Characterization of Shared Library Access Patterns of Android Applications

Xiaowan Dong¹, Sandhya Dwarkadas¹, and Alan L. Cox²

¹Department of Computer Science, University of Rochester
²Department of Computer Science, Rice University

Abstract—We analyze the instruction access patterns of Android applications. Although Android applications are ordinarily written in Java, we find that native-code shared libraries play a large role in their instruction footprint. Specifically, averaging over a wide range of applications, we find that 60% of the instruction pages accessed belong to native-code shared libraries and 72% of the instruction fetches are from these same pages. Moreover, given the extensive use of native-code shared libraries, we find that, for any pair of applications, on average 28% of the overall instruction pages accessed by one of the applications are also accessed by the other. These results suggest the possibility of optimizations targeting shared libraries in order to improve instruction access efficiency and overall performance.

Keywords—mobile platforms; shared libraries; Android process creation model;

I. MOTIVATION

Previous works have characterized the microarchitectural behavior of Android applications [1], [2]. In particular, Gutierrez et al. [1] found that the instruction translation lookaside buffer (TLB) and instruction cache performance of Android applications is significantly worse than the SPEC CPU2006 benchmarks. In an effort to explain the poor instruction locality of Android applications, Huang et al. [3] quantified the number of shared libraries invoked and the frequency with which execution switches between them.

In this paper, we provide a deeper analysis of the instruction access patterns of Android applications in order to identify targets for optimization. We examine the role of native shared libraries in the applications’ instruction footprint, as well as the degree to which the libraries are shared across applications.

II. SHARED LIBRARY ACCESS PATTERN ANALYSIS

We conducted our experiments on a Nexus 7 tablet (2012) running the Android KitKat 4.4.4 operating system. This tablet has a 1.2GHz Nvidia Tegra 3 processor with four ARM Cortex-A9 cores. We replaced the default application environment with Android Runtime (ART), which is a new environment that performs “Ahead-of-Time” compilation (AOT) of applications to native code at installation time [4].

To perform the instruction footprint analysis of Android applications, we collected page fault traces for the user address space from the kernel and interpreted these traces using the mapping information from /proc/pid/smaps.

To analyze instruction execution, we used perf to sample the execution of kernel and user space instructions. The reported results are based on an average of ten executions for each application.

We tested commonly used applications from eight different categories, including a game, a document reader, a text editor, two web browsers, an email client, a calendar, a video player, and an audio player. The Android browser and email client are installed by default. The others are frequently downloaded applications from the Google Play Store. To test both the Android browser and Chrome browser, we used the BBench benchmark [1].

A. The Impact of Native Shared Library on The Instruction Footprint

The number of native shared libraries linked into an application ranges from 88 to 107, and the number of native shared libraries called by an application during its execution ranges from 20 to 62. In Figures 1 and 2, we explore the contribution of these native shared libraries to each application’s instruction footprint. We find that, on average, 59.98% of the instruction pages accessed and 71.63% of the instruction fetches were from the native shared libraries.

Figure 1. % of instruction pages accessed that were from the shared libraries. Normalized to the total number of instruction pages accessed.

B. Android Process Creation Model

As is typical in Linux-based operating systems, when Android boots up, the first process to run in user-space executes the init program. Where Android differs is that
init then spawns a process called the zygote, which, as the name implies, is a seed for starting Android application and service processes. The zygote initializes the Android run-time environment, including preloading commonly used native shared libraries into its address space. Android applications and services are then started by forking a process from the zygote. Consequently, this process inherits the preloaded libraries from the zygote through copy-on-write (COW) mappings, which speeds up application launch.

Most of the native shared libraries in the address space of an Android application are preloaded by the zygote. In particular, 88 native shared libraries are preloaded by the zygote on our tablet. As illustrated in Figures 1 and 2, as much as 35.06% of the instruction pages accessed are from the native shared libraries that are preloaded by the zygote, and on average 50.09% of the instruction fetches are from these same libraries.

In Figures 1 and 2, the differences between the all shared libraries percentages (blue bars) and the preloaded shared libraries percentages (red bars) have a two-part explanation. First, application-specific shared libraries are heavily used by some applications, notably Angrybirds and Chrome. Second, many of the applications use shared libraries containing platform-specific drivers. However, neither application- nor platform-specific libraries are preloaded by the zygote.

C. Shared Library Code Accessed Across Different Android Applications

For each pair of applications, we determined the number of shared library instruction pages accessed that were common to (shared by) both applications (the intersection set). On average, 28% of all instruction pages accessed in each application are from this intersection, and this number can be as high as 64%. As one specific example, for each pair of the Android browser, Adobe reader, and MX player, the number of instruction pages shared represents greater than 50% of the total instruction pages accessed.

III. SUMMARY AND IMPLICATIONS

Our analysis shows that (1) native shared libraries play a very important role in the instruction footprint of Android applications; (2) a large fraction of these shared libraries are preloaded by Android’s zygote process, implying that all applications share the same physical and virtual addresses for this code; and (3) there is a considerable overlap in the shared library code accessed across different Android applications.

Moreover, this analysis points to opportunities for improving shared library instruction access efficiency. The Android zygote process creation model motivates a redesign of the Linux memory management subsystem to support sharing both page tables and TLB entries (with hardware support) for shared libraries. Besides reducing storage overhead, sharing page tables has the added benefits of improving fork performance and decreasing the number of soft page faults experienced by the Android applications: once a page table entry is populated in the shared page table page, it is visible to all the sharers. This benefit will likely improve application launch performance. By sharing TLB entries for shared libraries across processes, we can also reduce context switch overheads due to inter-process communication (IPC), which is a common operation on Android platforms.

ACKNOWLEDGMENTS

This work was supported in part by the U.S. National Science Foundation grants CCF-1217920, CNS-1319553, CNS-1320965, and CCF-137224.

REFERENCES


