The AM-Bench: An Android Multimedia Benchmark Suite

Chayong Lee Euna Kim Hyesoon Kim
School of Computer Science
Georgia Institute of Technology
{*chayong82, euna.kim*}*@gatech.edu*, hyesoon@cc.gatech.edu

***Abstract*—Despite the significant evolution of mobile devices and the increased use of mobile devices, not many mobile benchmarks have been studied. Even though mobile applications share similar characteristics with traditional desktop oriented applications, different programming environments and user usage patterns present different characteristics. In this paper, we introduce an open source based mobile multimedia benchmark for Android platforms (AM-Bench). The AM-Bench consists of several multimedia benchmarks running on Android platforms. We explain the characteristics of the AM-Bench and compare performance on four Android-based platforms.**

I. Introduction

The popularity of Smart phones shifts the major computing paradigms. Mobile platforms have different types of applications than those of traditional desktop applications. Nonetheless, not many benchmarks are available to understand the characteristics of the benchmarks and architectures.

Traditional benchmark suites such as MiBench [19] and SPEC benchmark suite [25] are not adequate to characterize the performance of mobile devices due to the compatibility issue with mobile device platforms such as Android and iOS. These new systems all run programs written in new object oriented program languages such as Java and Objective-C, which show very different program characteristics than many other desktop applications. This fact provides the motivation to propose a new commercially free benchmark suite for mobile devices. We present Android Multimedia Benchmark Suite, called AM-Bench. AM-Bench is composed of Android open source applications that are drawn from the multimedia domain. Our survey of Android applications shows that multimedia benchmarks are one of the most commonly used application categories. Since mobile applications are heavily dependent on underlying operating systems and programming platforms such as libraries and we are targeting the development of an open source benchmark suite, an Android operating system is chosen. Android instrumentation [17] also provides end users an easy interface for running each benchmark in AM-Bench.

In this paper, we analyze end users' activities on mobile devices and discuss the representatives of Android-open source application in the multimedia domain. We then describe our benchmarks, Android Multimedia Benchmark Suite, called AM-Bench. Using AM-Bench, we evaluate the performance of four different mobile devices in terms of application execution time and CPU performance on NVIDIA Tegra 2 [8], Samsung Galaxy Tab 7.0 [10], Galaxy Player YP-GP1 [9] and Nexus One architectures [4].

II. Background

The Android system includes a three part of software layer; mobile applications, middle-ware, and operating system [18]. Most Android applications are written in Java, and they are compiled as the .class file format. The Android system converts .class files into a Dalvik Executable (.dex) file and executes on the Dalvik virtual machine in the middle-ware layer. Each Android application runs on its own process with its own instance of the Dalvik virtual machine.

A. Android Software Stack

1) **Application Layer:** The application layer is the highest layer in the architecture and contains the built-in applications such as Phone Dialer, Email, Contact, Camera, Gallery and others. The application framework layer provides the core platform services such as activity manager, package manager, window manager and others. This layer provides an abstract interface to hardware access and also provides a way to manage the user interface or application resources. The application framework layers enable developers to piece together the services to create an Android application.

2) **Middle-ware Layer:** The middle-ware layer includes C/C++ core libraries and Android Run Time. C/C++ core libraries can be classified into four categories; bionic libc, function libraries, native servers, and hardware abstraction libraries. These core libraries support built-in Android-specific services and standard video, audio and other data storage.

The Android run time comprises core libraries that are required for Java libraries and the Dalvik VM, which is a register based virtual machine. The Dalvik VM is optimized for low memory requirements and designed to support multiple virtual machine processes per device with optimized bytecode interpreter using runtime memory efficiently. The Dalvik VM also supports process isolation, memory management and threading from the underlying operating system.

3) **Operating System:** Android relies on the Linux Kernel and communicates underlying hardware. This layer includes Android's memory management programs, security settings, power management software, several drivers, file system access, networking and inter-process-communication.

B. Memory Limitation

One of the key differences between mobile applications and desktop applications is the limitation of memory usage. Multimedia applications often deal with memory intensive items such as images, audio, and video, so the memory limitation is the key challenge of multimedia application on mobile devices. Also, how to utilize the memory limitation significantly determines the performance of the application. Typically, each application on an Android platform is allowed to use 16MB of memory (24 MB for recently released Android devices) [1]. Android devices with 16 GB and 32 GB are common these day; however, that is solid state storage and not RAM.

In AM-Bench, we exam how Android devices can handle this memory limitation issue and how they utilize the secondary storage to overcome this problem. We will describe this problem in more detail using the FBReader application.

C. Java Native Interface (JNI)

Some of the benchmarks in AM-Bench examine a computational performance using JNI. JNI refers to Java Native Interface [5] and it allows the interaction between Java VM and native code. There are two types of interactions [14], downcalls and upcalls. Downcalls is what Java applications call a native method[1] and upcalls is what a native method needs to access Java applications for the application's resources. Although the Android system does not include Java VM, it is still possible to use the JNI interface in an Android application using Android NDK [2].

III. Motivation

A. Existing Benchmark Suites

To the best of our knowledge, a benchmark suite specifically for Android mobile applications has not yet proposed. There are some benchmarks for mobile devices, however, they typically they target one specific feature on mobile devices. Also traditional benchmark suites such as MiBench [19] or PARSEC [13] are not adequate to characterize the performance of mobile devices due to the compatibility issue with mobile devices' operating system such as iOS, or Android. SD-VBS [26] is the San Diego Vision Benchmark Suite that comprises vision applications drawn from the vision domain. However, the applications are written in Matlab and C so there is no known way to run code on an Android platform. DENBench [24] is a software benchmark that contains widely used algorithms in mobile applications such as AES (Advanced Encryption Standard), DES (Digital Encryption Standard) and others. However, DENBench only evaluates the media library and this is not a collection of representative mobile applications. DENBench is not an open source or commercially free. GPSBenchmark [7] only examines the GPS performance of mobile devices and the source code is not available. JBenchmark [23] is a graphical benchmark for Java phones and PDAs. JBenchmark has several features such as heavy load on animation, mapping tests, user interface simulation and etc. However, JBenchmark is only compatible with J2ME enabled devices. GLBenchmark [22] is a 3D test suite designed for OpenGL ES in various operating systems; Android, iOS, Linux, Symbian and Windows Mobile. This is a licensed commercial 3D testing tool that does not allow source code to be manipulated. Recently, BBench [20] was proposed for modern Smart phones. However, BBench only targets Web browsers.

[1]In this paper, method means the method in Java programming language.

B. User Preferences of Mobile Application

To analyze user preferences of mobile applications, the top 50 Android applications have been investigated [15]. We assume that the most frequently downloaded applications would run on the mobile devices most frequently.[2]

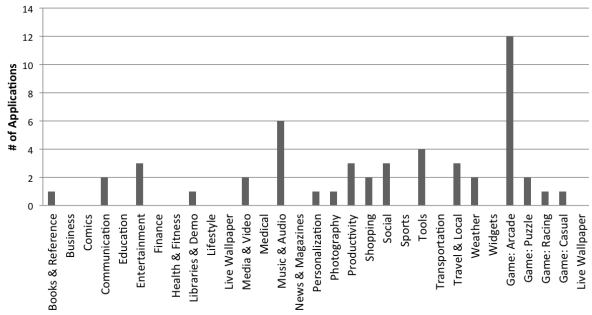

Fig. 1. Top 50 Android Applications

Figure 1 shows the top 50 free Android applications available in the official Android market from mid 2011. Since the Android market categorizes the application into several categories, we narrow down the categories to ensure the users preferences more clearly. We suggest three new categories; Information/Communication, Multimedia, and Utility. We rearrange the top 50 Android applications into the new categories as shown in Table I. As the Table I shows the total number of applications in Multimedia categories is more than twice larger than the total number of applications in the Information/Communication category and three time larger than the total number of applications in the Utility category.

If we consider native multimedia Android applications such as Camera, Gallery, Music, and others, the importance of multimedia applications is somewhat higher than what we can see from Table I. Based on this, we can ensure that the most frequent and important user activities on Android devices are related to the multimedia applications. This fact also conforms to the necessities of our benchmark suite, which is composed of multimedia Android applications.

IV. Benchmark Suite

In this section, we describe the characteristics of our benchmark suite, Android Multimedia Benchmark Suite, AM-Bench. AM-Bench is composed of six Android open source applications that are drawn from the multimedia domain to

[2]Although the number of downloads does not necessary represent actual usages, we believe that strong correlations still exist.

TABLE I
NEW CATEGORIES OF ANDROID APPLICATIONS

Category	Sub-Categories	Number of App.
Multimedia	Game, Books&References, Comics, New&Magazine, Entertainment, Photography, Media&Video, Music&Audio, Sport, LiveWallPaper, Widgets	29
Information & Communication	Business, Education, Finance, Weather, Medical, Transportation, Communication, Health, Fitness, Lifestyle, Medical, Social, Shopping, Travel&Local	12
Utility	Libraries&Demo, Personalization, Productivity, Tools	9

benchmark multimedia features on Android devices. Two applications in AM-Bench are selected from the Android native applications. Four applications are chosen based on the popularity and total number of downloads, such that it can ensure the overall performance and functionality of the open source applications.

A. Benchmark Design

AM-Bench covers four essential multimedia features; playback, compression, computation and rendering. Playback refers to the ability to decompress multimedia contents from a file system and load the multimedia contents on the application. Compression refers to the ability to compress a raw data from the built-in multimedia hardware component on the mobile device to a multimedia format. Computation includes the ability to handle heavy computational loads such as a matrix multiplication and a floating point calculation. Rendering refers to the capability to draw 3D objects on the screen. As shown in Table II six AM-benchmark applications examine at least two multimedia features.

B. Applications

1) **FBReader [3]:** FBReader is the most popular free Android e-book reader application. FBReader was selected to evaluate the e-book load process and text search operations on mobile devices. FBReader loads a complete e-book from a file system into the memory to allow fast computations on searches or text re-sizing. FBReader does not implement any sophisticated algorithm for text search but it simply iterates over all paragraphs in text model and over text blocks in each paragraph to search the text. As a result, the search operation will examine how the capacity of memory will affect on the performance of mobile devices.

Optimizations Because of Memory Limitations: Since each application can only use up to 16 MB of memory on an Android platform, FBReader can not store the entire book model onto the memory. Instead, it stores a compact version of the book model in the memory and serializes the book model onto the flash memory in a 100KB block. It then uses Java weak reference pointers to store the text blocks so it provides reading of the model from the text block as fast as reading from RAM.

2) **Barcode Scanner [12]:** Barcode Scanner scans barcodes on CDs, books, and other products. Barcode Scanner uses the ZXing library, which is an open source barcode image processing library. It decodes barcodes using the built-in camera on mobile devices. Supported formats are UPC-A/E, EAN-8/13, QR code and others. Barcode Scanner implements its own algorithm for the QR code detector. The key elements of the algorithm are the following.

- Making threshold decisions on Gray scale data to get binary data.
- Sampling raw image pixels into modules; each QR code module must be spanned by raw image pixels.
- Detecting of alignment patterns.
- Image rotation and correct perspective distortion.

Barcode Scanner was selected to evaluate the image decode processing and the performance of the image analysis algorithm.

3) **libGDX 2D [6]:** LibGDX is a commercially free game development framework written in Java/C/C++ and many Android game applications are based on the libGDX engine. LibGDX 2D was chosen to examine the performance of 2D graphics and related computations on mobile devices.

Table III describes seven benchmarks in libGDX 2D. 2D animation exams the basic abilities to draw 2D images on the screen of mobile devices. Sprite and Sprite Batch perform a full stress computation to check the maximum capacity of 2D computation. Matrix Computation compares the execution time of 4 by 4 matrix multiplications from Java source code and C native source code using JNI. Included computations are a basic matrix multiplication, a vector multiplication, and a matrix inversion. Parallex tests an animation moving side to side. Culling exams the ability to identify visible objects on screen. Vertex Buffer Object tests methods for uploading data such as vertex, normal, color, and other information in a given array to the video component on mobile devices.

4) **libGDX 3D [6]:** 3D graphic technology on mobile devices has been dramatically improved. Also, 3D graphics applications have been taken an important workload of mobile devices. Some of the mobile devices already have a GPU to enhance the performance of 3D graphics. Therefore it is important to benchmark the performance of 3D graphics on mobile devices. The libGDX 3D benchmark runs 3D modeling loaders to measure skin time, render time, and FPS.

5) **Camera [15]:** Since users carry mobile device in their daily lives, a built-in camera on mobile devices is often used to capture photographs. Camera is an Android built-in application that can capture photographs or record videos. AM-Bench includes a Camera benchmark to test the performance of built-in cameras on mobile devices. The Camera benchmark exams four primary operations of built-in cameras on mobile devices: start up, latency, JPEG call back, and switch mode. Detailed descriptions are explained in Table III.

6) **Gallery [15]:** Gallery is also an Android native application that displays multimedia contents on mobile devices.

TABLE II
AM-BENCH MULTIMEDIA FEATURES WITH SUB-ELEMENTS

Feature	Element	FBReader	Barcode Reader	libGDX 2D	libGDX 3D	Camera	Gallery
Media Playback	Still Image	O	O				O
	Audio						O
	Video						O
Media Compression	Still Image					O	
	Audio					O	
	Video					O	
Computation	Physic			O	O		
	Recognition		O				
	Search	O					
Rendering	2D			O			
	3D				O		

TABLE III
AM-BENCH SPECIFICATION

Application	Benchmark	Description	Input	Category
FBReader	e-book Process	Load e-books from a file system to application	ebooks(EPUB, MOBI)	Media Playback
	Text Search	search words in e-book	10 words	Computation
Barcode Scanner	Decode QR image	Decode QR images and data analysis	10 QR images	Media Playback, Recognition
libGDX 2D	2D Animation	Load 2D animation on screen	2DAnimationTest	2D Rendering
	Sprite	Exam Sprite performance	SpriteTest	2D Rendering
	Sprite Batch	Exam Sprite Batch performance	SpriteBatchTest	2D Rendering
	Matrix Computation	Perform 4 by 4 matrix multiplications from Java and native C source code using JNI	MatrixJNITest	Computation
	Parallex	Load 2D animation moving side to side	ParallexTest	2D Rendering
	Culling	Identify visible objects on screen	CullingTest	Computation
	Vertex Buffer Object	Upload object data in a buffer to a video component	VBOVOTest	Computation
libGDX 3D	3D Render	Render 3D images on screen	MD5Test	3D Rendering
	3D Skin	Draw 3D images on screen	MD5Test	3D Rendering
	3D Animation	Load 3D animation on screen	MD5Test	3D Rendering
Camera	Start up	Turn on built-in camera on mobile devices	Built-in Camera	Media Compression
	Latency	Check the delay between shutter triggered and a photography taken	Built-in Camera	Media Compression
	JPEG Callback	Create a JPEG image from raw data	Built-in Camera	Media Compression
	Switch Mode	Switch application activities between camera mode and camcorder mode	Application Activities	Media Compression
Gallery	Initial Building	Populate multimedia albums and items. Create meta data	JPEG Images	Media Playback
	Thumbnail Loading	Load thumbnail images	JPEG Images	Media Playback
	Full-size Loading	Load a full-size image from a thumbnail image	JPEG Images	Media Playback

As described in Table III, the application measures the performance of media decompression in three different places. The initial Building benchmark shows how the process of creating meta data affects on the multimedia decompression process. We will exam this in more detail in the evaluation section. In the Thumbnail Loading benchmark, the total number of images loaded at the first time users select an album depends on the screen size of mobile devices. Thus, we calculate the average of the execution time.

V. PERFORMANCE EVALUATION

A. Methodology

1) Profiling Devices: We select four Android devices that have different hardware configurations to gather the performance data using AM-Bench. Table IV specifies the characteristics of the evaluated Android devices.

2) Android Instrumentation and Measurements: We modify Android instrumentation [16] to provide a test frame for benchmarking. Android instrumentation is based on JUnit[21] and was originally designed to test Android applications. First, the modified Android instrumentation provides a fixed set of user inputs so that benchmarks always produce consistent results. Second, the modified Android instrumentation provides an interface to a running application to invoke benchmark methods.

Figure 2 summarizes the process of using Android Instrumentation to collect benchmark data in AM-Bench. Application refers to the target application in AM-Bench such as FBReader, Barcode Scanner, or Native Camera. AM-Bench Control Package is the set of methods we added to the applications to gather benchmark data such as application execution time and CPU performance of the application. Android Instrumentation [16] allows Benchmark Controller to access the target application. Benchmark Controller is an application that controls the benchmark process by invoking control methods in the target application or triggering an application's U.I. events.

By attaching Android Instrumentation to the target application, Benchmark Controller can invoke control methods in the AM-Bench Control Package through Android Instrumentation. If the target application was previously running, Android Instrumentation kills the application to attach itself to the application on the same process.

TABLE IV
CHARACTERISTICS OF THE EVALUATED ANDROID DEVICES

Feature	NVIDIA Tegra 250 Dev.Kit	Samsung Galaxy Tab 7.0	Nexus One	Samsung Galaxy Player
Operating System	Android 2.2 (Froyo)	Android 2.2 (Froyo)	Android 2.3 (Gingerbread)	Android 2.2 (Froyo)
Architecture	Dual-Core ARM Cortex A9	Samsung-Intrnsity SSPC110	Qualcomm QSD 8250	Samsung-Hummingbird S5PC111
Number of Core	2	1	1	1
CPU Clock	1 GHz per core	1 GHz	1GHz	1GHz
L1 Cache	32 KB per core	32 KB	32 KB	32 KB
L2 Cache	1 MB	512 MB	256 KB	512 KB
Memory	1 GB	593 MB	512 MB	512 MB
GPU	ULP GeForce	None	Adreno 200	PowerVR SGX540
Main Camera	12MP	3.15 MP	5MP	3.15 MP
Usage	Tablet	Tablet	Phone	Phone

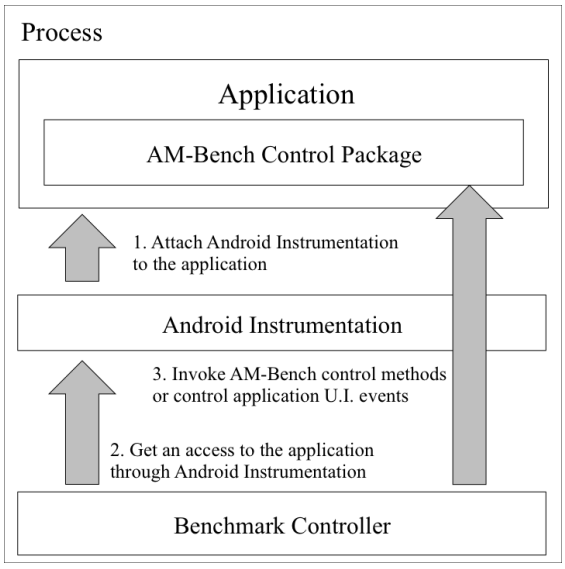

Fig. 2. Android Instrumentation in AM-Bench

3) Built-In Applications: Android contains built-in mobile applications such as Dialer, Email, Camera, and others, that are included by default. AM-Bench includes two of Android's built-in applications; Camera and Gallery.

Creating benchmarks using built-in applications is more complex than creating benchmarks using third-party applications. In order to install the modified built-in application, which has an AM-Bench control package, we had to uninstall the built-in application from profiling devices. Because Android devices required the root privilege in order to uninstall built-in applications, additional steps were required. We used SuperOneClick [11] to root Android devices.

B. Execution Time

To provide various of studies, AM-Bench includes different input sets to evaluate the execution time of multimedia tasks on Android devices as shown in Table IV.

1) **Load e-book:** We use an e-book that has a size of 2.5 MB in the MOBI format without images. FBReader loads the book from an SD card to the memory and examines the search operation. Figure 3 shows the execution time of e-book load and text search. Load e-book is a CPU intensive benchmark. As a result, NVIDIA Tegra 250 Dev. Kit shows the best performance. The importance of the memory capacity for the search operation is well captured in the figure as well. With the lowest memory capacity, Nexus One takes the longest time to finish the search operation.

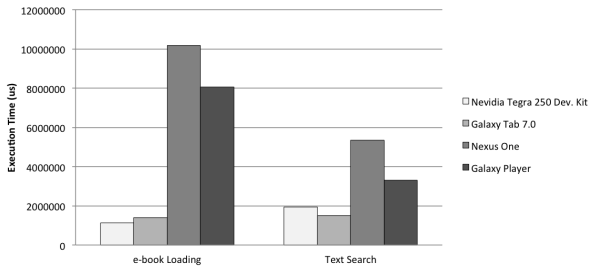

Fig. 3. FBReader Execution Time

2) **QR Code Process:** In order to measure the execution time of QR code processing, the Barcode Scanner application in AM-Bench was used. The QR code process is a computation-intensive benchmark with complex matrix operations involving threshold decision, sampling raw image pixels, finding the alignment patterns, etc. As shown in Figure 4, the benefit of having multi-core (Tegra 250 Dev. Kit) shows the best performance for the complex computation.

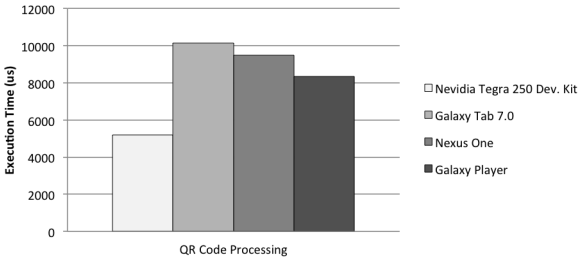

Fig. 4. QR Code Process Execution Time

3) **Matrix Operation:** Matrix operations are widely performed across the multimedia applications, and this is one of the most computation-intensive tasks. Thus, faster execution time for matrix multiplication implies faster running time for all other computation-intensive tasks. The execution time of matrix operations was evaluated by the Matrix Computation benchmark in libGDX 2D. Table V specifies the matrix operations in the Matrix Computation benchmark.

Figure 5 shows the execution time of matrix operations. The most intensive work is the Java inverse operation and we can see the benefit of using JNI in this case across profiling devices. Although NIVIDA Tegra 250 Dev. Kit shows the

TABLE V
MATRIX OPERATION INPUT SET

Name	Operation	Size	Iteration
M1	Java Matrix Multiplication	4 by 4	1,000,000
M2	JNI Matrix Multiplication	4 by 4	1,000,000
M3	Java Vector Multiplication	4 By 4	500,000
M4	JNI Vector Multiplication	4 by 4	500,000
M5	Java Matrix Inverse	4 by 4	1,000,000
M6	JNI Matrix Inverse	4 by 4	1,000,000

best performance throughout the benchmarks, Nexus One also shows also outstanding performance. The hardware configuration of Nexus One is similar to Galaxy Player; however, Nexus One takes an average of 68% of the execution time of Galaxy Player. Using JNI does not always guarantee the performance enhancement because of system overhead; therefore, application developers have to be careful of using JNI. As a result, JNI Vector and Matrix multiplication takes longer than the same operation in Java.

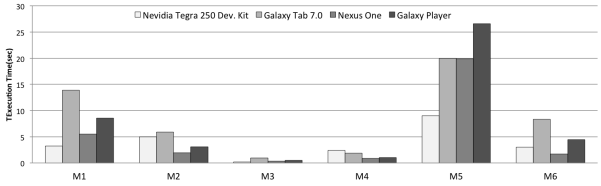

Fig. 5. Matrix Operation Execution Time

4) **2D & 3D Graphic Process:** In order to measure the ability to handle 2D and 3D graphics, various benchmarks were selected from libGDX 2D/3D to evaluate the frame rate on different computational loads. As for the 3D modeling evaluation, benchmarks measure the time taken to draw a 3D model on the screen of a mobile device with 2275 vertices and 3163 triangles. Figure 6 shows the frame rate of 2D/3D graphics. As for the full stress 2D benchmarks, all of the profiling devices report a very low FPS. In the case of 3D modeling benchmarks, the performance improvement with GPU is well illustrated in Figure 7.

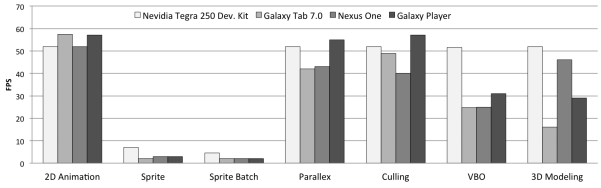

Fig. 6. Frame Rate of 2D/3D Graphics

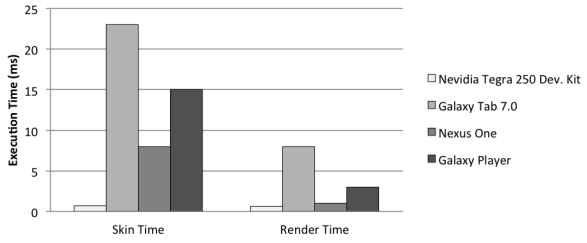

Fig. 7. 3D Modeling Execution Time

5) **Built-in Camera&Media Compression:** We used benchmarks in the Camera application to evaulate the performance of built-in cameras on mobile devices. In Table IV we can find the resolution of the built-in cameras of profiling devices. We did not include the Nexus One profiling result due to the OS version conflict. Despite similar hardware configurations, Samsung Galaxy Tab 7.0 shows an average of 1.3 times faster performance than Samsung Galaxy Player.

6) **Media Decompression:** For the media decompression benchmark, the built-in Gallery application was used. We prepared 100 different JPEG images and measured the time taken to decompress the images in the application procedure. Interesting data is captured in Figure 8. Galaxy Player spent a significantly less time than the other two devices for the initial building process. As a result, Galaxy Player spent the longest time loading thumbnail images. The initial building process involves creating meta data for the thumbnail images and storing the data into a skip list (memory). The importance of these steps is well captured in the figure. Since the media decompression process involves CPU-intensive computations, NVIDIA Tegra 250 Dev. Kit shows better performance than Galaxy Tab and Player.

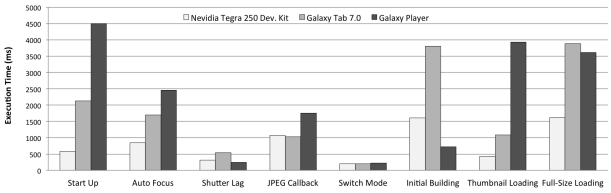

Fig. 8. Media Compression/Decompression Execution Time

C. CPU Usage

Many mobile devices these days have adopted extra hardware components such as GPU, GPS, and accelerometers, so it is not clear whether a task is done by the CPU or by other hardware components. Hence, AM-Bench provides CPU performance information to show a CPU load at a given amount of time. AM-Bench computes the CPU performance information using the proc/stat file from the Linux kernel. By default, AM-Bench does not include the file I/O work to the overall CPU performance due to the background file I/O process.

Figure 9 shows an example of CPU usage from Gallery application on Galaxy Player. Red marks indicate method calls during the benchmark process. By the nature of the Android background process, we experience irregular peaks during the benchmark process; however, we can see a constant CPU load (100% CPU usages) aligned with function calls.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we introduced AM-Bench, Android Multimedia Benchmark Suite. The benchmark suite includes four important multimedia components: media playback, media compression, computation and rendering features. In AM-Bench, from six representative applications, a total of 20

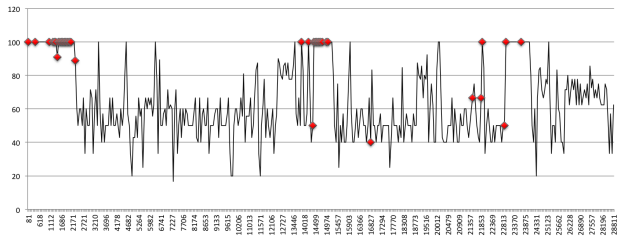

Fig. 9. Gallery CPU performance on the Galaxy Player

benchmarks are selected. All benchmarks are written in Java and have a method to evaluate several provided input sets.

Using AM-Bench, we evaluated four different mobile platforms: two tablet platforms and two Smart phones. We will release this benchmark suite into the public domain. In our future work, we will devise a method to simulate AM-Bench in architecture simulators to explore mobile architecture design options. We will also evaluate the characteristics of the benchmarks using underlying hardware performance counters.

REFERENCES

[1] "Android activity manager," http://developer.android.com/reference/android/app/ActivityManager.html.

[2] "Android ndk," http://developer.android.com/sdk/ndk/index.html.

[3] "FBReader," http://www.fbreader.org/.

[4] "Google nexus one," http://www.google.com/phone/detail/nexus-one.

[5] "Java native interface," http://java.sun.com/docs/books/jni.

[6] "libGDX," http://libgdx.badlogicgames.com/.

[7] "Mobile gps benchmark," http://www.gpsbenchmark.com/the-mobile-app1.

[8] "Nvidia tegra 2," http://www.nvidia.com/object/tegra-2.html.

[9] "Samsung galaxy s," http://www.samsung.com/global/microsite/galaxys/index_2.html.

[10] "Samsung galaxy tab 7.0," http://www.samsung.com/us/mobile/galaxy-tab/SCH-I800BKAVZW.

[11] "shortfuse.org," http://shortfuse.org/.

[12] "ZXing ("Zebra Crossing")," http://code.google.com/p/zxing/.

[13] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC Benchmark Suite: Characterization and Architectural Implications," Princeton University, Tech. Rep. TR-811-08, 2008.

[14] G. Czajkowski, L. Daynes, and M. Wolczko, "Automated and portable native code isolation," in *Software Reliability Engineering, 2001. ISSRE 2001. Proceedings. 12th International Symposium on*, nov. 2001, pp. 298 – 307.

[15] "Android Developers," http://developer.android.com, Google.

[16] "Instrumentation Testing," http://www.kandroid.org/online-pdk/guide/instrumentation_testing.html, Google.

[17] "Testing Fundamentals," http://developer.android.com/guide/topics/testing/testing_android.html/, Google.

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

[19] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on*, dec. 2001, pp. 3 – 14.

[20] A. Gutierrez, R. G. Dreslinski, T. F. Wenisch, T. Mudge, A. Saidi, C. Emmons, and N. Paver, "Full-system analysis and characterization of interactive smartphone applications," in *In IEEE International Symposium on Workload Characterization*, Nov 2011.

[21] C. hui Huang and H. Y. Chen, "A semi-automatic generator for unit testing code files based on junit," in *Systems, Man and Cybernetics, 2005 IEEE International Conference on*, vol. 1, oct. 2005, pp. 140 – 145 Vol. 1.

[22] "GLBenchmark," http://www.kandroid.org/online-pdk/guide/instrumentation_testing.html, Kishonti Information Ltd.

[23] "Jbechmark," http://www.jbenchmark.com/, Kishonti Information Ltd.

[24] M. Levy, "Evaluating digital entertainment system performance," *Computer*, vol. 38, no. 7, pp. 68 – 72, july 2005.

[25] *Welcome to SPEC*, The Standard Performance Evaluation Corporation, http://www.specbench.org/.

[26] S. Venkata, I. Ahn, D. Jeon, A. Gupta, C. Louie, S. Garcia, S. Belongie, and M. Taylor, "Sd-vbs: The san diego vision benchmark suite," in *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*, oct. 2009, pp. 55 –64.