Analysis of Effective Use of Thread-Level Parallelism in Mobile Applications

Ethan Bogdan and Hongin Yun
CS97 Project Proposal

10/5/2013

Abstract

We intend to study how software development paradigms have (or have not) evolved to take advantage of new potential for parallelism on mobile devices. We will be emulating a variety of popular apps for Android and iOS and tracing their use of threads via the DTrace framework. Our results will offer visual and mathematical evidence of prevalent inefficiencies and concurrency hazards, which could be avoided through smarter multi-threading practices.

I. Motivation

In recent years, hardware, even on mobile devices, has evolved to support parallelism through muli-threading on one or more cores. However, software developers are still catching up. We suspect that many common apps either make limited use of multi-threading, or else generate a lot of small, event-based threads. These threads may be convenient for the developer, but spend most of their time idle and risk unnecessary concurrency issues. We are interested in detecting this phenomenon in mobile devices and studying whether it can be improved by serialization.

II. Background

We draw inspiration for our work from the original study done by Flautner et al. on desktop applications in 2010. [1] The authors investigate whether software developers for desktop machines have followed the hardware trends by creating software for multi-processing. The authors analyze a range of desktop applications on Microsoft Windows 7 and Apple's OS X Snow Leopard for Thread Level Parallelism and conclude that the 2-3 cores are more than sufficient for most applications, and that current desktop applications are not fully utilizing multi-core architectures.

Other studies in a similar vein date back to 2000, when Flautner et al. investigated the thread-level parallelism and interactive response time of desktop applications. [2] This study was done when multiprocessing was prevalent mostly in servers and just began to enter into desktop machines. While servers were considered to be a natural fit for multiprocessing due to its nature of having multiple clients, the benefits of multiprocessing for desktop applications were not obvious. Now as the multi-processor systems are entering smartphone market, we believe it is a natural extension of studies to ask whether the benefits of multiprocessing are fully reaped on mobile devices.

III. Idea

Rather than instrumenting the mobile devices themselves, our plan is to emulate their hardware on a Macintosh computer, then use DTrace to analyze their use of threads on the host system. We are choosing this approach because the full power of Sun's DTrace framework, which supports writing versatile tracing tools in "D code," is only available on desktop systems. Moreover, convenient emulators for both the Android and iOS operating systems are freely available in their respective SDKs. These emulators will be optimal for situations in which we have direct access to source code. Compiled binaries, which exist in far greater supply, present a slightly greater challenge, particulary on iOS.[1] In this case, we will have to use a combination of the third-party BlueStacks (for Android emulation)[2] and Hopper (for iOS decompiling).[3]

Our interest is in running these tools on some of the most commonly used apps for both types of phone. These apps will fall into a representative cross-section of categories (e.g. games, productivity, networking) with extra emphasis placed on those with evidently parallel structure or event systems. We will use a combination of statistical analysis and visualization tools to categorize (i.e. computation-heavy vs. event-based) and characterize (i.e. efficient, suboptimal, etc.) the thread behavior of these apps.

IV. Milestones

Our work naturally lends itself to three separate phases: setup and preparation of tools; application of tools to research a variety of apps; visualization and statistical anaylsis of results. These phases are broken into more concrete goals below:

I. Goals for Milestone 1 (11/6)

The primary tool we'll be using is DTrace, with which neither of us have any experience. By Milestone 1, we will have:

- Installed and configured DTrace
- Learned and practiced D code syntax
- Written thread tracing tools suited to our needs, and tested them on sample desktop and mobile applications

We will also be using this time to set up our emulation software and explore the feasibility of reverse compiling for iOS. Hongin will likely focus more on the Android side, while Ethan tackles iOS. In the end, we should have a fairly representative pipeline put together for our research in Milestone 2.

II. Goals for Milestone 2 (11/20)

The next phase of our work will begin with selecting which apps (and how many of each type) to study. We will try to track down as much source code as we can,[4] but most of our apps will probably come directly from Google and Apple's respective app stores. Then the

[1] "...existing dynamic analysis techniques available for the x86 architecture are not immediately applicable to mobile devices executing binaries compiled for the ARM architecture. For example, many dynamic analysis approaches rely on full system emulation or vitalization to perform their task. For most mobile platforms, however, no such full system emulator are available. While Apple, for example, includes an emulator with their XCode development enviornment, this emulator executes x86 instructions, and therefore requires that the application to emulate is recompiled." [3]

[2] This software is actually still under development, but early versions are available, and a full release is expected before our project is due. See: http://www.bluestacks.com

[3] Hopper appears to be a fairly high-functioning program for disassembly and decompiling, but it remains to be seen whether apps recompiled for Intel processors will actually behave comparably to their original versions. If not, we may need to limit our research mostly to Android apps, (or iOS apps for which we have the source). I wonder whether any developers would be willing to provide us with binaries compiled specifically for desktop emulation?

[4] Ethan may be able to obtain source for some popular apps from his summer internship

main task will be to use our pipeline to collect relevant thread data, under a variety of theoretical use cases for each of these apps. Once again, Hongin will take the Android apps, while Ethan takes the iOS ones. Our reach goals for Milestone 2 may include:

- colllecting data for serialized re-writes of the apps for which we have source code
- assessing utilization of a phone's GPU

III. Goals for Milestone 3 (12/4)

If we've succeeded in performing the bulk of our coding and testing by Milestone 2, the remainder of our time should be free for collating our results and making sense of the data. While our priority is to perform a thorough mathematical analysis (probably involving some clustering algorithms), we would also like to present an intuitive visual representation of threading patterns. Finally, we will use Milestone 3 to evaluate the implications of our results, in terms of multi-threading more broadly and parallelism on mobile devices in particular, and consider where there is room for further work.

References

[1] K. Flautner, G. Blake, R. G. Dreslinski, and T.Mudge. Evolution of thread-level parallelism in desktop applications. *SIGARCH Computer Architecture News*, 38:302–313, 2010.

[2] K. Flautner, R. Uhlig, S. Reinhardt, and T.Mudge. Thread-level parallelism of desktop applications. *Workshop on Multithreaded Execution, Architecture, and Compilation*, 2000.

[3] Martin Szydlowski, Manuel Egele, Christopher Kruegel, and Giovanni Vigna. Challenges for dynamic analysis of ios applications. In Jan Camenisch and Dogan Kesdogan, editors, *Open Problems in Network Security*, volume 7039 of *Lecture Notes in Computer Science*, pages 65–77. Springer Berlin Heidelberg, 2012.