Generation and Analysis of Android Benchmarks with Different Algorithm Design Paradigms

Andrws Vieira, Cristian Bosin, Luciano Agostini, Felipe Marques, Julio de Mattos

Group of Architectures and Integrated Circuits – GACI
Federal University of Pelotas – UFPEL
Pelotas, Brazil
{aavieira, cmbosin, agostini, felipem, julius}@inf.ufpel.edu.br

***Abstract*—In embedded application, features like performance, energy consumption, software and hardware size reduction, among others aspect must be considered. However, between on todays embedded systems, the mobile devices with Android operation system are in highlighted spot in the current market, because of the large number of devices sold, thus requiring an analysis of non-functional requirements during the development of applications for these devices, in order that the energy these devices is supplied by a battery. In this line of research, this work will propose an implementation of a lot of applications for Android platform, doing use of a different algorithm design paradigms. These ways being possible to evaluate the impact of these methods for the Android OS embedded in portables devices like, smartphones and tablets. To evaluate the applications that will be describes in this work, will be use tools and applications of profiling that will be provide report about energy consumption, dissipate power and execution time. In this work are shown the results of analyzes of different algorithm design paradigms like recursion, divide and conquer, among other techniques, and the diagnosis of which techniques have a better performance for the Android platform in different situations. In general, recursive structures demonstrate an inferior performance when compared to others methods, also was proved that replacement extensive calculations by static tables shows very high performance gains with only a irrelevant memory storage increase. Furthermore, on the different sorting algorithms analyzed the Tim Sort shows the best performance for the Android platform. Already among the Java Collections when subjected to benchmark developed in this work, the best performance was demonstrated by the ArrayList.**

***Keywords*— Embedded Systems, Android, Energy Consumption, Performance Analysis, Algorithm Design Paradigms**

I. INTRODUCTION

Android is a development platform for mobile applications based on Linux operating system [1] derived from an open source project led by Google. The fact Android is an open source project, allied with Apache 2.0 license, makes it flexible, allowing developers and companies to make customizations without need of sharing such changes. The Android application development is simplified by the Software Development Kit (SDK) that provides tools and APIs needed to develop applications, favouring an easy integration with many resources available on the device. During the development of mobile applications, it is necessary to analyze non-functional requirements, such as performance, power dissipation and energy consumption, because these applications run on battery-based devices.

Performance analysis of an Android application can be done with Traceview [2], and power and energy consumption estimations can be obtained with PowerTutor [3]. Traceview is a graphical viewer for execution logs that can be done with the Debug class, this way is generating log tracing information of the source code. Traceview can help a debug any application and profile its performance [2]. With the help of this tool, it is possible to find bottlenecks in an application, allowing developers to make modifications that increase the application's performance as much as possible. PowerTutor is an application for Google devices that shows the power consumed by major system components, such as CPU, network interface, display, GPS receiver and different applications. This application allows software developers to see the impacts of design changes on power efficiency. Application users can also use it to determine how their actions are impacting battery life. PowerTutor uses a power consumption model built by direct measurements during careful control of device power management states. This model generally provides power consumption estimates very close to actual values, with a 5% error margin. A configurable display for power consumption history is provided. It also provides users with a text-file based output containing detailed results. One can use PowerTutor to monitor the power consumption of any application [3].

The focus of this work is shows a comparative analysis between different design patterns. Among them, it is shown the impact of recursive and iterative algorithms, algorithms that use more memory versus algorithms that use more CPU and the most common data structures of Java Collections Framework.

II. STATE OF THE ART AND RELATED WORKS

The current increase of processing capacity of mobile devices brings as consequence an energetic consumption increase by applications. Since it is undesirable (because decreases battery life), the research for methods which avoid this is necessary.

In this year is expected that mobile devices (smartphones and tablets) will overtake PCs as the most widely used devices for Internet access [4]. The most modern smart-phones are capable of running video and audio in high definition, to

provide high speed access on internet, allow taking pictures and making videos on high quality - HD - besides having interface with sensors such as GPS and accelerometer.

The focus of this session is to describe and introduce systems that perform real-time estimates of energy consumption on mobile devices with Android. PowerRunner and PowerTutor are results of research related to this work.

PowerRunner is automated management software that provides energy and optimizes energy consumption on mobile devices, using learning Linux kernels and a low-level API for power management at runtime [5]. Modern Linux kernels define a structure for the code to facilitate power management off and on an individual component of the system at runtime. Through careful monitoring and learning the behavior of the system components used by various user activities, PowerRunner analyzes and estimates the resources required by an activity on Android. In order to maximize the user experience, minimize power requirements and maximize energy savings by applying various combinations of optimizations at runtime. The project has proven to save up to 25% of energy consistently.

PowerTutor is a system for estimating energy consumption in real time implemented for the Android platform smart-phone. PowerTutor provides accurate estimates of energy consumption in real time to the hardware components, including CPU and LCD display, as well as GPS, Wi-Fi, audio and mobile network interfaces. This tool was chosen because it provides estimates of energy consumption by hardware components, allowing an analyze the behavior of energy consumption in different components of the device.

III. Description of Developed Work

A. Algorithms that use more memory versus algorithms that use more CPU

The sine and cosine functions are a greater example for this analysis, because is possible make a static table with these results and access them. And also calculate the sine (1) and cosine (2) by Taylor Series [6]. The trigonometric functions Java implementations also will be used (which is the Math class) [7].

$$\sin(x) = \sum_{n=0}^{\infty} \frac{(-1)^n \cdot x^{2n+1}}{(2n+1)!} \quad (1)$$

$$\cos(x) = \sum_{n=0}^{\infty} \frac{(-1)^n \cdot x^{2n}}{(2n)!} \quad (2)$$

For this, it was previously calculated a table with the sine and cosine values between zero and ninety with an accuracy of a tenth (for to make a sine's table and cosine's table). With these values is possible to calculate all the others values, just changing the signal and the accessing order in the table. This can be done implementing a hash function for to access this table.

B. Sorting Algorithms

A computational problem that arises with frequency is the sorting problem, which consists, basically, in sorting a numeric sequence in increase order. Formally the sorting problem can be defined the following way:

Input: a sequence with *n* numbers: $\langle a_1, a_2, \dots, a_n \rangle$.

Output: a permutation (reordering): $\langle a_1', a_2', \dots, a_n' \rangle$ of the sequence so that $a_1' \leq a_2' \leq \cdots a_n'$.

A sorting algorithm can be classified by computational complexity of the elements comparison in terms of the size of the sequence n, by computational complexity of the swap realized, memory used, among others.

In general for to determinate if an algorithm is more efficient than other, it is analyzed the asymptotic growth of the algorithm, in others words, how is the comportment of this algorithm comparison in terms of the size of the sequence for very large *n*, *in limit*.

For the development of this part, it was selected and analyzed different sorting algorithms to evaluate the impact of the characteristics of the energy consume of each algorithm in embedded devices. Among the popular sorting algorithms, those chosen were: Merge Sort, Heap Sort, Counting Sort, Tim Sort and Insertion Sort (iterative and recursive versions).

The data set used as entry was the same to all algorithms. The data entry consists in an array of 200 positive integer numbers, where the values of this set range from 0 to 200.

C. Java Collections Framework – List Interface

Since Java 2SDK, the platform J2SE includes a collections framework (called Java Collections Framework – JCF). The JCF is a set of classes and interfaces that implements some generic collection data structures [7]. On the other hand a collection is an object that represents an object group. With this, a collections framework is a unified architecture for representing and manipulating collections, allowing them to be manipulated independently of the details of their representation. It reduces programming effort while increasing performance. It allows for interoperability among unrelated APIs, reduces effort in designing and learning new APIs, and fosters software reuse. The framework is based on nine collection interfaces. It includes implementations of these interfaces, and algorithms to manipulate them.

As JFC consist in many different generic data structure implementation, only List interface will be analyzed.

The List interface extends Collection and defines ordered collections (also known as sequence). In most cases a List may be compared to a primitive array (the access can be done by numeric index) with insert and remove methods.

There are three classes that implement List interface: ArrayList, Vector and LinkedList. The LinkedList class implements a doubly linked list. The insert, remove and index operations into the list will traverse the list from the beginning or the end, whichever is closer to the specified index (complexity $O(n)$).

The ArrayList class is as primitive array but it has any facilities. Internally it implements an object array. But when the size of this array isn't enough, a new array 1.5x bigger is allocated and all elements is moved to this new array. The insert and remove operation run in a linear time ($O(n)$). But this linear time is amortized, because only elements after the

stated index are displaced. The access operations run in a constant time.

The Vector class is equivalent to ArrayList with synchronized operations [9].

1) Benchmark: For this analysis was determined a set of operations that will be tested the same way in all List interface implementations (ArrayList, LinkedList and Vector). Also will be tested the Java primitive array. This set of operations was defined in six steps and each step run the following routine: (a) Values are inserted into a list; (b) The elements are removed one by one, in increasing order; (c) The same vector is reinserted; (d) The elements are removed one by one, in decreasing order; (e) The same vector is reinserted; (f) The elements are removed at random order (predetermined);

Each step runs this way: (1) Run the routine with an increasing sorted vector; (2) Same as with step 1, but with a decreasing sorted vector; (3) Same as with step 1, but with a random vector; (4) Same as with step 3, but after insertions, reverse the list; (5) Same as with step 4, but sort the list; (6) Same as with step 4, but reverse sort the list.

IV. Results Achieved

All these analyze were performed on a Tablet Coby Kyros MID7016 with the following features: Processor Telechips ARM 11 800 MHz, 256 MB RAM and 4GB flash memory for data storage and Android OS 2.3. After the process of developing and testing of algorithms have been finalized, all algorithms were then installed on the device to start the analysis.

A. *Analysis of algorithms that use more memory versus algorithms that use more CPU*

The main objective of this session is to present a method that reduces the information processing by the CPU of the device in order to save energy, aiming at a longer battery life.

First the applications sine and cosine only 900 values were calculated (in a range from 0 to 90). In this case, the table had all the values.

The cosine and sine had a similar comportment, where the table shows the best efficiency, as illustrated in Figure 1 illustrates. Because this, the sine graphics were occulted.

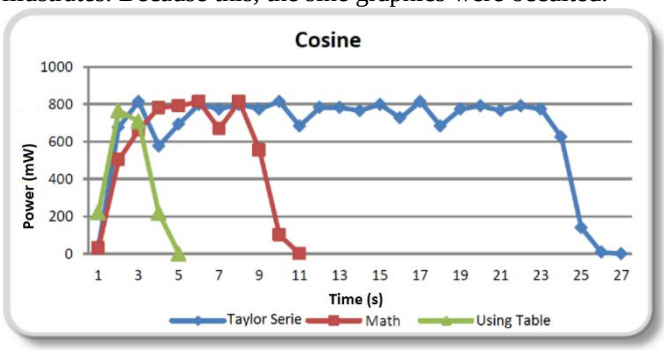

Fig. 1. Execution of the Cosine

But this implementation only returns results in first quadrant of the trigonometric circle. For the results of the other quadrants, just implement a hash function that convert any angle between 0 and 360 for a corresponding value into table. However, in this case, the result shows a difference from the first, where the table version was worse than version that uses the Math class, as illustrated in Figure 2.

The fact that table with the hash function was worse than Math class solution, suggest that Math class contain a table with the values previously calculated using a fast hash function.

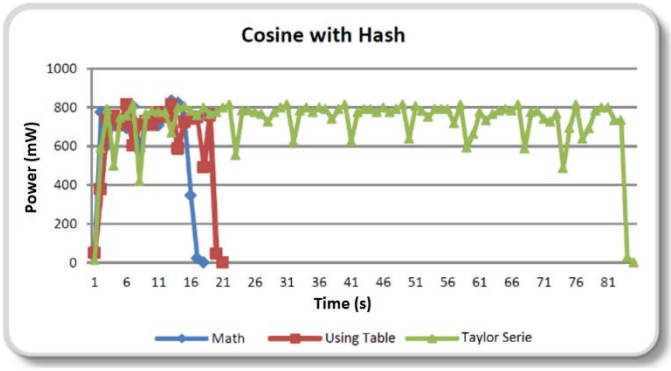

Fig. 2. Execution of the Cosine with Hash Function

Although the input set is small for this case study, is possible to note the use of a table containing the data previously calculated always becomes more efficient than to calculate each time it is needed. The only hitch is the use for additional memory for store this values, what is not really a problem in current devices.

B. *Sorting Algorithms*

This section will show the results of the analysis of energy consumption of sorting algorithms. The Figure 3 shows a comparison between the runtime in seconds (s) and the power dissipation in milliwatts (mW). As can be seen, the algorithms that had the worst performance were Insertion Sort (both implementation) and Merge Sort. As Table I shows, Merge Sort has complexity $O(n \lg(n))$ and Insertion Sort has complexity $O(n^2)$, but the Merge Sort has worst performance than Iterative Insertion Sort. This happens because in Android platform recursive algorithms do not show good results [8] [10].

The Table I also show that Tim Sort had the best performance, running in 10s and consuming 7023mJ, followed by Counting Sort and Heap Sort.

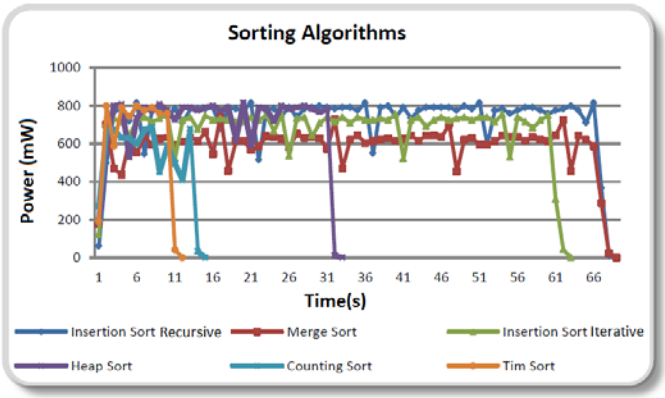

Fig. 3. Sorting Algorithms

TABLE I
SORTING ALGORITHM: ENERGETIC COST AND PERFORMANCE

Algorithms	Complexity	Energy Consumed (mJ)	Time (s)
Rec. Ins. Sort	$O(n^2)$	49742	66
It. Ins. Sort	$O(n^2)$	42290	60
Merge Sort	$O(n \lg(n))$	40144	66
Heap Sort	$O(n \lg(n))$	22866	31
Counting Sort	$O(n + K)$	7466	13
Tim Sort	$O(n \lg(n))$	7023	10

C. *Java Collections Framework – List Interface*

This session will present the results obtained from of analyze of some Java Collections (ArrayList, LinkedList and Vector) and Java primitive array.

The Figure 4 shows the runtime versus dissipate power by each structure. It is noted that the ArrayList presented the best performance, running in 17s. On the other hand, the LinkedList show the worse performance, running in 33s, which also influences higher power consumption (22.3 J).

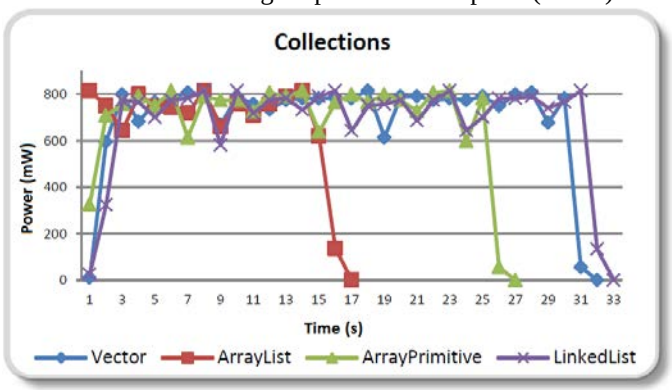

Fig. 4. Java Collections. Time in seconds and Power in miliWatt.

In real systems, the synchronization in Vector methods causes an unnecessary overhead, because even when there is access multitasking to the list (what does not happen in most cases), almost always is needed do a new synchronization in the methods for to atomize the concurrent operations into Vector.

So if there is no the use of multiple threads, for reasons of efficiency, the ArrayList is the most suitable.

V. CONCLUSIONS

This paper presented an evaluation of performance and energy consumption of applications for the Android platform in order to find a pattern of energy consumption in relation to the design paradigm of algorithm used, so diagnosing the best version of a particular algorithm to an end specific. Given that most modern mobile devices are capable of running video and audio in high definition, providing access to high speed internet, allow pictures and videos of high quality, besides having interface with sensors like GPS and accelerometer. But with all these features the challenge is to extend the battery life at this point is that this work fits.

It was evaluated algorithms that use more memory versus algorithms that use more CPU, recursive and iterative algorithms with the same complexity, and the impact of using Java Collections Framework (in this case, the implementations of the List interface), within the Android platform.

With the aid of the tool PowerTutor [3] which provides estimates of energy consumption by hardware components, it was possible to analyze the behavior of energy consumption of the device running Android.

In cases where a good solution to a problem involves a lot of calculations, and memory is not a crucial factor for the project, the use of a table with some (or even all) of the pre-calculated values becomes much more efficient than calculate every time the result using a function of recurrence or where the complexity of calculation is large, causing a relative delay. On the other hand, the results showed that recursive algorithms in Android platform cause a great loss of performance and a high energetic consumption [8]. This is a reason for the Tim Sort had a better performance compared to other sorting algorithms, justifying its recent inclusion as a standard sorting algorithm java language.

Among the collections of Java, ArrayList showed the best performance when subjected to the benchmark that was developed to assess in general the use of collections. With these results it was found that except in very specific cases when studies shown that another class may have superior performance, it is preferable to use ArrayList in Java for Android.

The importance of the results obtained in this study lies in the fact that we can determine the impact of design changes in software power dissipated by mobile applications for the Android platform.

REFERENCES

[1] R. R. LECHETA, Google Android: Aprenda a criar aplicações para dispositivos móveis como o Android SDK, 2ª ed., São Paulo: Novatec Editora, 2010.

[2] G. Android, "http://developer.android.com/index.html," 10 2011. [Online].

[3] L. Zhang, R. Dick, Z. Mao and L. Yang, "PowerTutor," 2012. [Online]. Available: http://powertutor.org. [Accessed March 2012].

[4] "Gartner," 2010. [Online]. Available: http://www.gartner.com/it/page.jsp?id=1278413. [Accessed Abril 2012].

[5] E. Kreiman, "Using Learning to Predict and Optimise Power Consumption in Mobile Devices," Longdon, 2010.

[6] J. Stewart, Cálculo, vol. II, São Paulo: Thomson Learning, 2007.

[7] "Oracle," 2012. [Online]. Available: http://docs.oracle.com/javase/tutorial/collections/intro/index.html. [Accessed January 2013].

[8] A. VIEIRA, D. DEBASTIANI, A. L., F. MARQUES and C. MATTOS, "An analysis of power and performance of applications for mobile devices with Android OS," in *South Symposium on Microelectronics*, São Miguel das Missões, 2012.

[9] "Javafree.org," 2013. [Online]. Available: http://javafree.uol.com.br/artigo/940/ArrayList-ou-Vector.html. [Accessed January 2013].

[10] A. VIEIRA, D. DEBASTIANI, A. L., F. MARQUES and C. MATTOS, "Performance and Energy Consumption Analysis of Embedded Applications Based on Android Platform" in *Computing System Engineering (SBESC)*, Natal, 2012.