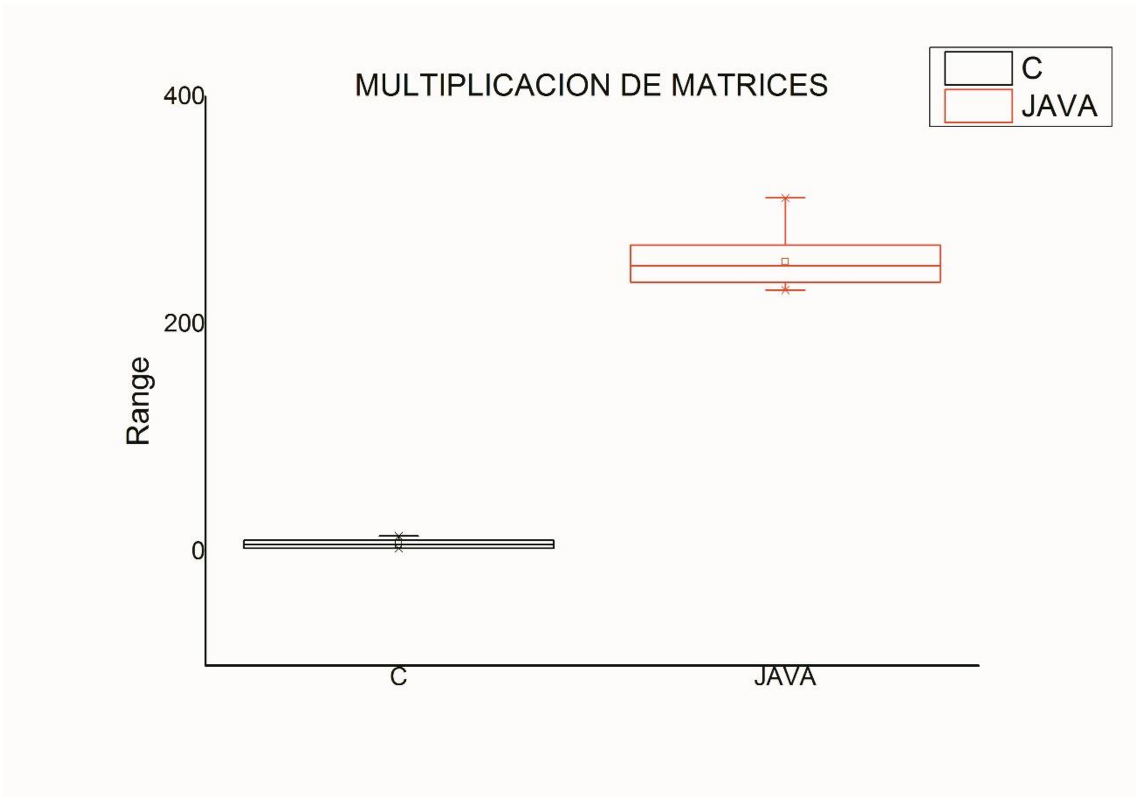

Figure 1. Runtime matrix multiplication on a mobile device.

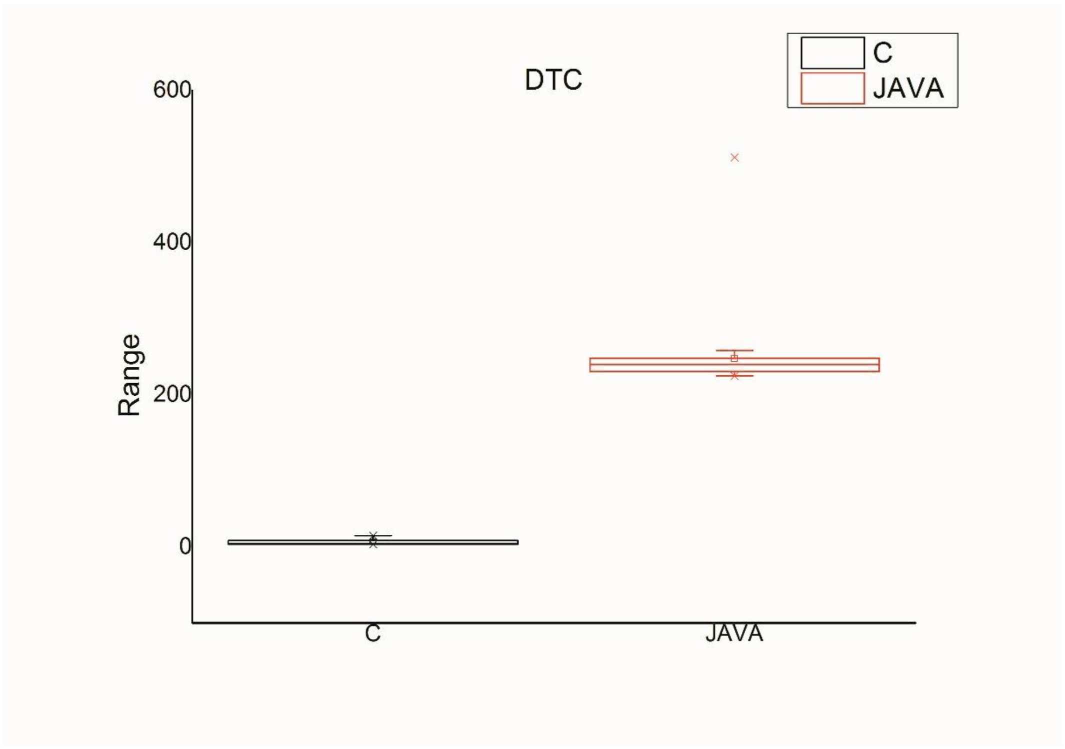

Figure 2. Runtime in the application of discrete cosine transform.

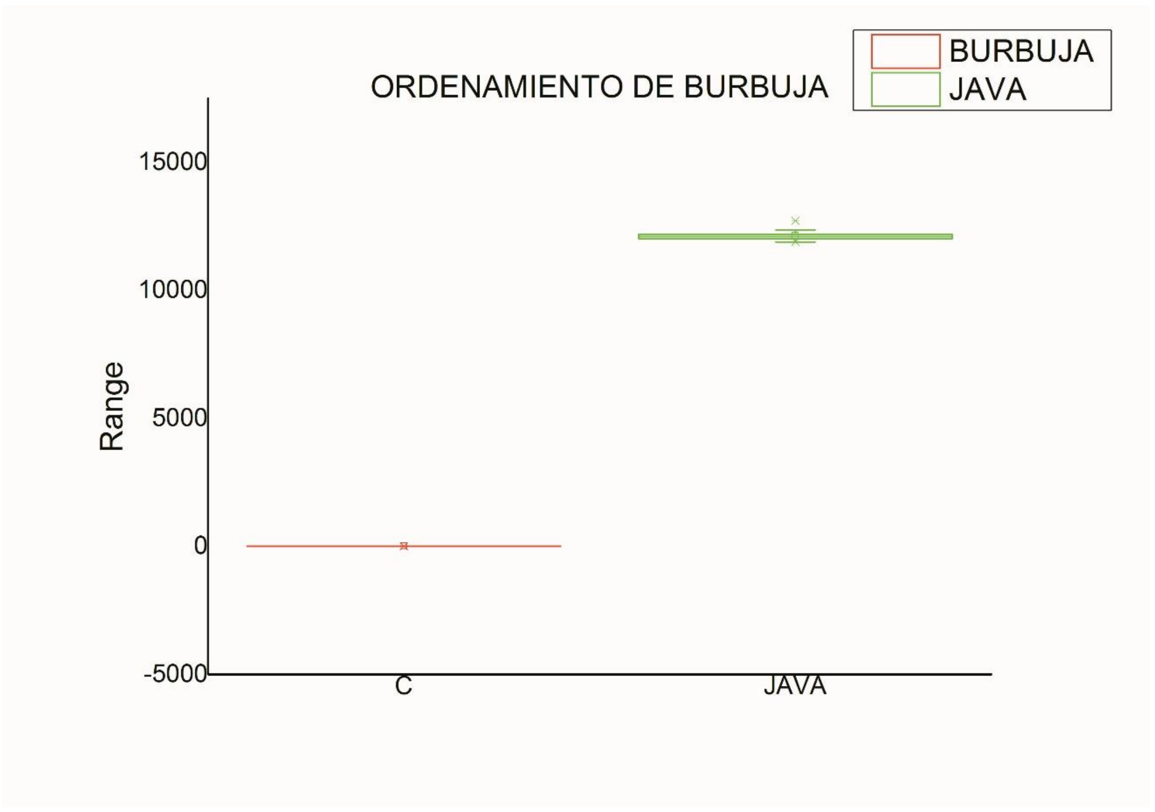

Figure 3. Runtime sorting algorithm on the bubble method in milliseconds.

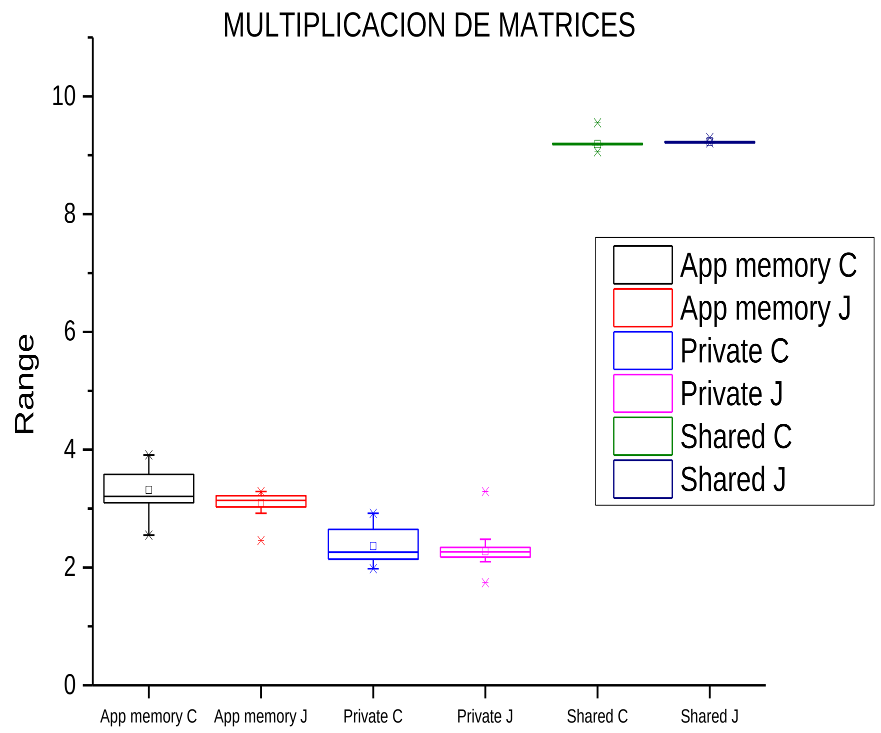

Figure 4. Memory use in implementing matrix multiplication.

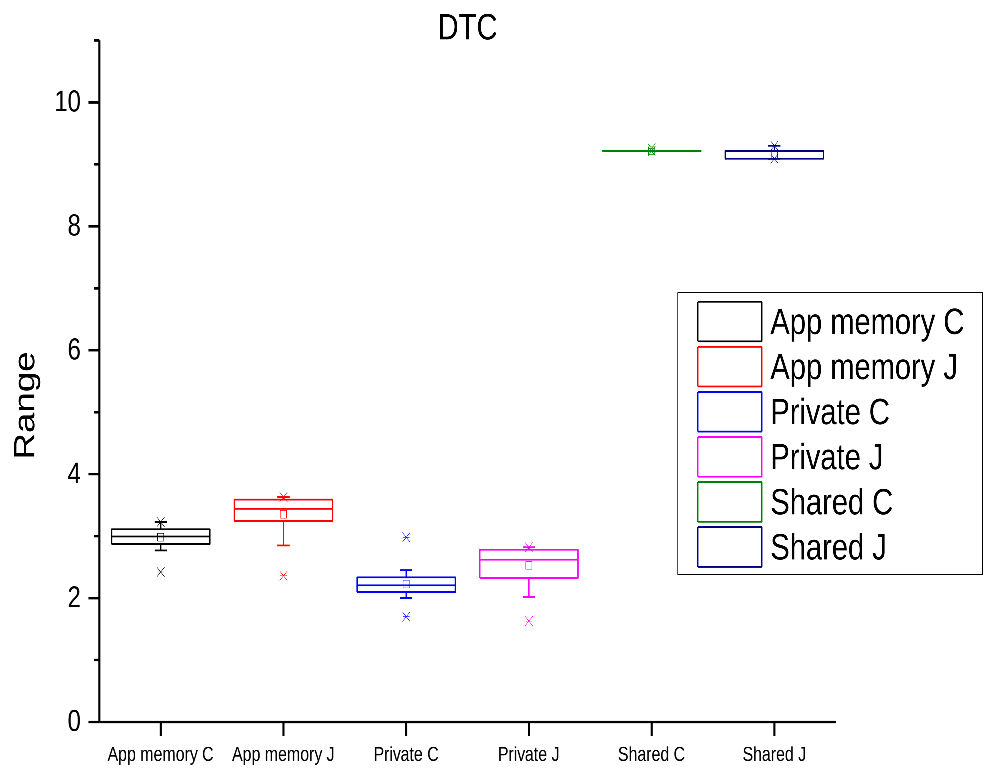

Figure 5. Memory use in the application of discrete cosine transform.

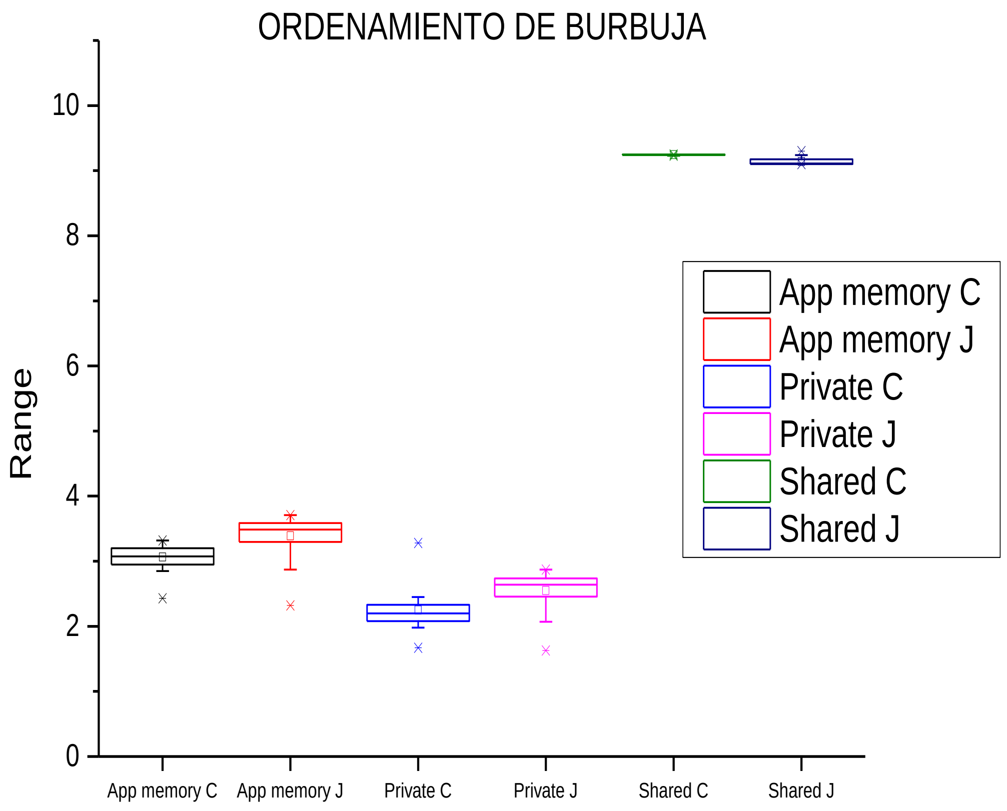

Figure 6. Memory use in applying bubble sort.

Finally, it can be observed in multiplications of matrices, that the use of memory in the C language is wider, with respect to Java.

4. CONCLUSIONS AND DISCUSSIONS

A methodology based on the comparison between different routines executed in C language and Java was presented. Applications were tested on a LG-E510f mobile device, as well as on a Toshiba Satellite Laptop.

In the first test that was performed, it was observed that when working with arrays a greater execution speed of 97.45% is perceived with C programming language. In the second test, in which was worked with the discrete cosine transform, less time was recorded in the "C" language execution, compared to 95.5% Java.

With bubble sorting, the execution time in the Java language had an average of 12.17 seconds being considerably slower when performing this type of operation compared to the "C" language, with a runtime 77.5% faster than Java. These results suggest that the runtime on a mobile device with 800 MHz Qualcomm processor is faster compared to the Android NDK. This is a goodness that offers the C language to be able to develop applications that require a greater number of operations and with the possibility of accessing the devices of the mobile, for example the video camera, accelerometer etc. Doing so in this way increases the difficulty of programming, taking into account that a bad code can damage the entire application or generate unexpected results. In addition, because it is no longer fully available with the benefits of Java, it would have to compile or recompile the code to work on different architectures.

The use of memory in applications is extremely important, so as not to affect the performance of the mobile device. It should be mentioned that when working with the native Android environment, a part of the application is built on Java. This advantage provided by the programming language "C" is used to improve the applications of the mobile devices that require a higher information processing, as it is the video compression or the encryption of the information; these programs use several mathematical functions that require a lot of calculations. The results obtained will help the programmer to have a greater visibility of the tools that can be used at the time of realizing their programs.

ACKNOWLEDGMENTS

The authors wish to express appreciation to the Mexican National Council of Science and Technology (CONACYT), for total support to carry out this study.

REFERENCES

Cheng-Min, L., Chyi-Ren, D., Jyh-Horng, L., et al. (2011). Native Code for Android System. Second International Conference on Innovation in Bio-inspired Computing and Applications, IEEE conference proceedings. 320-323.

Descamps-Vila, L., Casas, J., Conesa, J., et al. (2011). Cómo introducir semántica en las aplicaciones SIG móviles: expectativas, teoría y realidad. V Jornadas de Sig Libre. Servei de Sistemes, D´informació Geográfica i Teledetecció, Universitat de Girona. [http://www.sigte.udg.edu/jornadassiglibre2011/uploads/articulos/art6.pdf].

Kernighan, B. & Ritchie, D. (1998). C Programming Language. Second Edition. Prentice-Hall, USA.

Ki-Cheol, S., Jong-Yeol, L. (2011). The Method of Android Application Speed up by using NDK. Awareness Science and Technology (iCAST), 3rd International Conference IEEE. 382 – 385.

Kyu, L. & Yeol, J. (2011). Android Programming Techniques for Improving Performance. Awareness Science and Technology (iCAST), 3rd International Conference, IEEE conference proceedings. Pp 386-389.

Lam, E.Y. & Goodman, J.W. (1998). Discrete Cosine Transform domain restoration of defocused images. Applied Optics. 37(26); 6213-6218.

Liang, S. (1999). The Java TM Native Interface Programmers Guide and Specification. Addison-Wesley Professional; 1st edition, USA.

Palmieri, M., Singh I. & Ciccchetti A. (2012). Comparison of Cross-Platform Mobile Development Tools. 16th International Conference on Intelligence in Next Generation Networks. 180-186.

Tauro, C., Prabhu, M. & Saldanha V. (2012). CMS and G1 Collector in Java 7 Hotspot: Overview, Comparisons and Performance Metrics. International Journal of Computer Applications 43(11).

Strang, G. (1999). The discrete Cosine Transform. SIA Review. 41(1); 135-147.