Customization of Android and Performance Analysis of Android Applications in Different Environments

Thesis submitted in partial fulfillment of the requirements
for the award of degree of

Master of Engineering
in
Computer Science and Engineering

Submitted By
Vaibhav Kumar Sarkania
(Roll No. 801132030)

Under the supervision of:
Vinod Kumar Bhalla
Assistant Professor

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR UNIVERSITY
PATIALA – 147004

July 2013

CERTIFICATE

I hereby certify that the work which is being presented in the thesis entitled, ***"Customization of Android and Performance Analysis of Android Applications in different Environments"***, in partial fulfillment of the requirements for the award of degree of Master of Engineering in *Computer Science and Engineering* submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of ***Vinod Kumar Bhalla*** and refers other researcher's work which are duly listed in the reference section.

The matter presented in the thesis has not been submitted for award of any other degree of this or any other University.

Signature:

Vaibhav Kumar Sarkania

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

Vinod Kumar Bhalla
Assistant Professor
Computer Science and Engineering Department

Countersigned by

(Dr. Maninder Singh)
Head
Computer Science and Engineering Department
Thapar University
Patiala

(Dr. S. K. Mohapatra)
Dean (Academic Affairs)
Thapar University
Patiala

ACKNOWLEDGMENT

No volume of words is enough to express my gratitude towards my guide, ***Vinod Kumar Bhalla****, Assistant Professor, Computer Science and Engineering Department, Thapar University, who have been very concerned and have supervised the work presented in this thesis report. He has helped me to explore this vast field in an organized manner and provided me with all the ideas on how to work towards a research oriented venture.*

I am also thankful to ***Dr. Maninder Singh****, Head of Department, CSED and* ***Mr. Karun Verma****, P.G. Coordinator, for the motivation and inspiration that triggered me for the thesis work.*

I would also like to thank the staff members and my colleagues who were always there in the need of the hour and provided with all the help and facilities, which I required, for the completion of my thesis.

Most importantly, I would like to thank my ***parents****,* ***friends*** *and the* ***Almighty*** *for showing me the right direction out of the blue, to help me stay calm in the oddest of the times and keep moving even at times when there was no hope.*

Vaibhav Kumar Sarkania

(801132030)

Abstract

Android has become most popular and powerful embedded OS. Now a days it is used in other electronic items other than mobile phones like Tv, Camera, etc.Mobile phones have limited storage and battery life. Android applications and games are becoming more advance and complex resulting in using more memory and battery. Mobile phones also need more advance processors to run these high end Games and application. As the mobiles are built by different manufactuters ,they have their own user interface and pre installed applications and if more and more applications are installed the battery is consumed more and the phone becomes slow. So the purpose of this thesis is to increase the performance of Android operating system by customizing the Stock ROM and creating a new and enhanced custom ROM. Normally an android application is build using Java because its easy to use and implement .An android application can also be build using C/C++. So this study will also give a Performance Analysis of implementation of Android Application on Java and Native C/C++ on Customized ROM.

Table of Contents

List of Figures

Abbreviation

OS	Operating System
ROM	Read Only Memory
HTC	High Tech Computer Corporation
LG	Lucky-Goldstar
RAM	Random Access Memory
VM	Virtual Machine
IPC	Inter Process Communication
CPU	Central Processing Unit
IPC	Inter Process Communication
JVM	Java Virtual Machine
JEE	Java Enterprise Edition
JSE	Java Standard Edition
XML	eXtra Markup Language
LED	Light Emitting Diode
API	Application Program Interface
NDK	Native Development Kit
AVD	Android Virtual Device
SDK	Software Development Kit

Chapter 1
Introduction

This chapter includes basic introduction of Android OS and its history. It also consists types of Android ROMs, how Android O.S Boot and difference between Android and Linux.

1.1 Background of Android

Android is a Linux based mobile operating system not Linux, it uses Linux kernel that means commands, directory structure and many things are similar to the Linux. Android is an open source operating system and Google releases the code under the Apache license and allows software to be easily modified and distributed by device manufacturers, wireless carriers and developers. Android Incorporation was founded in Palo Alto, California, United States in October, 2003 by Andy Rubin and Rich miner. From starting Android Incorporation operated secretly, expose only that it was working on mobile software's [1].

Google took over Android Incorporation in August 2005, making Android Incorporation an entire owned property of Google Incorporation. Main employees of Android Incorporation, including Andy Rubin, Rich Miner and Chris White, stayed at the company after the possession of Google. Not much was known about Android Incorporation at the time of the acquisition, but people conclude that Google was planning to penetrate the mobile phone market with their weapon i.e. Android [7].
On November 5, 2007, the Open Handset Alliance, a bunch of several companies which include Broadcom Corporation, Google, HTC, Intel, LG, Marvell Technology Group, Motorola, Nvidia, Qualcomm, Samsung Electronics, Sprint Nextel, T-Mobile and Texas Instruments exposed themselves in front of media itself. The aim of the Open Handset Alliance is to develop open standards for mobile devices. On 9 December, 2008 14 new members accompany, including ARM Holdings, Asustek Computer Inc, Garmin Ltd, Huawei Technologies, Packet Video, Atheros Communications, Vodafone, Sony Ericsson, Toshiba Corp. On 23 September, 2008 the first Android device is launched, that is HTC Dream G1 which operates Android

1.0,and after that android shows 1.1 update which was released for T-Mobile G1 only [1].

Different Android versions

- Android 1.5 (Cupcake)
- Android 1.6 (Donut)
- Android 2.0 (Éclair)
- Android 2.2 (Froyo)
- Android 2.3 (Gingerbread)
- Android 3.0 (Honeycomb)
- Android 4.0 (Ice-cream sandwich)
- Android 4.1 (Jellybean)

1.2 Android ROM

ROM is read only memory and refers to the internal memory of a device where operating system instructions and applications are stored.
Three types of ROMs available for android

i. Truly Stock ROMs / firmware: This is the operating system in its default form, without any modifications made to it except for any device-specific support required to run it on the particular device. Truly stock firmware provides the standard user experience of the operating system without any cosmetic or functional changes made. These days, truly stock firmware is primarily found in cases where both the device and the operating system is built by the same company. Amongst modern mobile devices, examples of truly stock firmware can be found on Apple's iOS devices, Palm's WebOS devices and some Android devices shipped without any modifications made to the operating system by their manufacturers [8].

ii. Manufacturer or Carrier branded Stock ROM / Firmware: This type of firmware has had enhancements added over the default operating system by the device manufacturer or the mobile service carrier. This often includes interface enhancements, proprietary applications and in most cases, restrictions intended to limit the use of the device with a specific carrier or region. There are often further restrictions preventing installation of firmware not released by the carrier or

manufacturer. Most Android and Symbian devices fall under this category and so do most Windows Phone 7 devices but in their case, the changes made from the truly stock firmware are minimal and limited to the inclusion of additional apps only [9].

iii. Custom ROM / firmware: Almost all devices ship with either of the above two categories of firmware, though things don't end there. Independent developers who like to customize their devices beyond the standard options provided often tend to release the fruits of their labor for the rest to enjoy, in form of custom ROMs. The more open the platform, the more independent development it attracts, a good example of which is the independent custom ROM development for Android [9].In case of proprietary firmware such as iOS and Windows Phone 7, there is often little or no room for customization of the operating system itself but regardless of that, developers still tend to release custom ROMs bundled with useful tools and hacks applied to provide functionality beyond the stock features. In fact custom ROM development for the otherwise proprietary and closed-source Windows Mobile platform lead to the formation of the largest independent mobile development community [8].

1.3 Android Boot up Process

1.3.1 Power on

Master boot record (MBR) is a boot sector which contains partition table which has the information about how the device is partitioned in a structure. There is no MBR or partition when the device is started for the first time. When the phone is switched on, CPU will be in a no initialization state. Internal RAM is available and no internal clocks are set up. The device starts executing code located in the ROM and finds a specific block which has first Stage boot loader. The first boot loader points to a second stage boot loader, which is located in a known block. This "pointing" process is called raw partition table [10].

1.3.2 Bootloader

Boot loader is a code which is executed before android operating system runs. It loads kernel to the RAM and sets up the initial memories. Manufacturers use existing boot loaders or they create their own boot loaders.

- The First stage boot loader will find and setup the external RAM.
- Now Main boot loader is loaded and placed in external RAM.
- The First important program is in the second boot loader stage which contains code for file systems, additional memory and network support etc.
- When the boot loader is done it goes to the Linux kernel [11].

1.3.3 Linux Kernel

A kernel acts as a bridge between hardware and software. It setups cache protected memory, scheduling and loads drivers. After initializing Memory management units and caches, virtual memory can be used and user space processes can be launched by the system. After finishing the setup Kernel looks for init process which can found under system/core/init and launch it [5].

1.3.4 Init Process

This process is the root process. Every process will be launched from this process. Init process mounts directories like /sys, /dev, /proc. It will run init.rc script and system service processes. This script is located in system/core/root dir in the Android open source project and describes system services, file system and other parameters [7].

1.3.5 Zygote

After starting various daemons like Android Debug Bridge (adb), Radio Interface Layer Daemon (rild), etc, Init process initiates a process called Zygote. In java there is a separate instance of a Virtual Machine for each application. In android Dalvik, virtual machine is used as VM. So there is high consumption of memory and time because of different instances of Dalvik VM for every application [11]. Now Zygotes comes into play. It enables shared code across Dalvik VM, lower memory footprint and minimal start-up time. Zygote process starts at system boot up and it preloads and initializes core library classes. After initialization, zygote process waits for socket request coming from the runtime process. If any request comes then it forks starts processes with VM instances [10].

1.3.6 Runtime Process

The next init initiates the Runtime process and this process starts the service manager. All the services should be registered with the service manager and it provides local lookup service and binds services given their name. The runtime requests zygote to start system server process [7]. Zygote splits and starts up a new Dalvik vm instance and starts the service. To control display device and audio output device the system server starts surface flinger and audio flinger. These services get registered with the service manager so that other applications can use display and audio. Now the system server will start all the core platform services and hardware services like activity manager, window manager and power manager etc. All of these services will get registered with the service manager [11].

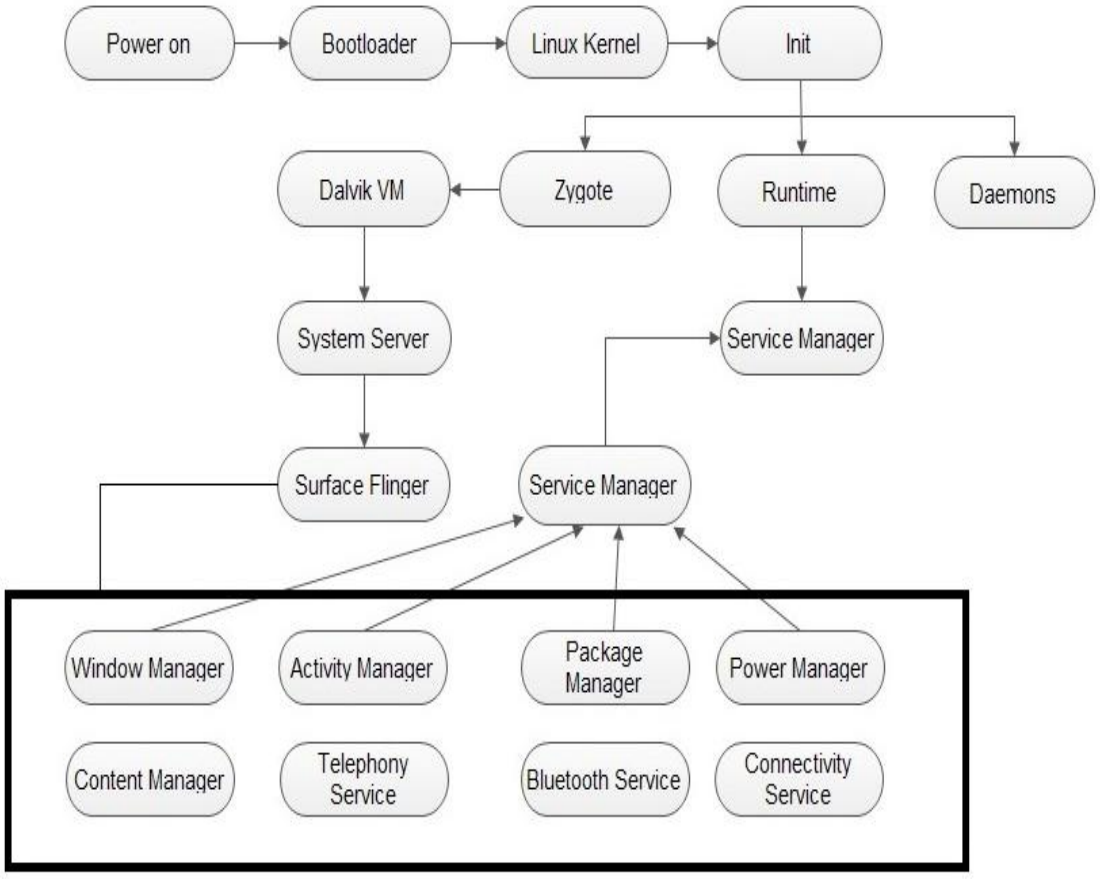

Figure 1.1 Boot Up Process

1.3.7 Inter Process Communication

At this point home screen or idle screen is launched. Activity manager will send a request to zygote to initiate the “home” activity and in return Zygote will fork a new process a Dalvik VM and home activity. Now for each application launched by the user, Zygote will fork each time and create a new Dalvik VM instance inside a new process. A unique user id is assigned to each application. An application has access to only those files which it needs and these permissions are set up by the system [12].

Applications run in separate processes, so to communicate with each other and with the system services an IPC (Inter Process Communication) is needed and this mechanism in android is known as Binder and is based on shared memory. On registration of each process with the service manager, it gets a reference called a context object. Let there be an application A and service B which is running in spate processes and wants to communicate with each other application. A passes the name of the service to context and requests for service. B in return context sends a reference to the service to A [11].

After getting the reference app, A calls a method which is intercepted by the Binder which arranges the object and passes the reference to Receiver. The object is serialized because a proxy object is passed and not the original objects. There is a thread pool maintained by the binder at Receiver B side and one of the threads receives the incoming call, locates the actual object and makes the call. Return value is passed back to the application A [12].

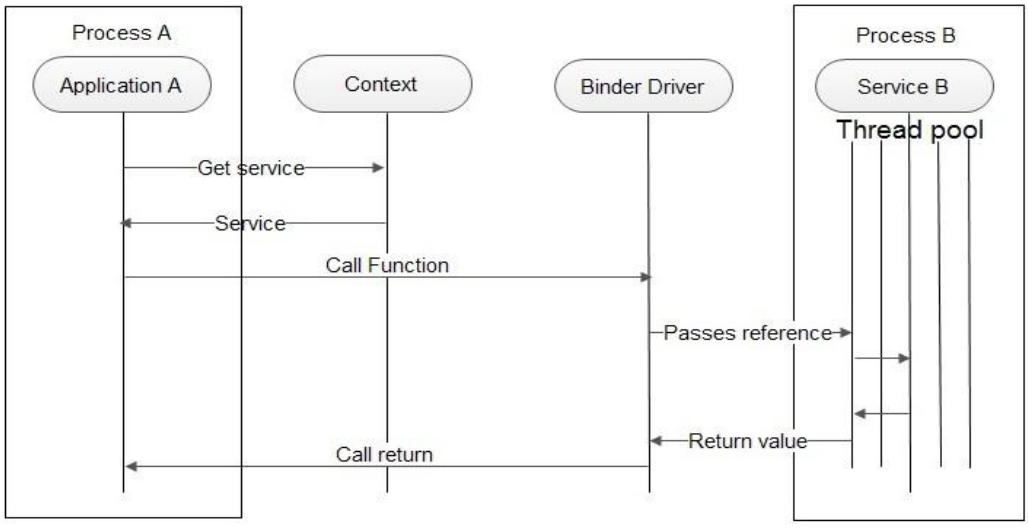

Figure 1.2 Inter Process Communication

1.4 Android vs. Linux

First of all, the Android kernel was derived from Linux, but has been significantly altered outside the mainline Linux kernel distribution. To further illustrate that point, Android is neither equipped with a native X-Windows setup, nor does it support the full set of standard GNU libraries. Hence, it is a daunting task to port any existing GNU/Linux application or library to Android (support for X-Windows would be possible in Android though). The biggest difference between Linux and Android revolves around the Java abstraction layer embedded into Android. The Android design is based on a deeper implementation stack than Linux. In other words, the Android applications are farther removed from the actual kernel than in Linux (have a longer code path down into the OS layer). The core of Linux applications are developed in C and C++, hence C and C++ code represents the predominant Linux application environment. In Linux, the user applications (via the libraries and the system call subsystem) have direct kernel access, not so with Android [13].

In Android, the kernel is almost hidden deep inside the Android operating environment. Under Linux, the make process for (C, C++) applications can directly be optimized via special compiler flags, further boosting application performance [7]. Further, the Linux operating setup natively incorporates a very rich infrastructure of libraries, debuggers, and development tools that are not accessible by Android. While the Android design is based on a deeper implementation stack, and hence the applications are farther removed from the kernel compared to Linux, Android kernel performance is still important and has to be quantified and understood. As in Linux, aggregate application performance is still impacted by the efficiency of the implemented kernel primitives [13].

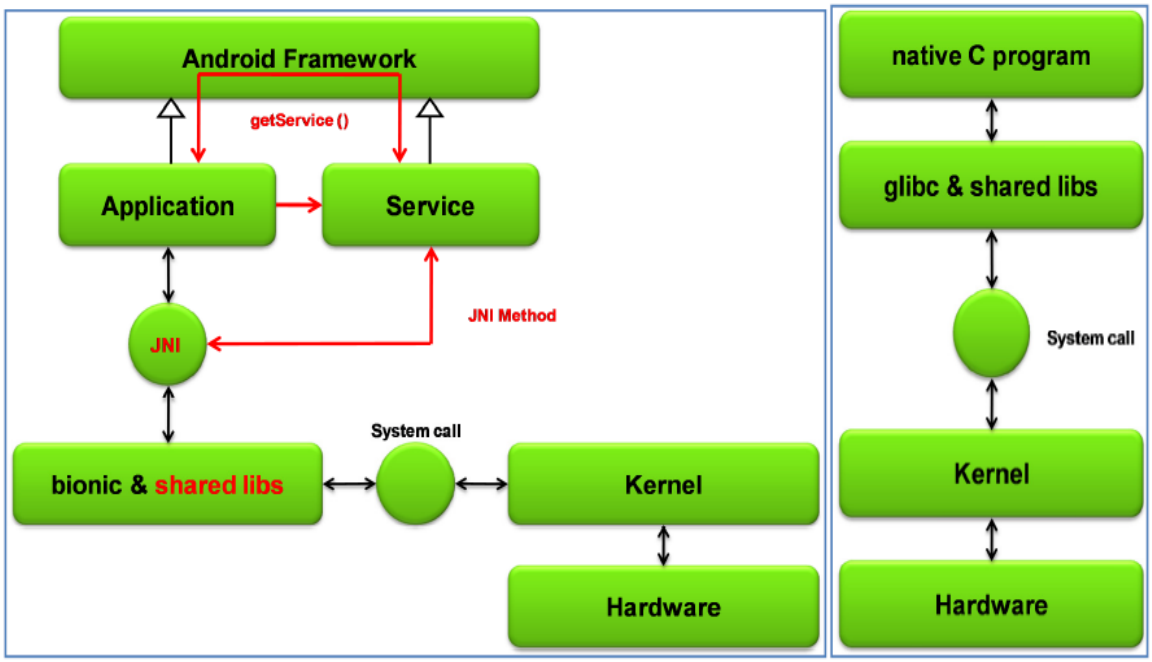

Figure 1.3 Differences between Android and Linux [13]

Chapter 2

Literature Survey

This chapter explains Android Architecture and its layers .It also define the components and whole life cycle of an Android Application.

2.1 Android Architecture

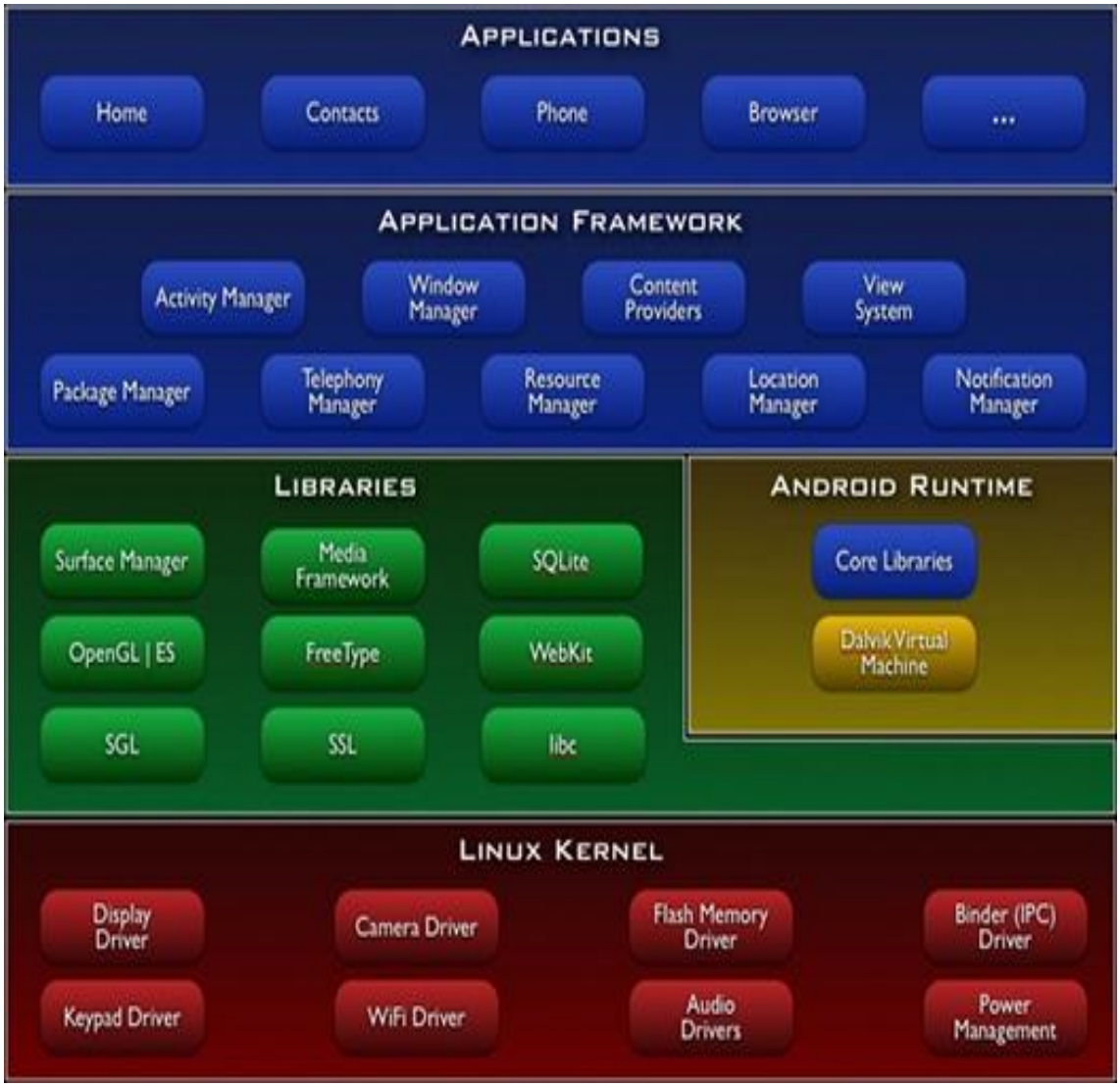

Figure 2.1: Android Architecture [14]

The Android software stack as shown in figure can be subdivided into five layers: The kernel and low level tools, native libraries, the Android Runtime, the framework layer and on top of all the applications.

2.1.1 Linux Kernel

The Android system architecture comprise of four layers. The lowest of all is Linux kernel layer, used as an abstraction between hardware and the remaining software stack of Android. The kernel used in Android is a 2.6 series Linux kernel modified to fulfil some special needs of the platform. The kernel is mostly extended by drivers, power management facilities and adjustments to the limited capabilities of Android's target platforms. The power management capabilities are crucial on mobile devices, thus the most important changes can be found in this area. Like the rest of Android, the kernel is freely available and the development process is visible through the public Android source repository [15].

Differences to mainline

The changes to the mainline kernel can be categorized into: bug fixes, facilities to enhance user space (low memory killer, binder, ashmem, logger, etc.), new infrastructure (esp. wake locks) and support for new SoCs (msm7k, msm8k, etc.) an boards/devices [15]. The Android specific kernels and the mainline Linux kernel are supposed to get merged in the future, but this process is slow and will take some time [16].

i. Wake locks: Android allows user space applications and therefore applications running in the Dalvik VM to prevent the system from entering a sleep or suspend state. This is important because by default, Android tries to put the system into a sleep or better a suspend mode as soon as possible. Screen stays on or the CPU stays awake to react quickly to interrupts. The means Android provides for this task are wake locks. Wake locks can be obtained by kernel components or by user space processes. The user space interface to create a wake lock is the file /sys/power/wake lock in which the name of the new wake lock is written. To release a wake lock, the holding process writes the name in /sys/power/wake unlock. The wake lock can be furnished

with a timeout to specify the time until the wake lock will be released automatically. All by the system currently used wake locks are listed in /proc/wake locks [17].

ii. Power manager: The Power Manager is a service class in the application framework that gives Dalvik VM applications access to the WakeLock capabilities of the kernel power management driver. Four different kinds of wake locks are provided by the Power- Manager

Partial Wake Lock: The CPU stays awake, even if the device's power button is pressed.
Screen Dim Wake Lock: The screen stays on, but is dimmed.
Screen Bright Wake Lock: The screen stays on with normal brightness
Full Wake Lock: Keyboard and screen stay on with normal back light.

Only the Partial Wake Lock assures that the CPU is fully on, the other three types allow the CPU to sleep after the device's power button is pressed. Like kernel space wake locks, the locks provided by the Power Manager can be combined with a timeout. It is also possible to wake up the device when the wake lock is acquired or turn on the screen at release [4].

iii. Memory management: The memory management related changes of the kernel aim for improved memory usage in systems with a small amount of RAM. Both ashmem and pmem add a new way of allocating memory to the kernel. Ashmem can be used for allocations of shared memory and pmem allows allocations of contiguous memory.

ASHMEM: The Anonymous/Android Shared Memory provides a named memory block that can be shared across multiple processes. Other than the usual shared memory, the anonymous shared memory can be freed by the kernel. To use ashmem, a process opens /dev/ashmem and performs mmap( ) on it.

PMEM: Physical Memory enables e.g. drivers and libraries to allocate named physically contiguous memory blocks. This driver was written to compensate hardware limitations of a specific SoC – the MSM7201A [18].

Low memory killer: The standard Linux kernel out of memory killer (oom killer) utilizes heuristics and compute the process' "badness" to be able to terminate the process with the highest score in a low memory situation. This behaviour can be inconvenient to the user as the oom killer may close the user's current application when there are other processes in the system, that do not affect the user [4].

iv. Other changes

In addition to the already described kernel changes, there are various other changes that touch miscellaneous areas of the kernel. A few minor changes are described in this section [18].

Binder : Unlike in the standard Linux kernel, the IPC mechanism in Android's kernel is not System V compatible. The user space side of the IPC mechanism and the underlying concept. The kernel side of the mechanism is based on OpenBinder thus focused on being light weight. In order to enhance performance, the binder driver uses shared memory to pass the messages between threads and processes [4].

Logger : Extended kernel logging facility with the four logging classes: main, system, event and radio. The application framework uses the system class for its log entries.

Early suspend : Drivers can make use of this capability to do necessary work, if a user space program wants the system to enter or leave a suspend state[19].

Alarm : The alarm interface is similar to the hrtimer interface but adds support for wake up from suspend. It also adds an elapsed real-time clock that can be used for periodic timers that need to keep running while the system is suspended and not be disrupted when the wall time is set. [18]

2.1.2 Libraries

The native libraries of Android are written in C/C++, used by various components of Android.

- Surface Manager support Android for composing various drawing surfaces on to the mobile screen. The seamless composition of 2D and 3D graphics layers through multiple applications is managed by surface manager.

- OpenGL ES 1.0 APIs are use to implement 3D libraries, 3D graphics are created (by using such libraries) either by hardware 3D accelerator or through software base rasterization, depending upon the availability of resource.
- SGL is core for 2d graphic libraries.
- The Packet Video libraries support multi-media services, such as playback and recording for all media files, including static images.
- Rendering bitmap images and vector fonts supported by Free Type libraries.
- The SSL libraries are available in Android's native libraries stack to ensure the privacy and integrity of data transmission between web server and your mobile device browser.
- SQLite is relational database management system use to manage databases for each application of Android.
- WebKit is an integrated open source web browser, same as in safari and Apple with subtle modification [20].

2.1.3 Android Runtime

Along with native libraries, the Android runtime is on second layer right above the Linux kernel. The Android runtime consist of Dalvik virtual machine and some core libraries.

Dalvik Virtual Machine: Java has always been marketed as "write once, run anywhere." The capability has largely been made possible by the Java Platform, the foundation of which is the Java Virtual Machine (JVM).Although this goal has largely been met for the Java platform on desktop (JSE) and server (JEE) environments, the mobile Java ecosystem (JME) is a bit more fragmented with various configurations, profiles, and packages causing significant modifications to applications in order to support different devices.

Every Android application runs in its own process, with its own instance of the Dalvik virtual machine. Dalvik has been written so that a device can run multiple VMs efficiently. The Dalvik VM executes files in the Dalvik Executable (.dex) format which is optimized for minimal memory footprint. The VM is register-based, and runs classes compiled by a Java language compiler that have been transformed into the .dex format by the included "dx"tool. The Dalvik VM relies on the Linux kernel for

underlying functionality such as threading and low-level memory management. Given every application runs in its own process within its own virtual machine, not only must the running of multiple VMs be efficient but creation of new VMs must be fast as well [3].

Zygote: Since every application runs in its own instance of the VM, VM instances must be able to start quickly when a new application is launched and the memory footprint of the VM must be minimal. Android uses a concept called the Zygote to enable both sharing of code across VM instances and to provide fast start-up time of new VM instances. The Zygote design assumes that there are a significant number of core library classes and corresponding heap structures that are used across many applications. It also assumes that these heap structures are generally read-only. In other words, this is data and classes that most applications use but never modify. These characteristics are exploited to optimize sharing of this memory across processes [3].

The Zygote is a VM process that starts at system boot time. When the Zygote process starts, it initializes a Dalvik VM, which preloads and pre-initializes core library classes. Generally these core library classes are read-only and are therefore a good candidate for preloading and sharing across processes. Once the Zygote has initialized, it will sit and wait for socket requests coming from the runtime process indicating that it should fork new VM instances based on the Zygote VM instance. Cold starting virtual machines notoriously takes a long time and can be an impediment to isolating each application in its own VM. By spawning new VM processes from the Zygote, the start-up time is minimized. The core library classes that are shared across the VM instances are generally only read, but not written, by applications. When those classes are written to, the memory from the shared Zygote process is copied to the forked child process of the application's VM and written to there. This "copy-on-write" behaviour allows for maximum sharing of memory while still prohibiting applications from interfering with each other and providing security across application and process boundaries. In traditional Java VM design, each instance of the VM will have an entire copy of the core library class files and any associated heap objects. Memory is not shared across instances [21].

2.1.4 Application Framework

Application Framework is on third layer going from bottom to top. It is basically a built-in toolkit, use to provide different set of services to Android applications. All those services which utilizes by core applications are make available for the Android developers to build innovative and rich Android applications. The application architecture is designed to simplify the reuse of components, any application can publish its capabilities and any other application may then make use of those capabilities (subject to security constraints enforced by the framework) through underlying components of application framework layer. Brief description of some components is discussed here [20].

- The Activity Manager manages the lifecycle of the applications and provides a common navigation back stack of applications that are running in different processes.
- The Package Manager maintain track of all applications that are installed in the device.
- The Windows Manager manages screen of mobile device and creates surfaces (an allocated memory block) for all running applications.
- The Telephony Manager support applications to access the information regarding telephony services. Access to some telephony information is protected and some applications may not have permission to access such protected information (Constrain to permissions declared in its manifest File).
- Content Providers supports the sharing and accessing of data among applications suppose the messaging service is an application that can access the data of other application contacts.
- In Android the term resources refers to the non code external assets. The Resource Manager provides access to such external resources to native Android application at built time. Android supports a number of different kinds of resource Files, including XML (use to store anything, other than bitmaps and Raw), Bitmap (Use to store images), and Raw Files (other resources such as sound, string, etc).
- The development of new Android applications requires access to views of previously built application. The View System provides access to some

extensible set of views, such as lists, text boxes, grids, embedded browser, and button.

- Notification Manager allows application to notify users about events that take place. The notification can be inform of vibration, Flashing LEDs of mobile device, playing specific sound against specific event, or could be an icon (persistently displayed into the status bar)[20].

2.1.5 Application Layer

Application is a set of core applications both native and third party that are built on the application layer by the means of same API libraries. The Application layer runs within the Android runtime, using the classes and services made available from the application framework.

2.2 Application Fundamentals

2.2.1 Context and Packaging

Android applications run in a rather isolated context. When an application is started it begins to live in its own world in many respects:

- By default, every application runs in its own Linux process
- each process has its own Dalvik VM
- Each application gets a unique Linux user ID by default. Meaning the application's files are only visible and accessible by this user

Though it is possible for two applications to share the same user ID, this is not the norm. To preserve system resources, applications with the same user ID may also run in the same process and share the same Dalvik VM [21].

Android applications written in the Java programming languages are compiled and then along with all data and resources required by the application are bundled into an Android package (.apk) file by the Android Asset Packaging Tool (aapt). As an apk file, the application can be distributed and installed on different devices, it's the file that get's downloaded from the Android Market when you install a new application. One apk file contains exactly one application.

There are three possible ways to install an Android Package on a device:

- by publishing and downloading it from the Android Market
- by using the Android Debug Bridge (adb)
- by copying it on the SD card and using another app to install it (many file managers offer this feature)

2.2.2 Components

Android is designed in a component-based way, which is a central feature: Any one application may use any other application or part of another application, provided the other application permits it. Instead of programming your own Contacts list view you can reuse the one provided by the Android framework. Or you could write your own Contacts list view and permit other applications to reuse your implementation (without having to link against your application or incorporating the code from your application). Android applications do not have a static entry point like a main() function since this would not work well with the component-based approach. Instead, to start an application or a component of an application, a message is sent (called Intent in Android) to the framework, which will locate instantiate and start the corresponding component based on some criteria that may be packaged into the message. Android ensures that, whenever there is a request to start a specific component, that component is running (launching it if necessary) and an instance of that component is available (instantiating it if necessary) to be used by the requester[22].
Android has four types of basic components that will be described in the following sections.

i. Activities: An Activity represents a graphical user interface catered to allow the execution of a single action by the user. Although activities work together to present a seamless user experience, each Activity is independent of the others. An Activity must be derived from the base class Activity or a subclass of it. An application may consist of multiple Activities or only a single one, depending on the complexity of the task and the design of the application. One of the Activities is typically marked as the entry Activity which will be presented when the application is launched. Navigating through Activities is done by starting another Activity from the current one by

sending an intent to the framework [7]. This will build an Activity stack pushing and popping Activities in the way the user navigates through the app. Activities are given a window to draw in that fills the screen by default, but there may be smaller screens floating on top of others, in which case the Activity below will still be visible in the background. Pop-up dialogs are a good example of this design. The visual content of the windows is provided by a hierarchy of Views .Objects derived from the View base class that manage the rectangular space they're assigned to. Parent views hold and organise the layout of their children, passing any draw call to them as well. Views are at the heart of user interaction and Android already comes with several view components that can be used, like buttons, text fields, check boxes and more [22].

ii. Service: A service is an application without a graphical user interface. It runs silently in the background either to do some work or idling while waiting for requests by other components. Every service needs to be inherited from the Service base class. Services should be used to execute computationally intensive tasks to keep the user interface responsive or run tasks that don't need the user's attention like playing music (the GUI to control the music playback, on the other hand, should be a regular visible Activity). Services can either be used by simply telling the framework to start them or by binding to them, which means getting a reference to use the services' interface. The framework will take care of launching the service if it isn't already running. Services, like every other component, are started in the applications main thread by default. In order to avoid blocking the user interface, services should be started in another thread [22].

iii. Broadcast Receiver: A Broadcast Receiver is a component that listens to and receives broadcasted announcements (mostly by the system) and reacts to those according to its programming. Broadcast receivers must inherit from the Broadcast Receiver base class [7]. Applications can have any number of Broadcast Receivers which respond to any announcement the application considers important. A Broadcast Receiver does not display an user interface, but may use any other component to respond to an announcement, like starting an Activity or using the Notification Manager. Broadcast Receivers are not meant to deal with tasks them self, but rather to listen to events and start an Activity or Service that will deal with the event in their stead [22].

iv. Content Providers: A Content Provider makes an application's data or a certain set of the application's data available to other applications, thus allowing data to be shared amongst all applications. It must extend the Content Provider base class. Other applications use a Content Resolver object to access the data ordered by any one of the Content Providers, utilising Android's inter process communication mechanisms to do so. There are already some built-in Content Providers, for example to access the contacts or media data of the device [22].

2.2.3 Launching and shutdown

Activities, Services and Broadcast Receivers are activated by an asynchronous message called Intent, which is an object of the Intent class. For Activities and Services, the Intent object names the action being requested and the data to be acted on by the receiver. For Broadcast Receivers, it only names the event being announced.

i. Launching an Activity

An Activity can be launched (or given something new to do) from another Activity using Intents in two ways:

- It can be simply started with the Context.startActivity( ) method, which immediately presents the new Activity on top of the current one.
- It can be started for a result with Activity.startActivityForResult( ) which immediately presents the new Activity, like startActivity() does, but also gives feedback to the initiating Activity after the launched Activity has finished. A return value may be passed back in an Intent object which can be extracted in the automatically called onActivityResult( ) method of the initiating Activity.

Either way, the newly started Activity can fetch the Intent that launched it by calling its getIntent( ) method. This can be used to pass parameters to the new Activity. Also, when an Activity is launched by an Intent, its onNewIntent( ) method gets called by the framework. This hook allows to react to newly received Intents on already instantiated Activities [4].

ii. Launching a Service

A Service is started by calling Context.startService( ). Android will call the Service's onStart( ) method in correspondence to that, passing it the Intent object that started the Service. To exert control over the Service, an Activity has to be bound to the Service with the bindService( ) method, which establishes an ongoing connection between the Activity and the Service in the form of a reference. The onBind( ) method is called on the bound Service in correspondence to this, with the Intent object as one of its parameters. bindService( ) may also start the Service if it's not already running.

iii. Initiating a Broadcast

A broadcast can be initiated using an Intent object by calling one of the following asynchronous Context methods:

- send Broadcast( ), which simply sends the Intent to all interested Broadcast Receivers
- sendOrderedBroadcast( ) delivers the Intent one at a time to allow prioritiszation
- sendStickyBroadcast( ) which performs a regular sendBroadcast( ) with the Intent "staying around" after the broadcast is complete, so that others can instantly retrieve that last Intent when calling registerReceiver( ) rather than waiting for the next broadcast to be initiated Subsequently, Android will call the onReceive( ) method on all interested Broadcast Receivers, passing the broadcasted Intent alongside the call[2].

iv. Shutting Down Components

Memory and pro cess management is handled exclusively by the Android runtime. Because of this, Activities and Services may remain running for quite some time before they are shut down. Although Android provides methods to indicate that it should properly shut down an Activity or Service for when it fulfilled its task, this will give no guarantee of when or even if it will actually happen.

An Activity can be closed by calling the finish ( ) method. Activities that were started for a result must return a resulting intent object by calling setResult( ) before closing it. Also, an Activity can close its Sub-Activities – denoting Activities it started using

the startActivityForResult( ) – by calling finishActivity( ). Services, on the other hand, are not closed (since they are not visible) but stopped by calling its stopSelf( ) method or by calling Context.stopService( ). Closing does not mean shutting down the component, but it can lead to the runtime just doing so. Components might be shut down by the Android system when they are no longer needed or the need to free resources arises – which will cause a component to be shut down prematurely [1].

v. The Manifest File

The manifest file tells the Android system about the existence of the application and define other metadata about the application, which includes an entry for every component the application contains, such as Activities, so the framework knows what components there are. The component entries are used to specify additional settings and properties for the components, which include if an Activity should be started when the application gets selected in the Application Launcher the screen listing all applications that can be launched on the device. This manifest file will also be automatically packaged into the Android Package (apk) file. It is a structured XML file that is always named AndroidManifest.xml and must be located at the root of the hierarchy [7].

vi. Intent Filters

Intent filters are part of the Android manifest file and define to what kind of Intents the components of an application listen to. These filters can be as specific as to say "activate Activity/Service XYZ" or as coarse as "open a web view window ".Intent filters allow you to specify certain criteria like a type of action, category or even data that the component can act on. Android will try to locate the best application to handle the Intent by comparing the Intent object with the Intent filters and selecting the best fit. When there are multiple applications installed that fit the criteria Android automatically presents a list and lets the user select his preferred application to use. This even allows replacing built-in Android applications that come bundled with the system, like the email or phone app [4].

2.2.4 Activity Lifecycle

Activity states

Activities can be seen as website. Just as a website contains multiple web pages, an android app contains multiple activities. One webpage can redirect to another page and so on. In an android app one activity can redirect to another and so on. All the handling activities are done by activity Manager. Its task is to create, destroy and manage activities [7].

i. Active/Running: An activity is said to be in running state if it is completely visible and the user can interact with it. There can be only one running activity at a given time.

ii. Paused State: An activity is said to be in paused state if it is not in focus but partially visible. For example while using an app on android if any notification or a dialog box appears then the activity of the app goes in paused state. While in paused state, an activity still maintains all states. It remains attached to the window manager and it can be killed by OS under low memory.

iii. Stopped: An activity is said to be in 'stopped state' when it is not at all visible on screen but it is still alive and maintains all states. To fulfil the resource requirements of higher priority activities, it can be killed by OS.

iv. Destroyed state : An activity is said to be destroyed when it is no longer in memory. OS destroys an activity after a 'paused' or 'stopped state' to free the resources [7].

Activity Lifecycle Methods

i. Oncreate( ) : When an Activity is launched the first method called is the oncreate( ) method All the User Interface creation and initialization of data elements is done in this method. A bundle object as parameter is passed in this method to restore the UI state [7].

ii. Onstart( ): Just before an Activity is being visible to the user, Onstart() method is called and if there is any task needed to be performed just before the Activity becomes visible to the user, it is defined in this method as ramping up frame rates [4].

iii. OnResume( ): Now the activity is in 'active' or 'running state' and user can interact with the activity. This method can also be called when the activity is in 'paused state' to make it in 'running state' [23].

iv. Onpause( ) : When an activity is in running state, a user might navigate to other parts of the system or the system is about to put an activity in background then Onpause( ) method is called. After this an activity can be killed by the system or Onresume( ) method can be called to resume an activity [7].

v. Onstop( ): When an Activity is not visible to the user, Onstop method is called and the application can be killed at anytime by the system in case of low memory. After this, an activity is either restarted or destroyed [23].

vi. Onrestart( ): In 'stopped state' an activity can't be resumed but it can be restarted by calling Onrestart method and it is always followed by onstart method.

vii. Ondestroy( ) : This method is called just before an activity is destroyed because the activity has finished or system kills it to save space. This is the last method called on an Activity [7].

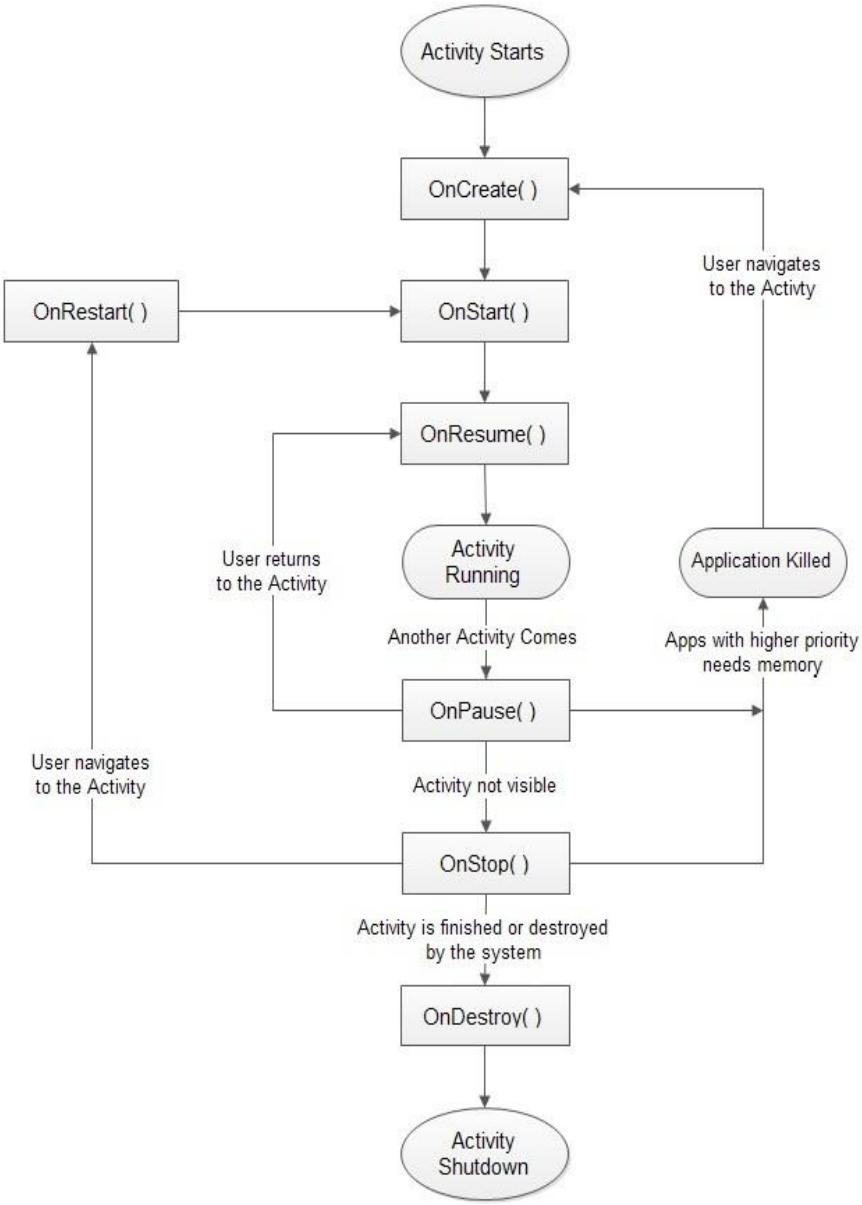

Figure 2.3: Android Activity Lifecycle

Chapter 3

Problem Statement

3.1 Problem Statement

Android Custom Rom is relatively new and few related studies has been done. By default android phones are loaded with stock ROM and the users are given guest privileges. Different manufactures have different user interface and many services running in the background in Android OS. Services and preinstalled Applications are decided by the manufacturer. And these preinstalled services and Applications consume so much memory and battery of mobile devices. So to modify a Stock Rom ,Root access is needed. So a ROM is needed which can be customized to get the root access to the phones and many background services which are of no use can be stopped to increase the performance of an Android phone. Further a Rom can be more customized to specific needs of the users.

A few related work has been done on NDK, some indicating performance benefits using NDK. Since the old experiments have been done using Android emulator or low performance Android devices.A different Environment is needed to test Java and NDK Perfromance So these Applications can be tested on Custom ROMs.

Chapter 4
Proposed Work

This chapter contains the details of the work and the implementation that has been done to meet all the thesis objectives.

4.1 Developing Android Applications

Android applications are written in Java and by the most part the standard Java API is supported by the Dalvik Virtual Machine. The Android SDK comes with all tools and APIs necessary to write Android apps. For developers that already have experience with Java, the transition is smooth and easy. Developing for Android is, by the most parts, like developing regular Java applications, though you have to keep a more limited hardware platform in mind.

4.1.1 Software Needed

The following order of installation is recommended:

- Java Development Kit
- Eclipse
- Android SDK and NDK
- Android Development Tools

Installing the JDK and Eclipse: Installing the JDK and eclipse is easy. Both can be easily downloaded from the internet.

Installing the Android SDK and NDK: The Android SDK and NDK comes bundled in a ZIP package.

i. Unzip the package to a desired output folder.

ii. Start the Android SDK and AVD Manager.

- On Windows, run the “SDK Setup.exe”.

- MacOS and Linux users need to execute the “android” executable in the tools\ subfolder.

iii. Click the Available Packages option on the left panel.
iv. Select the SDK Platform versions you wish to install. It doesn’t hurt to include the Documentation as well.
v. Click Install Selected.
vi. Install NDK plugins
vii. Set path for NDK.
viii. Right click on an Android project and select Android Tools -> Add native support [24].

Installing the Android Development Tools: The Android Development Tools offer some compelling tools for Android development, integrating the development tools, an emulator and a .class to .dex converter into the Eclipse IDE [4].

i. In Eclipse, select Help > Install New Software....
ii. Click Add on the right side to add the following site: https://dl-ssl.google.com/android/eclipse/
iii. Eclipse will now search the site for suitable Eclipse plug-ins. Check Developer Tools and then click next.
iv. An Install Details screen is shown. Ensure that both, Android DDMS and the Android Development Tools are selected and click next once again.
v. Read and accept the terms of the license agreement. Select Finish to start the installation process.

vi. After the installation is complete, restart Eclipse.
vii. Go to Window > Preferences.
viii. Select the Android entry on the left side and enter the location to the Android SDK. Confirm it by clicking OK.

Setting up a Virtual Device: An Emulator is needed to run, debug and test apps on. Though it is possible to use a real device for this, it’s easier and more comfortable with an AVD. Setting an AVD up involves the following steps:

- Start the Android SDK and AVD Manager, either from Eclipse.
- Select Virtual Devices in the left pane.
- Click New... to open the creation dialog.
- Configure the virtual device and click creates AVD [24].

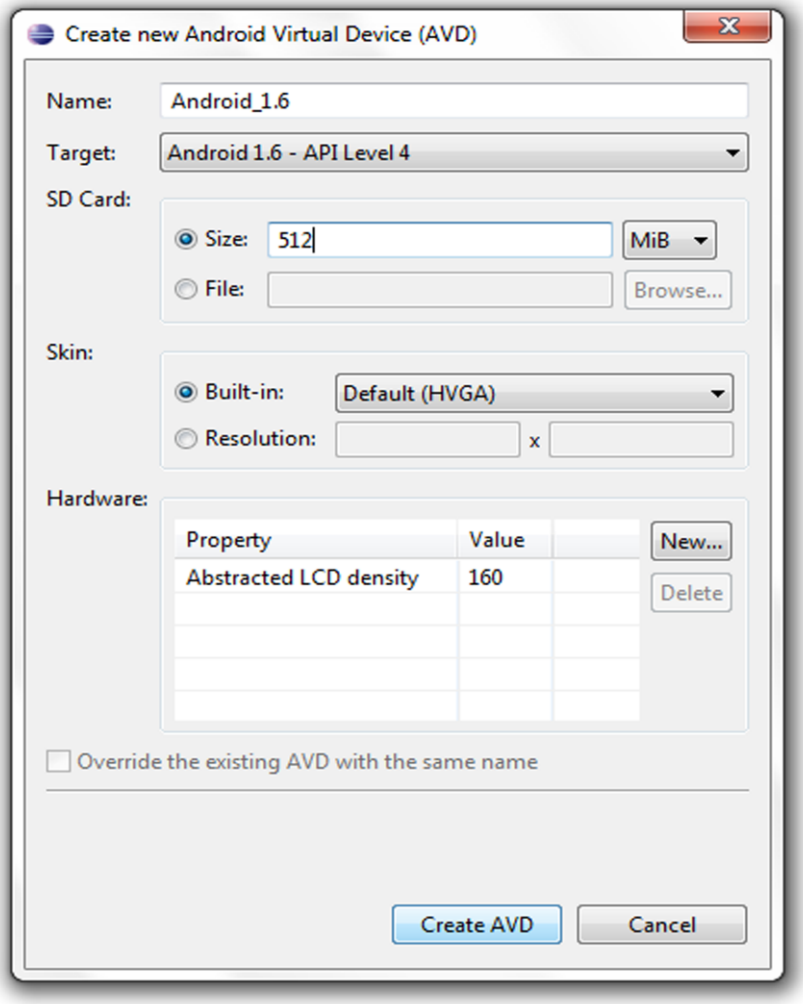

Figure 4.1: Virtual Device Creation [24]

4.1.2 Hardware Used

Two high end Android mobile phones are used to run the java and ndk applications

- HTC one x plus: 1.7 GHz Quad core processor, 1 Gb RAM, 64 Gb inbuilt memory, 4.3 inch LCD displays.
- Samsung Galaxy Grand : 1.2 GHz dual core processor, 1 Gb RAM, 8 Gb inbuilt memory, 5 inch TFT display.

4.1.3 Implementation

4.1.3.1 Integer Calculation

The purpose of the Application Armstrong number Calculation using iteration is to implement an algorithm that perform calculations with integers, thereby exposing the possible performance difference between a native and a standard Java implementation of basic operations on integers.

This Application will take a number "n"as input calculates Armstrong numbers between 1 to n using java and C++ And gives the result with the time taken by both in Java and NDK in milliseconds(ms).Both the Applications are installed on HTC and Samsung and results are evaluated. Here is screenshot of the Application.

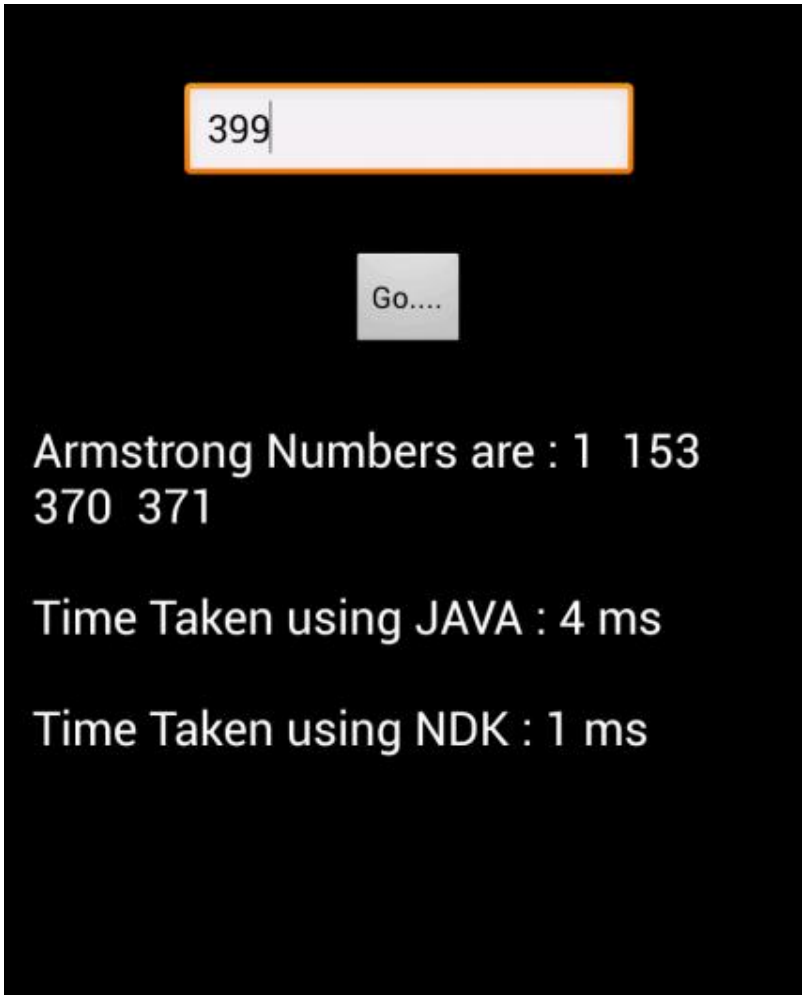

Figure 4.2 Armstrong Number Applications

4.1.3.2 Recursion

Recursion is when the solution of a problem depends on dividing the problem into smaller instances of the same problem. Recursion result in a lot of method calls, which is interesting to benchmark. It is also interesting since recursion is frequently used in computer science generally. So Fibonacci series is implemented using Recursion as an Android Applications.

This Application will take a number “n” as input calculates nth Fibonacci element of the series using java and C++ and gives the result with the time taken by both in Java and NDK in milliseconds(ms).Both the Applications are installed on HTC and Samsung and results are evaluated. Here is screenshot of the Application.

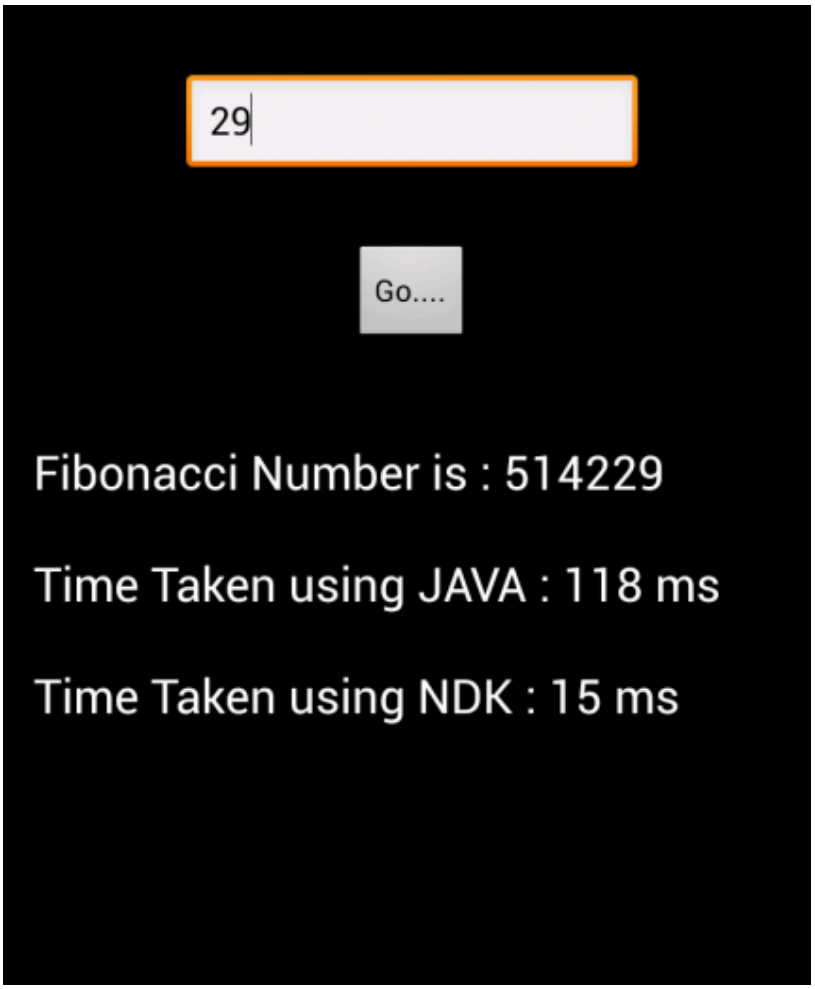

Figure 4.3 Fibonacci number Application

4.1.3.3 Floating Point Calculation

A floating point is a standard primitive type typically used for representation of non-integer numbers. Floating point calculation is usually performed by a hardware unit called the Floating Point Unit (FPU), but some processors' lack this unit and is therefore software calculated. Software based calculation is usually much slower than calculation utilizing dedicated hardware.

The purpose of the implementation is to perform basic operations (such as addition, subtraction, multiplication and division) on floating points and measure the possible performance difference. It gives the result with the time taken by both in Java and NDK in milliseconds(ms).Both the Applications are installed on HTC and Samsung and results are evaluated. Here is screenshot of the Application.

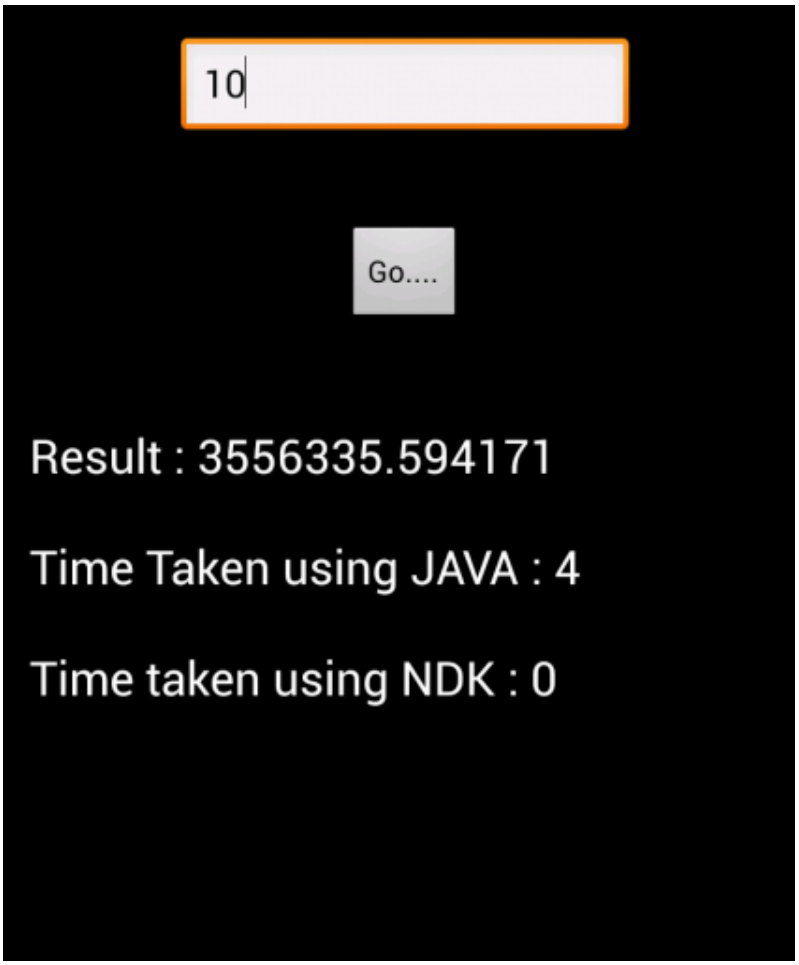

Figure 4.4 Floating point calculation Application

4.1.3.4 Memory Access

The Memory access test is an implementation of an algorithm that performs a lot of memory accesses.All sorting Algorithms accesses memory a lot. So Merge Sort has been implemented as an Android Application. Memory access is an important benchmark due to the fact that most applications read from the memory frequently.

This Application will take a list of numbers as input and Sort the elements using java and C++ and gives the result with the time taken by both in Java and NDK in milliseconds (ms).Both the Applications are installed on HTC and Samsung and results are evaluated. Here is screenshot of the Application.

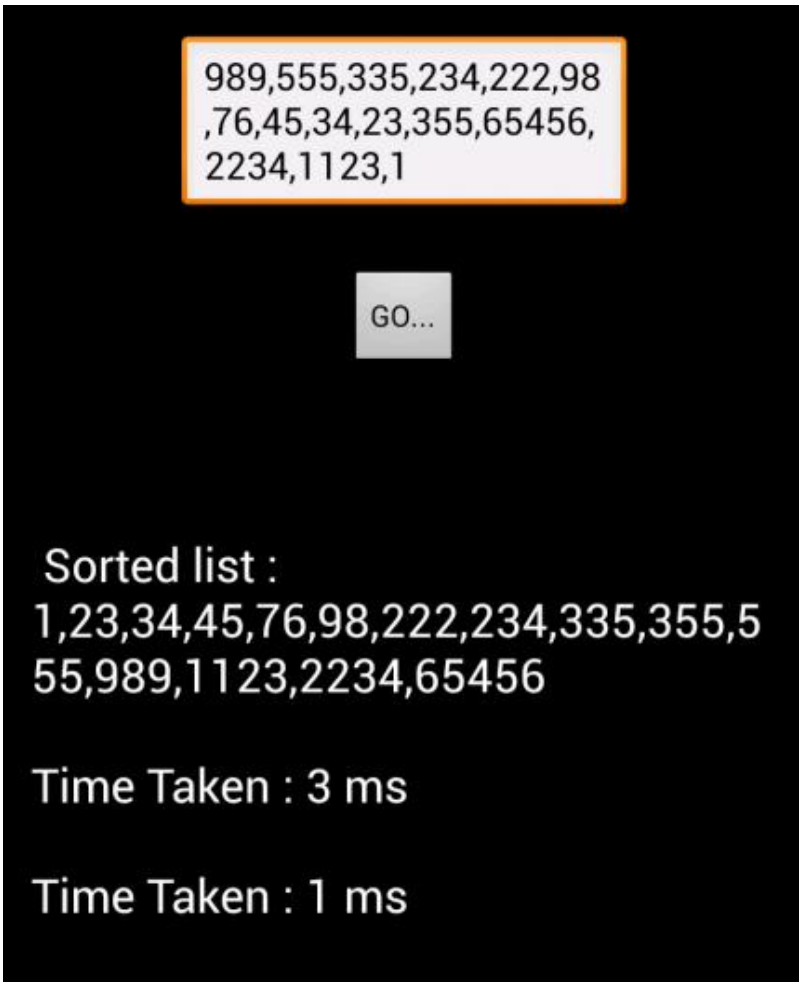

Figure 4.5 Merge Sort Application

4.2 Developing Custom ROM

A Custom ROM is a fully standalone version of the OS, including the kernel (which makes everything run), apps, services, etc - everything you need to operate the device, except it's customized by someone in some way

4.2.1 Software Required

- 64 bit Linux or Mac operating system.
- GNU Make 3.81 - 3.82
- JDK 6 or higher
- Git 1.7 or newer

4.2.2 Hardware used

Samsung Galaxy Fit : 600 MHz processor , 280 Mb RAM, 3.3 inch display.

4.2.3 Building source code

i. Installing Repo : Repo is a tool that makes it easier to work with Git in the context of Android.

- Create a directory bin in home directory and include it in the PATH.
- Download the Repo tool and ensure that it is executable:
  $ curl https://dl-ssl.google.com/dl/googlesource/git-repo/repo > ~/bin/repo
  $ chmod a+x ~/bin/repo
- Create a directory for working Files.
- Run “Repo init” Command .There are two source code original and cyanogenmod source code.
- To download original source code for Gingerbread use “ $ repo init -u https://android.googlesource.com/platform/manifest -b android-4.0.1_r1”
- To download Cyanogen mod source code for gingerbread use “$ repo init -u git://github.com/CyanogenMod/android.git -b gingerbread”.
- Run “Repo sync” command to download the android source tree.

ii. Building the code : To build the files,run envsetup.sh, lunch and make from within the working directory.

- $ source build/envsetup.sh

- $ lunch userdebug full-eng.
- $ make –j32.

iii. Flashing :

- To flash the code on device connect device to the PC through USB and put the device in fastboot mode and then run the command on linux "$ fastboot flashall -w" .
- To flash the code on emulator run command "$ emulator"

4.2.4 Source Overview

Now source can been seen in working directory . It contains many directories and files.Here is description of directories in the Android source code.

- /bionic/ contains all the C libraries,math and other core runtime libraries developed for Android.
- /boot/ contains boot and startup related code.
- /build/ contains the build system implementation including all the core make file templates. An important file here is the envsetup.sh script that will help a lot when working with the platform source. Running this script in a shell will enable commands to setup environment variables, build specific modules and grep in source code files.
- /Cts/ Compatibilty Test. The test suite to ensure that a build complies with the Android specification.
- /Dalvik/ contains the source code for the implementation of the Dalvik Virtual Machine.
- /Development/ contains proects related to development such as the source code for the sdk and ndk tools.
- /Device/ contains product specific code for different devices. This is the place to find hardware modules for the different Nexus devices, build configurations and more.
- /External/ contains source code for all external open source projects such as SQLite, Freetype and webkit.

- /Frameworks / this folder is essential to Android since it contains the sources for the framework. A lot of the mapping between the java application APIs and the native libraries is also done here.
- /Hardware/ contains hardware related source code such as the Android hardware abstraction layer specification and implementation. This folder also contains the reference radio interface layer (to communicate with the modem side) implementation.
- /Kernel/ It is not part of the default source download but can get access to this code either by downloading it manually or by adding the repository to the repo tool. Contains the sources for the Android version of the Linux kernel.
- /Out/ contains the build output placed here after run make. The folder structure is out/target/product/. In the default build for the emulator the output will be placed in out/target/product/generic.
- /Packages/ contains the source code for the default applications such as contacts, calendar, and browser.
- /Prebuilt/ contains files that are distributed in binary form for convenience. Examples include the cross compilations toolchains for different development machines.
- /System/ contains source code files for the core Android system. That is the minimal Linux system that is started before the Dalvik VM and any java based services are enabled. This includes the source code for the init process and the default init.rc script that provide the dynamic configuration of the platform

4.2.5 Configuring a new product

- Create a company directory in //vendor//
  mkdir vendor/<company name>
- Create a products directory in the company directory
  Mkdir vendor/<company_name>/products/
- Create a product specific makefile in the products directory like myproduct.mk and in this file the information can be overridden.

- Create a board specific directory in company directory and also create boardconfig.mk that can be configured for particular board. For example some boards doesn't support.
- To modify system properties , create a system.prop file in <board_name > directory.
- Android.mk file must be included in vendor/<company_name>/<board_name>

4.2.6 Adding a new application as system application

- Create an application in eclipse.
- Place Application project source code in /packages/apps/
- Create Android.mk file for application.Right click on project in eclipse go to new > file and give the name Android.mk and create the file.

  LOCAL_PATH := $(call my-dir)
  include $(CLEAR_VARS)
  LOCAL_SRC_FILES := $(call all-subdir-java-files)
  LOCAL_PACKAGE_NAME := <Package Name>
  include $(BUILD_SHARED_LIBRARY)
- Edit /build/target/product/core.mk and add application in the list of PRODUCT_PACKGES.
- Run make

4.2.7 Changing init.rc startup

- init.rc is located in system/core/rootdir/init.rc
- Change the line "User shell" to "User Root" to get the root permissions.
- All the services are listed in this file . This file can be edited to start and stop any service.
- Run make

4.2.8 Changing Permissions

- There is a file android_filesystem_config.h in the directory /system/core/include/private.
- Linux users,groups,directory and file permissions can be edited in this file.
- Run make

Chapter 5

Experimental Results

This chapter contains all the results of the implementation done.

5.1 Application Comparision Results

5.1.1 Integer Calculation

Results : A Graph showing timed performance in milliseconds for each device and implementation of the Armstrong Number calculation test.

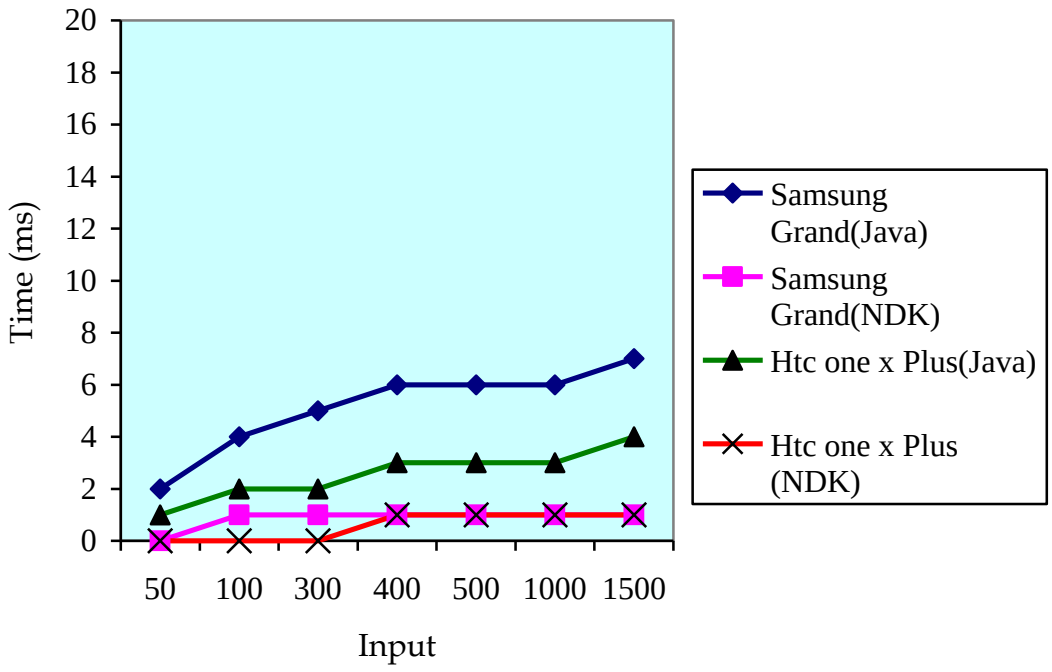

Figure 5.1: Integer Calculation

5.1.2 Recursion

Results: A graph showing timed performance in milliseconds for each device and implementation of the Fibonacci Number calculation test.

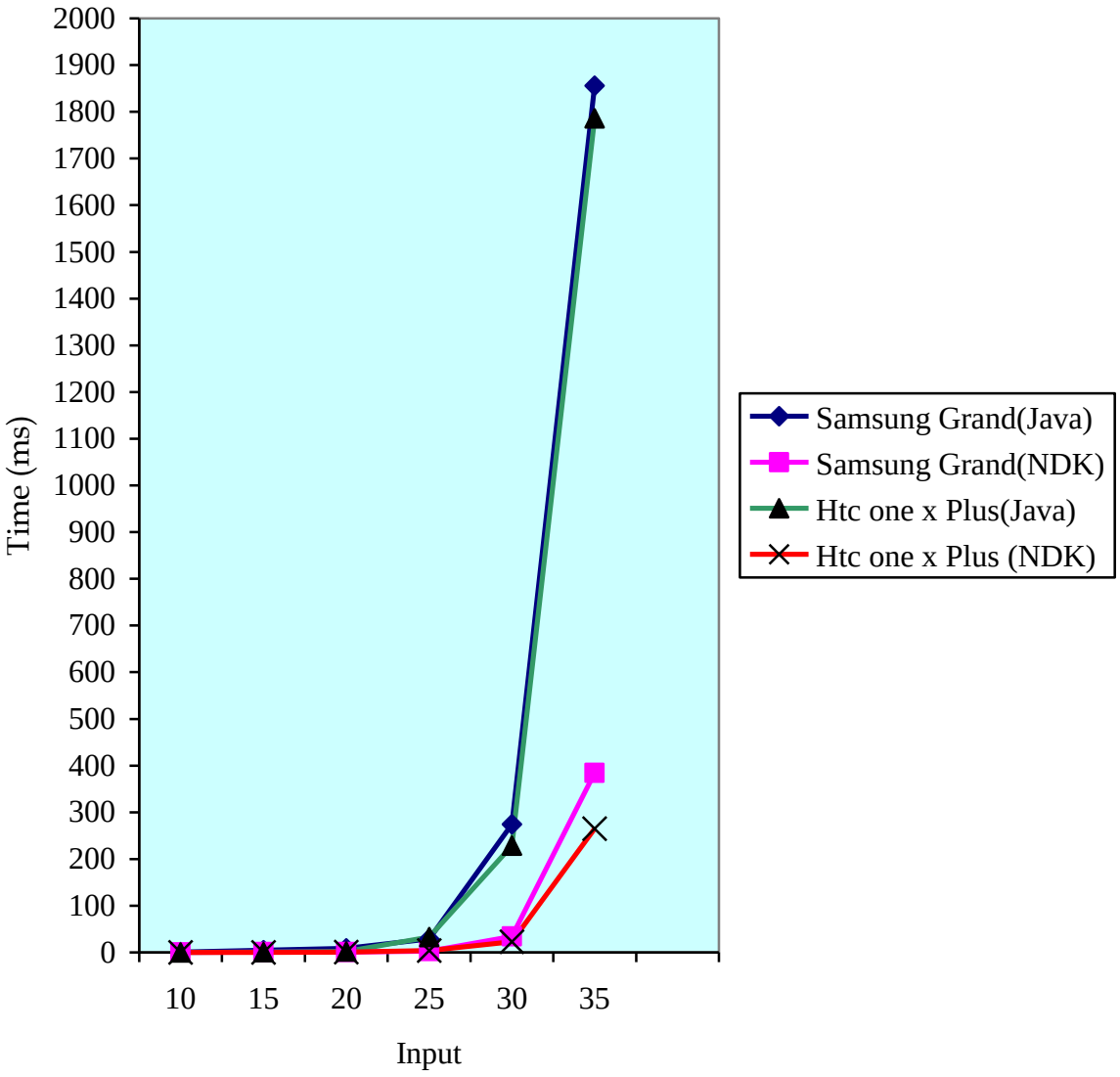

Figure 5.2: Recursion Calculation

5.1.3 Floating point calculation

Results: A graph showing timed performance in milliseconds for each device and implementation of the Floating Point calculation test.

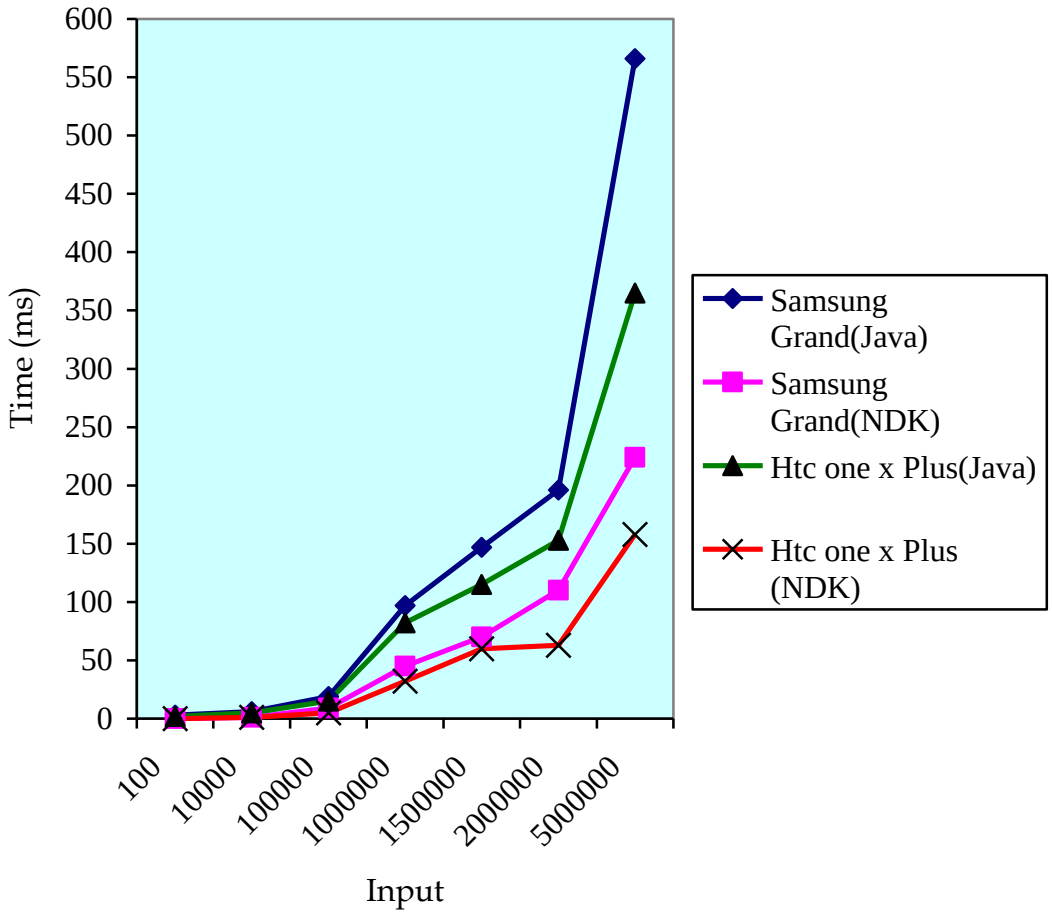

Figure 5.3: Floating Point Calculation

5.1.4 Memory Access

Result: A graph showing timed performance in milliseconds for each device and implementation of the Memory Access test.

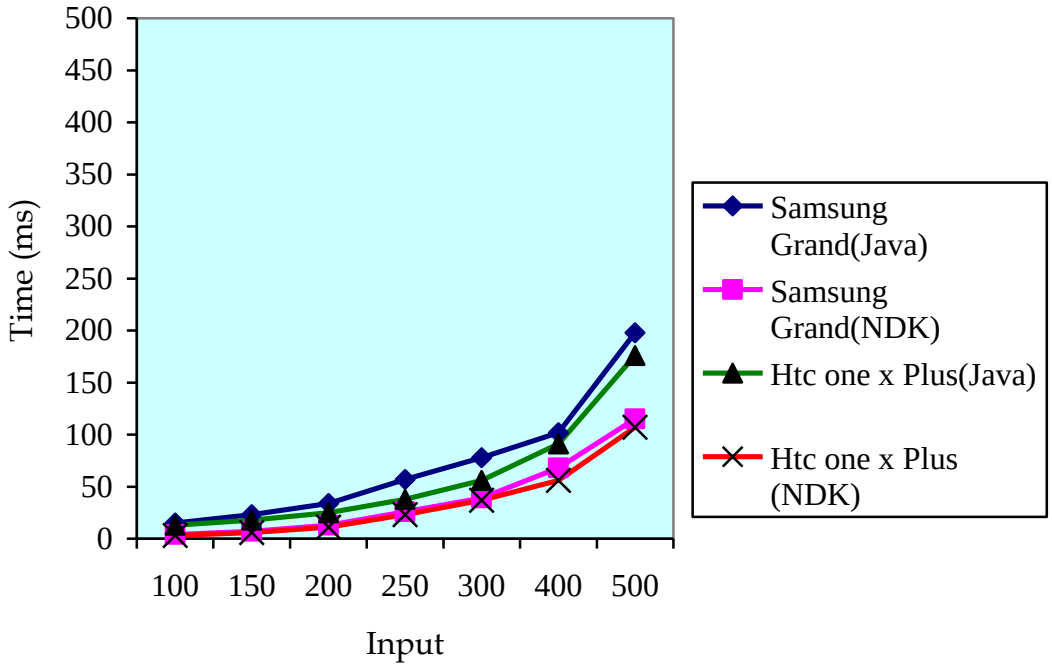

Figure 5.4: Memory Access Calculation

5.2 Stock and Custom ROM comparison result

Linpack is a benchmark used to estimate the performance of computers and phones. Linpack software is installed in both Stock rom and custom Rom on Samsung Galaxy Fit. This software calculates the performance of an Android phone in Mega FLOPS(Floating point operations per second).FLOPS is the measure of speed of computer or phone especially in fields of scientific calculations that make heavy use of floating-point calculations. 1 MFLOPS= 1.000.000 FLOPS. More mflops means the device is giving better performance [26].

Results:

Linpack benchmark's results on Samsung Galaxy Fit with Stock Rom Android version 2.2.1 froyo.

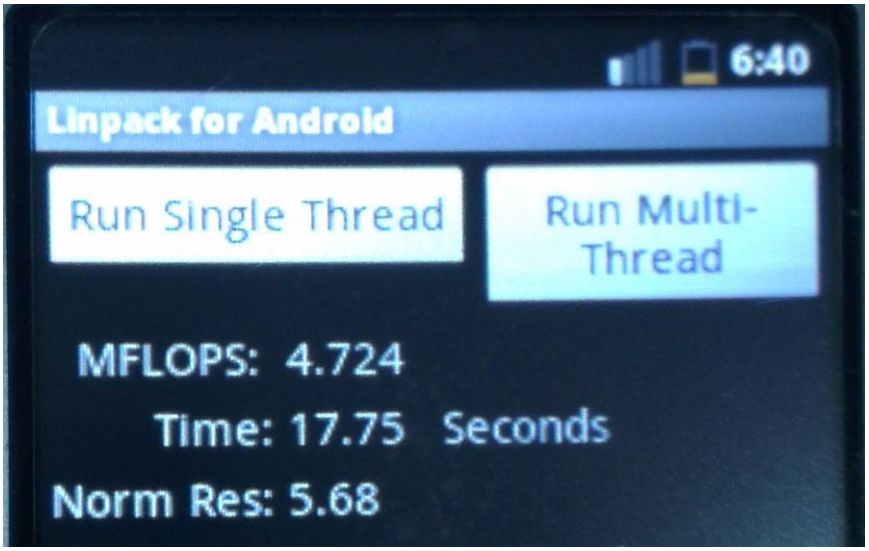

Figure 5.5 Stock Rom Linpack Benchmark

Linpack benchmark's results on Samsung Galaxy Fit with Custom Rom Android version 2.2.7 Gingerbread.

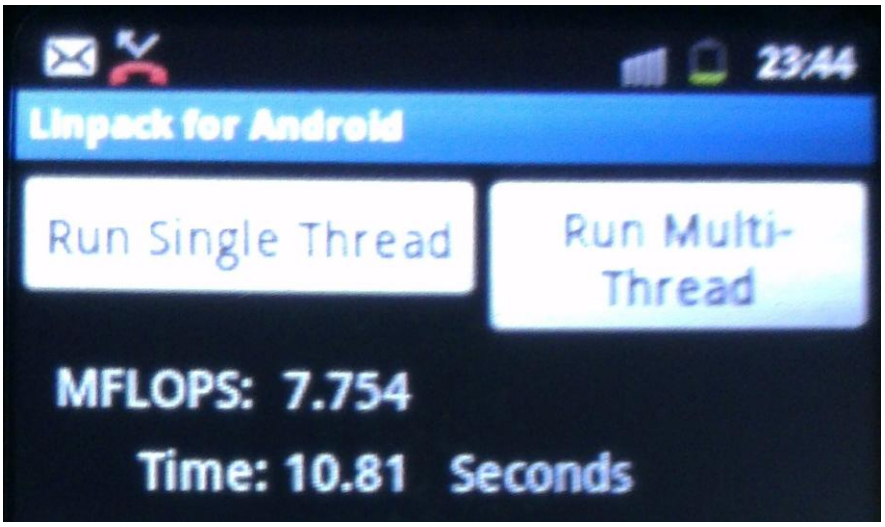

Figure 5.6 Custom Rom Linpack Benchmark

Chapter 6
Conclusions

This section presents discussions and conclusions of the results presented above. It also discusses the future scope of the work.

6.1 Conclusions

HTC one x plus has 1.7 quad core process which is far much better than Samsung Grand 1.2 dual core processer. In Integer calculation there is only slight difference between the performances of the devices Both the devices showed almost same results. In Integer calculation not much difference was seen in the time of Native C/C++ implementation and Java implementation. In Floating point calculation there was lot of difference between the results of HTC one x plus and Samsung grand due to processors. Native C/C++ performance was much better than the Java implementation on both the devices. This result is probably due to the fact that the Dalvik JVM is not utilizing JIT compilation and not performing any advanced optimization and thus the native C code runs much faster since it is not bound by any JVM. Using recursion showed remarkable differences between the performance of native C/C++ and Java. As the input grows the difference increases on both the devices. Merge sort results showed not much difference in the native and Java implementation

Custom ROM scores more Mflops in less Time than Stock Rom this shows that the Custom Roms gives better performance than stock ROMs. Using these methods Android can be customized for specific needs of the user.As the user got root permissions , now user can play with the Android. User can remove useless preinstalled applications and build its own custom Rom with its application installed. Increased Battery performance was seen in custom Rom as compare to the stock Rom.

6.2 Future Scope

As the Dalvik will improve and phone will come with high processing power the need of NDK will decrease. The main need for the NDK in the future thus appears to be the need of running already written C/C++ code on Android, instead of having to port the

existing code into Java. Android programmers are not recommended to use the NDK while developing simpler applications for newer Android smartphones.In Future Dalvik may improve on new Android versions and whether Java performance will beat the NDK performance or not.

As the technology is increasing very fast , new phones are launching in the market with same speed. Android will update their versions with new technology . But the old phones might not get these updates . so custom Rom might be a good option to compete their phones with the updated versions.

References

[1] F. Khomh, H. Yuan, Y. Zou, "Adapting Linux for Mobile Platforms: An Empirical Study of Android,"28th IEEE International Conference on Software Maintenance (ICSM),2012.

[2] N. Lee, S. Lim "A Whole Layer Performance Analysis Method for Android Platforms," IEEE, 2011.

[3] E. Azimzadeh, M. Sameki, M. Goudarzi, "Performance Analysis of Android Underlying Virtual Machine in Mobile Phones," IEEE Second International Conference on Consumer Electronics, 2012.

[4] G. Lim, C. Min, Y.I Eom, "Enhancing Application Performance by Memory Partitioning in Android Platforms," IEEE International Conference on Consumer Electronics (ICCE),2013.

[5] S. Brahler, "Analysis of android architecture", Department of computer science, Karlsruher institute of technologies, Germany, june 2010.

[7] J. Steele, "The Android Developer's Cookbook," New York ,Pearson, 2011.

[8] "What Is Firmware, Stock Custom ROMs & Flashing (Full Guide)," http://www.gorytech.com/2013/03/what-is-firmware-stock-custom-roms.html, [Feb. 15,2013].

[9] Z. Liu, C.S. Nam, D.R Shin, " UAMDroid: A user authority manager model for the Android platform," Advanced Communication Technology (ICACT), 2011 13th International Conference on, pp 1146- -1150 , 2011.

[10] Ketan Parmar "In Depth : Android Boot Sequence / Process ," http://www.kpbird.com/2012/11/in-depth-android-boot-sequence-process.html, Nov 8 2012 [March 26,2013].

[11] "The Android boot process from power on," http://www.androidenea.com/2009/06/android-boot-process-from-power-on.html, June 11,2009[Jan 14, 2013].

[12] R. I. Mutia, "Inter-Process Communication Mechanism in Monolithic Kernel and Microkernel", Department of Electrical and Information Technology Lund University, Sweden.

[13] D. Heger, "Quantifying IT Stability ," 2nd Edition, Instant Publisher, 2010.

[14] H.J Yoon, "A Study on the Performance of Android Platform," International Journal on Computer Science and Engineering (IJCSE), Vol. No. 4, 2012.

[15] F. Maker,Y.H Chan"A Survey on Android vs. Linux," Department of Electrical and Computer Engineering, University of California, Davis Department of Computer Science, University of California.

[16] Corbet, Jonathan, "What comes after suspend blockers," http://lwn. net/Articles/390369/ [Jan 23,2013].

[17] I. Rassameeroj , Y. Tanahashi "Various Approaches in Analyzing Android Applications with its Permission Based Security Models," IEEE International Conference on Electro/Information Technology, 2011.

[18] "Android Kernel Features," http://elinux.org/Android_Kernel_Features [Jan 22,2013].

[19] B. Patrick, "Anatomy & Physiology of an Android," http://sites.google.com/site/io/anatomy--physiology-of-an-android [Feb 17, 2013]

[20] Preetham Chandrian, " Efficient Java Native Interface for Android based Mobile Devices, " Depart. comp. Science, Arizona State University,2011.

[21] Bornstein, "Dalvik VM Internals," http://sites.google.com/site/ io/dalvik-vm-internals [Feb 15,2013].

[22] Hsckborn, Dianne, "Android Developers Blog – Multitasking the Android Way," http://android-developers.blogspot.com/2010/04/multitasking-android-way.html [Jan 19, 2013].

[23] “Managing the Activity Lifecycle,” http://developer.android.com/training/basics/activity-lifecycle/index.html,[Jan 28,2013].

[24] Rick Rogers, John Lombardo, “Android Application development,” O’Reilly ,USA , 2011.

[25] Cherrystone Software Labs, “Algorithmic Performance Comparison Between C, C++,Java and C# Programming Languages,” http://www.cherrystonesoftware.com/doc/AlgorithmicPerformance.pdf , 2010, [April 6,2013].

[26] “ Linpack Benchmark,” http://www.roylongbottom.org.uk/linpack%20results.html [April 28, 2013].

List of Publications

Published

1. Vaibhav Kumar Sarkania, Vinod Kumar Bhalla, “Android Internals,” International Journal of Advanced Research in Computer Science and Software Engineering,” Vol. 3, Issue 6, pp. 143-147, June 2013.

Communicated

1. “Performance Analysis of Android Applications in Different Environments” International Journal on Computer Science and Technology (IJCST).