Agave: A Benchmark Suite for Exploring the Complexities of the Android Software Stack

Martin K. Brown[1], Zachary Yannes[1], Michael Lustig[1], Mazdak Sanati[2],
Sally A. McKee[2], Gary S. Tyson[1], and Steven K. Reinhardt[3]

[1]Florida State University (mbrown@cs.fsu.edu), [2]Chalmers University of Technology, [3]AMD Research

I. Introduction

A 2015 GlobalWebIndex survey showed that 80% of internet users own a smartphone, almost half own a tablet, and the majority of these almost two trillion mobile devices run the Android Operating System. In spite of Android's popularity, the compiler and architecture communities have lacked a standard benchmark suite for designing and optimizing Android systems. To address this need, we propose Agave[1], a suite of full applications representing commonly executed activities and utilizing the diverse components of the Android software stack. Agave follows the spirit of and builds on several earlier Android benchmark suites to create a unified resource consisting of entirely open-source programs.

We begin by defining requirements. A good Android benchmark suite requires real-world applications that can be run within the Android execution environment on hardware devices, emulators, or simulators, and their code should be available under open-source licenses. Android applications often support multiple modes of operation (e.g., an MP3 run in the foreground as a user-facing application and in the background as a service), and benchmarks should reflect this. And although the majority of Android applications use the Native Development Kit (NDK), many still rely on Java, and so both native and Java applications should be represented.

Android is based on a Linux® kernel, but it also includes native libraries, a virtual machine runtime, and an application framework with multiple components for managing resources. All these interact to create a rich, complex software ecosphere. The Android runtime creates many dynamic virtual memory areas (VMAs), with most references to these VMAs generated by system services. This virtual memory organization differs dramatically from the typical C/Linux execution environment, which has a relatively simple virtual memory layout and in which most references come from application code. Android applications also rely heavily on multiprocessing amongst system support services and on internal multithreading in which auxiliary threads often use more resources than main application threads. This structure complicates application profiling, but it also create opportunities to design hardware that better supports Android applications and their underlying software stack. Android benchmarks should thus exhibit diverse process, thread, and memory behaviors.

[1]The name is a play on words: agave nectar is a sweet and increasingly popular dessert ingredient, thus it seems appropriate to use it to describe a companion to the dessert-named versions of the Android OS.

II. The Agave Suite

To meet these requirements, we are developing the Agave benchmark suite for Android. Our initial version of the suite consists of 12 popular applications spanning eight categories (we assume that the number of Google Play downloads reflects popularity). We create multiple versions of some applications, e.g., to run in the foreground versus background or to process different kinds of inputs. The resulting 19 applications are open source, making Agave appropriate for activities ranging from comparing product platforms and modeling new architectures to studying optimizations at any level of the software stack. We make Agave freely available together with our toolchain, and for ease of use we encapsulate the benchmarks in "plug-and-play" virtual machines. Information on how to download and use Agave is available at http://mobile.cs.fsu.edu/benchmarks.

III. Experiments

We run Agave on top of Android 2.3.7, "Gingerbread" and Linux kernel 2.6.35 within the gem5 simulator [1]. We modify gem5 and the OS kernel to generate virtual memory statistics on all processes and threads, and we run gem5's atomic CPU model without caches to quickly generate these statistics. We compare Agave application behaviors with those of a selection of SPEC® CPU2006 [2] to illustrate how Android applications differ from traditional C benchmarks.

Virtual Memory Layout. Android applications have a richer virtual memory layout. Figure 1 shows a breakdown of code regions for Agave versus SPEC CPU2006. The Android applications use instructions from over 65 different regions, and the majority of these references are to `mspace` (for buffering pixel operations) and `libdvm.so` (for executing the Dalvik Virtual Machine). In contrast, the vast majority of instructions executed by the SPEC applications are from the application binary or the OS kernel. Figure 2 shows a similar breakdown for data references to virtual memory regions. The Agave applications use almost 170 different data regions, whereas the SPEC applications largely reference the traditional text, stack, and heap regions. Note that Linux uses the "anonymous" region for heap allocations exceeding `MMAP_THRESHOLD`. Individual Agave applications use between 42-55 code regions and 32-104 data regions.

Multitasking and Multithreading. Android applications consist of many concurrently executing processes. For instance, it is typical for an Android application to use a foreground process for its user interface and a background

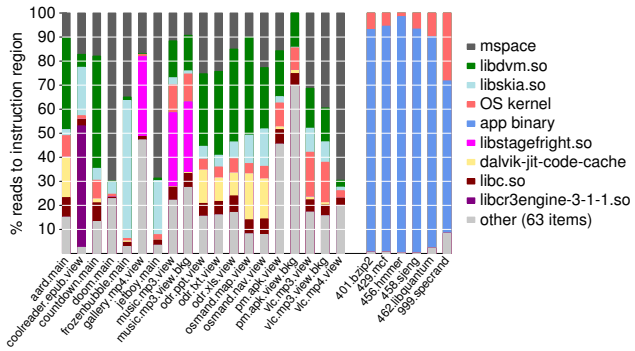

Fig. 1: Instruction references by VMA region

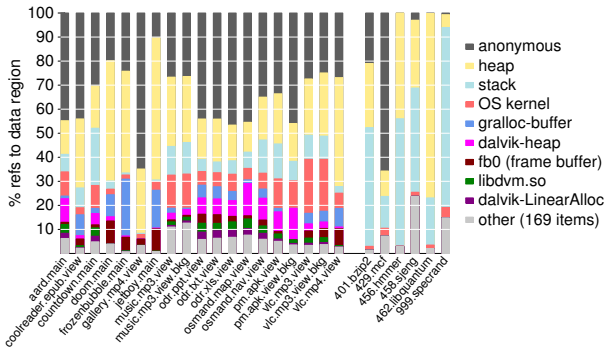

Fig. 2: Data references by VMA region

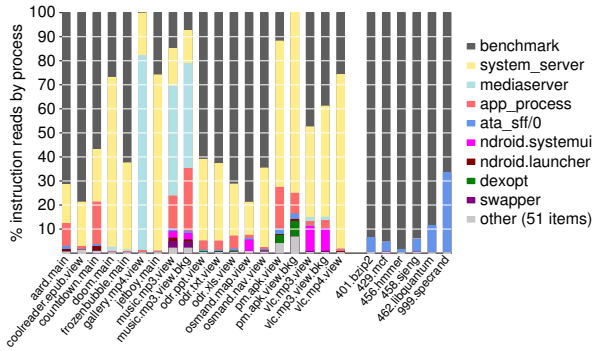

Fig. 3: Instruction references by process

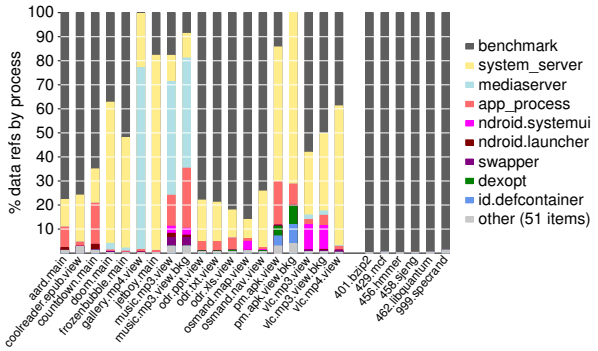

Fig. 4: Data references by process

Thread	% Total Memory References across Suite
SurfaceFlinger	43.4
Thread	8.0
AsyncTask	7.6
Compiler	7.1
AudioTrackThread	5.9
GC	5.3

TABLE I: Memory references from the most-executed threads

process for running its service. Figure 3 and Figure 4 show the breakdown of Agave instruction and data references by process. Agave applications run 20-34 processes, the most heavily referenced of which are `system_server` (which updates the display), `mediaserver`, and `app_process` (one of which is forked for every other process the application spawns). The application process is not always dominant: for instance, in `gallery.mp4.view` (Gingerbread's default video player) `mediaserver` accounts for 81% and 77% of instruction and data references, respectively. In contrast, the single-threaded SPEC benchmarks compete mainly with the `ata_sff/0` system process that manages storage devices.

Table I shows threads ranked by contribution to total memory references. Executing Agave applications spawns 32-147 threads. Many references also come from system service threads. For instance, the `SurfaceFlinger` thread that updates the display accounts for 43.4% of all references.

IV. Conclusions

Traditional suites used for benchmarking high-performance computing platforms or for architectural design space exploration use much simpler virtual memory layouts and multitasking/multithreading schemes, which means that they cannot be used to study the complex interactions among the layers of the Android software stack. To demonstrate this, we present memory reference and concurrency data showing how Android applications differ from traditional C benchmarks. We propose the Agave suite of open-source applications as the basis for a standard, multipurpose Android benchmark suite. We make all sources and tools available in hopes that the community will adopt and build on this initial version of Agave.

Note that execution profiles of some Android libraries appear to be independent of who calls them. Static profiling could thus prove more useful for studying Android application behavior than it has for other types of applications in the past.

V. Acknowledgments

Linux is a registered trademark of Linus Torvalds. SPEC is a registered trademark of the Standard Performance Evaluation Corporation.

References

[1] N. Binkert et al., "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, pp. 1–7, Aug. 2011.

[2] Standard Performance Evaluation Corporation, "SPEC CPU benchmark suite," `http://www.specbench.org/osg/cpu2006/`, 2006.