Full-System Analysis and Characterization of Interactive Smartphone Applications

Anthony Gutierrez, Ronald G. Dreslinski,
Thomas F. Wenisch, Trevor Mudge
Advanced Computer Architecture Laboratory
University of Michigan - Ann Arbor, MI
{atgutier, rdreslin, twenisch, tnm}@umich.edu

Ali Saidi, Chris Emmons, Nigel Paver
ARM - Austin, TX
{ali.saidi, chris.emmons, nigel.paver}@arm.com

***Abstract*—Smartphones have recently overtaken PCs as the primary consumer computing device in terms of annual unit shipments. Given this rapid market growth, it is important that mobile system designers and computer architects analyze the characteristics of the interactive applications users have come to expect on these platforms. With the introduction of high-performance, low-power, general purpose CPUs in the latest smartphone models, users now expect PC-like performance and a rich user experience, including high-definition audio and video, high-quality multimedia, dynamic web content, responsive user interfaces, and 3D graphics.**

In this paper, we characterize the microarchitectural behavior of representative smartphone applications on a current-generation mobile platform to identify trends that might impact future designs. To this end, we measure a suite of widely available mobile applications for audio, video, and interactive gaming. To complete this suite we developed *BBench*, a new fully-automated benchmark to assess a web-browser's performance when rendering some of the most popular and complex sites on the web. We contrast these applications' characteristics with those of the SPEC CPU2006 benchmark suite. We demonstrate that real-world interactive smartphone applications differ markedly from the SPEC suite. Specifically the instruction cache, instruction TLB, and branch predictor suffer from poor performance. We conjecture that this is due to the applications' reliance on numerous high level software abstractions (shared libraries and OS services). Similar trends have been observed for UI-intensive interactive applications on the desktop.

I. Introduction

Embedded processors, such as those found in smartphone and tablet devices, are quickly becoming the most widely used processors in consumer devices. International Data Corporation (IDC) estimates that vendors will ship 472 million smartphones in 2011 and that number is expected to double by 2015 [19]. The cores in modern smartphones are growing increasingly sophisticated. TI's OMAP 4 [30] and NVIDIA's Tegra 2 [23] contain chips that include dual-core, out-of-order ARM Cortex-A9 [4] processors. Future devices will continue this trend towards high performance, for example TI's OMAP5 platform [30] will contain two ARM Cortex-A15 processors [5].

With the increase in performance available in modern smartphones, users expect rich interactive applications, including web-browsers, games, and multimedia, to be delievered on their smartphones at PC-like performance. Computer architects and systems engineers need to consider this important application space when designing and optimizing their hardware. In this paper, we characterize the microarchitectural behavior of representative smartphone applications on a current-generation mobile platform to identify trends that might impact future mobile CPU designs.

Traditionally, when designing new platforms, systems designers and researchers often use the SPEC benchmarks [27]—the most widely used general purpose benchmarks in the research community—as a way to measure the baseline performance of their systems. By design, the SPEC benchmarks stress primarily the CPU and are meant to be portable, avoiding extensive use of system calls and shared libraries. Over time, the datasets of successive SPEC CPU generations have been grown to ensure that the CPU remains busy and the benchmark runtime is long enough to facilitate repeatable measurement. Although these design choices make the SPEC benchmarks useful for evaluating CPU performance, they give an incomplete picture when trying to evaluate complete systems. This fact has been observed in previous studies on desktop and server platforms [1], [7], [25], [31], [32].

We design and measure a new suite of real-world interactive smartphone applications including interactive gaming and multimedia playback. To complete this suite we developed *BBench*, a new fully-automated benchmark to assess a web-browser's performance when rendering some of the most popular and complex sites on the web. BBench comprises a set of self-contained snapshots of some of the most popular and sophisticated sites on the web selected to excercise a wide range of browser functionality. Modern web-browsers are increasingly feature rich and have capabilities far beyond simply browsing the web; we design BBench to capture this range of behavior. BBench is fully-automated and self-contained such that it can be run offline or through a local server to obtain meaningful and reproducible performance results. The importance of the web-browser is evident in efforts such as Google's Chrome OS [14] and a new feature in Mac OS X Lion that allows users to boot into the Safari browser [2]. The web-browser is also one of the most frequently executed interactive application on smartphones. Though we only analyze BBench behavior on smartphone platforms in this study, it is portable and has been tested on a wide variety of mobile and desktop browsers.

We contrast the behavior of our real-world smartphone application suite with a subset of the SPEC CPU2006 benchmarks. There are some obvious qualitative differences between SPEC CPU and interactive smartphone applications. The smartphone applications are interactive and contain rich graphical user interfaces. To facilitate these interfaces and reduce time-to-market while ensuring maintainability and consistency of look-and-feel, the real-world applications often rely heavily on shared libraries and OS features to improve programmer productivity and add functionality. The applications are often multi-threaded to increase performance or facilitate event-driven programming paradigms spawning numerous threads over the course of execution—a trend we expect to increase as more cores are integrated on chip. Their interactive nature also makes their execution inherently less predictable. In contrast, the SPEC benchmarks are explicitly designed to be single-threaded, CPU-intensive, and portable across a wide range of platforms and compilers. Hence, they eschew extensive use of OS services and shared libraries and tend to have more compact code footprints.

These qualitative differences do not however imply that there are necessarily quantitative differences between the micro-architectural characteristics of real-world interactive smartphone applications and the SPEC benchmarks. It is thus our objective to characterize our representative smartphone applications and identify trends that reveal implications for future mobile CPU designs that perhaps differ from what the SPEC benchmarks might reveal.

Our study demonstrates that there are indeed tangible and important differences between real-world smartphone applications and the SPEC benchmarks. Most notably, we discover that smartphone applications suffer from increased code size and sparseness [31]—as evidenced by high branch misprediction, instruction cache and instruction TLB miss rates, and extensive time spent in the operating system and shared libraries—in stark contrast to the SPEC benchmarks, but much like interactive applications on the desktop. Conversely, we find that mobile CPUs are well-equipped to handle the data caching demands of the interactive applications, and that these applications exhibit small footprints and good locality comparable or more favorable than SPEC. These observations suggest future mobile CPUs should be designed to better handle the large and varied code footprints of interactive applications.

In summary we make the following contributions:

- We indentify several of the most important classes of interactive smartphone applications and construct a suite comprising representative benchmarks.
- We develop a new web-browser page rendering benchmark, *BBench*, which is fully automated and can be run offline or from a local server.
- We demonstrate that real-world smartphone applications differ markedly from SPEC, and suffer performance penalties incurred due to the use of high level software abstractions on current mobile CPUs.

The rest of this paper is organized as follows. In Section II, we describe our benchmark suite and explain the setup for each benchmark. We also detail the design of our browser benchmark, BBench. In Section III, we present our experimental framework and our methodology for characterizing each benchmark. We present and discuss our experimental results in Section IV. In Section V, we describe related work. Finally, we conclude in Section VI.

II. Benchmarks

A key objective of our work is to design a benchmark suite that represents the most important application classes relevent to smartphone users, including web browsing, gaming, and multimedia. In particular, our goal is to run real-world smartphone applications on top of the Android operating system in realistic use cases to allow us to capture representative microarchitectural characterizations. We include four benchmarks: *BBench*, a web-browser performance test; *Taps of Fire (TOF)*, an interactive game. *ServeStream*, a streaming video player; and *Rockbox*, an mp3 audio player.

Our web-browser benchmark, BBench, is fully automated. It exploits the JavaScript engine within the browser to automate navigation between pages. The other three applications we study are all interactive and we are aware of no easy means by which to automate them within the Android ecosystem. Hence, we develop procedures to operate each benchmark manually while minimizing run-to-run variation. We have tested these procedures to ensure repeatable results and average our reported characterizations over ten runs. Similar methods have been used in prior studies of interactive games [8].

This benchmark suite and associated procedures will be made publicly available at *http://www.gem5.org/BBench*. The following subsections describe each benchmark in greater detail.

A. BBench

BBench is an automated benchmark that tests a browser's page rendering performance. It comprises a sequence of snapshots of a varied selection of the most popular sites on the web in 2011. Because its goal is to test only rendering (as opposed to network) performance while minimizing run-to-run variance, BBench renders pages offline or from a local server. Although we characterize the Android browser in this study, BBench is portable to any JavaScript-enabled browser and has been tested on all major desktop web-browsers—Chrome, Firefox, Internet Explorer (Windows Only), Opera, and Safari—on Linux, Mac, and Windows. Similar web-browser benchmarks have been developed in the past [10], [18] however these are either outdated, encounter compatibility problems on the Android browser, or are not freely available for academic use.

We select webpages to include in the benchmark based on two criteria: (1) to maximize the diversity of content and page styles (e.g., dynamic content, video, images, Flash, CSS, HTML5, etc.), and (2) to cover some of the most visited sites on the web (e.g., as reported by Google [13]). The list of

Webpage	Description
http://www.amazon.com/	Amazon is a leading online retailer. This site contains much dynamic content that is generated based on a user's recent viewing history and preferences.
http://www.bbc.co.uk/	The BBC is the world's largest news broadcaster. The BBC's website is a comprehensive news website and contains dynamic content, including flash that highlights several different news stories and also contains content for all of the television programming on the BBC. There is also a significant amount of advertising contained on the page.
http://www.cnn.com/	CNN is the leading cable news provider in the U.S. This site contains a significant number of images and dynamic content, including scrolling news stories and advertisements.
http://www.craigslist.org/	Craigslist is a popular site for listing free classified advertisements and personal ads. This site is almost purely html and consists of many links.
http://www.ebay.com/	eBay is the leading internet auction site. This webpage has a very complicated layout and contains dynamic content based on a user's recent viewing and or purchase history.
http://espn.go.com/	ESPN is a sports news website. This site contains an extremely complicated layout, several Flash elements, and much dynamic content.
http://www.google.com/	Google is a popular web search engine. Google's main page is well known for its simple, bare-bones design however, it still contains significant amounts of JavaScript.
http://www.msn.com/	MSN is the internet portal for the Microsoft Network. MSN has a very complicated page layout and features a significant amount of dynamic content including popup search bars, scrolling news stories, and advertisements.
http://www.slashdot.org/	Slashdot is an internet and technology news site that focuses on displaying brief news updates. This site contains a significant amount of advertising and makes heavy use of CSS. The CSS engine is one of the most CPU intensive parts of a modern web-browser.
http://www.twitter.com/	Twitter is a leading social networking site and blogging site. While Twitter has a relatively simple page layout it contains dynamically generated user updates and a significant number of images.
http://www.youtube.com/	YouTube is the leading high definition video streaming site. It contains many elements such user comments, advertisements, and a Flash video player. We include a YouTube page with a 1080p high defition video.

TABLE I: BBench Webpages.

pages, along with a brief description of each, are included in Table I.

While we have created fully functional offline versions of both ESPN and YouTube we did not include them in our evalution of BBench on Android. The reason for this is that we could not find a way to determine with certainty that the Flash/video player was fully rendered and that the Flash content or video actually played before jumping to the next page.

Much of our effort in preparing the benchmark lay in capturing all the necessary content (images, audio/video streams, etc.) from sophisticated web sites to render the page offline without error. To collect these pages for offline use, we use a tool called HTTrack [17]. HTTrack allows us to download a nearly complete archive of a webpage, including the HTML, JavaScript, CSS, images, and other content. However, even HTTrack is not sufficient to capture all of the content required for some pages, as many of these sites download content from dynamically-generated URLs (e.g., JavaScript code that selects an ad to display). For cases in which the HTTrack archive is not complete, we manually download and insert the remaining content pointed to by the dynamically generated links. To locate this content, we analyze each page in the Firefox browser and use its built-in JavaScript debug console and the Firebug plugin [11]. The debugger reports an error for each missing link, which we can then repair by manually downloading and inserting the missing content directly into the page's HTML source.

To automate benchmark execution, we use the browser's built-in JavaScript engine to navigate to each page when rendering of the previous page is complete, using the JavaScript *onLoad* event, which is triggered once a page has loaded. One of the challenges of testing rendering times in a fair manner is to ensure that a page is fully rendered before navigating to the next page—many web-browsers exploit gray areas in the JavaScript spec and trigger the *onLoad* event before complete pages (in particular, off-screen elements) are fully rendered. To prevent this behavior, the *onLoad* handler uses the *document.body.scrollHeight* object to determine the height of the page and then scrolls through the length of the page via JavaScript commands. Then, once the scrolling is complete (which requires rendering the full page), the script sets the *window.location.href* object to the next page in the benchmark sequence. The driver script is configurable and can iterate through a set of pages any number of times.

We validate the reproducibility of BBench results through a test experiment that examines the run-to-run variation of the microarchitectural performance counters that we study in greater detail in Section IV. We execute the benchmark ten times, where each run iterates through the set of webpages five times; further details of our methodology and test platform appear in Section III. We report the coefficient of variation (CV) of the performance counters in Table II. As the results show, the CV is below 1.5%—easily low enough to meet our reproducibility objectives—for all counters except the TLB miss counters, which are susceptible to slightly higher variance due to the smaller number of TLB misses relative to other microarchitectural events.

B. Taps of Fire

With the introduction of high performance CPUs and mobile GPUs, interactive games have become important applications on smartphone devices. To represent this application class, we

Counted Event	Coeff. of Var.
Predictable Branches	0.55 %
Mispredicted Branches	0.49 %
Instruction Cache Stall Cycles	1.38 %
Data Cache Stall Cycles	1.16 %
TLB Stall Cycles	0.82 %
Instruction Cache Misses	0.84 %
Data Cache Misses	0.70 %
Instruction TLB Misses	7.61 %
Data TLB Misses	5.47 %
Instructions	0.65 %
Cycles	0.78 %

TABLE II: BBench Variability. Coefficient of variation of key microarchitectural statistics is low.

select Taps of Fire (ToF) [29]. ToF is an open source rhythm game for Android that is written in Java. It is similar to the popular Guitar Hero franchise. We select this game because of its high level of fast moving on-screen graphics and because it is representative of one of the most popular genres in gaming.

To play the game, the user must strike a note by clicking a button at the correct time (i.e., on the beat) based on a marker moving across the screen. In our benchmark procedure, we play a single song to completion. We measure performance counter statistics from the start to the end of the song. Because the nature of the game elicits the user to produce near-identical input in each session (indeed, this is the very objective of the game), there is little variability from run to run. We select an easy playing mode and play only a single string of notes to maximize the benchmark operator's ability to produce a repeatable input. We repeat the experiment ten times with the same song and average results over these ten runs.

C. ServeStream

High definition video is now ubiquitous, and with mobile applications such as YouTube [33] smartphone users now frequently view high definition video. To represent this application class, we select ServeStream [26]. ServeStream is an open source HTTP media server and stream player that is written in Java for the Android platform. It has a graphical user interface that displays player controls and track information. ServeStream is representative of mobile video player applications. We select this particular video player because of its ability to stream many of the most popular and current high definition video standards, including MPEG 4.

Our benchmark procedure consisted of playing a 30 second MPEG 4 video clip, hosted by a local server, over a dedicated wireless network. We begin capturing performance counter statistics and then immediately navigate the StreamServe browser to the HTTP address of the video file. Upon completion of the video, we manually terminate performance counter collection. We repeat this procedure ten times using the same video clip and report the average results.

D. Rockbox

With the advent of mobile MP3 players, users have become accustomed to having their music on the go. To avoid the inconvenience of carrying multiple devices many consumers opt to use their smartphones as MP3 players. We select Rockbox [24] to represent this important application class. Rockbox is an open source audio player for a wide variety of different audio formats that is written in C and has been ported to Android. Rockbox is representative of mobile audio player applications. We select this particular player because of its portability to a variety of platforms and its rich graphical user interface comprising full player controls, track play time, and album art.

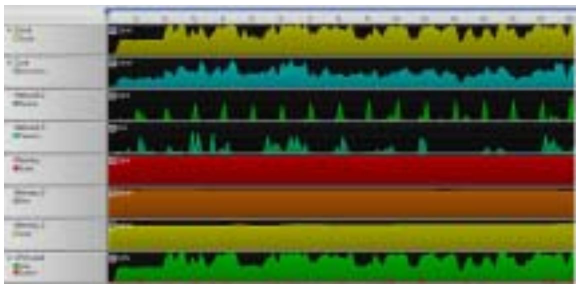
Our benchmark procedure consists of playing the first minute of an MP3 file using Rockbox. We begin collecting performance counter statistics and then immediately start playback in the Rockbox player interface to begin playing the selected MP3. As soon as the player timeline displays one minute of elapsed playback time, we stop collection performance counter statistics. The MP3 file is stored on the device's local file system. As with the other benchmarks, we repeat the procedure ten times with the same MP3 and report average results.

E. SPEC CPU2006

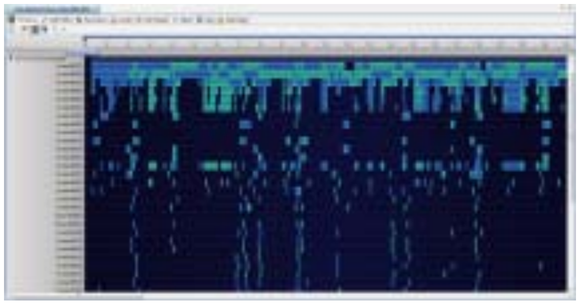
Finally, we include a subset of the SPEC CPU2006 benchmarks [27] in our characterization to support our main thesis that modern mobile applications differ markedly from the widely used CPU-intensive desktop/engineering benchmark suite. We choose to run a subset of the suite and use the train rather than the reference inputs due to memory and run-time considerations. We exclude SPEC benchmarks and input sets that do not fit within the memory footprint of a typical smartphone today and chose the train input sets to limit the experimentation time. From CINT2006, we include *astar* (rivers.bin), *bzip2* (combined), *h264* (raw), *libquantum*, *mcf*, *omnetpp*, and *sjeng*. Finally, we include a single floating point benchmark, *milc*. We compile the benchmarks using CodeSourcery's GNU compiler toolchain for ARM Linux. The SPEC CPU2006 benchmark programs and their key characteristics are described in detail by Henning [16].

III. Methodology

To investigate the microarchitecture-level behavioral differences between our smartphone application suite and SPEC, we capture performance counter data on a smartphone development board. All experiments were performaned on an NVIDIA Tegra 250 development board running Android version 2.2 and version 2.6.32.9 of the Android kernel. The Tegra 2 chip contains a 1GHz dual-core ARM Cortex-A9 processor with 32KB private instruction & data L1 caches, a 1MB shared L2 cache, and an NVIDIA GPU. It is representative of the latest in mobile and smartphone technology. It is important to note that we run unmodified applications (obtained directly from the developers) on the same commercial-grade Android operating system, driver, and library software stack as typical modern smartphone. We execute each benchmark in an otherwise idle system, and unload all drivers and daemons that are not



(a) Streamline Timeline Displaying Counter Statistics



(b) Streamline Displaying Thread Usage For BBench

Fig. 1: The Streamline Performance Analyzer.

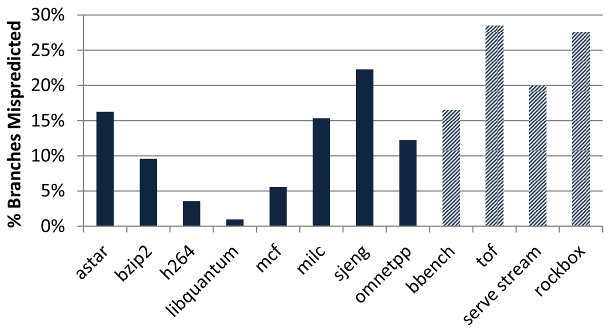


Fig. 2: Branch Misprediction Rate. The branch misprediction rates of the interactive applications are similar to or worse than the worst SPEC benchmarks.

neccessary for the correct execution of the benchmarks. We run a single benchmark at a time.

The ARM Cortex-A9 has a performance monitoring unit that gives access to a wide variety of performance counters. [3], [9] The chip's performance monitoring unit collects many of the statistics throughout the execution pipeline rather than at the commit stage. Hence, some of the statistics (e.g., instructions and miss events) included both committed and wrong-path instructions (speculative instructions that are subsequently squashed due to a branch misprediction). However, because the Cortex-A9 instruction speculation depth is relatively shallow, we do not expect these overcounts to distort results significantly. Furthermore, the data and instruction miss counters include counts that are satisfied by any level of the memory hierarchy (L2 or main memory). However, there is a separate counter for the number of stall cycles due to misses, allowing us to obtain good estimates for average memory access latency. The chip can count up to six events simulataneously in addition to the number of elapsed clock cycles. To gather more statistics, we repeat each experiment multiple times with different combinations of counters, and report average values from ten measurements.

To collect the performance counter data we use a tool called Streamline [6]. Streamline is a sampling-based performance counter monitoring tool used for system level analysis on ARM-based platforms. With Streamline, we are able to capture a variety of additional pertinent information including shared library usage, network requests, memory and disk usage, and the approximate amount of thread level parallelism exhibited by the applications. An example of the output produced by Streamline can be seen in Figure 1. Figure 1(a) shows the performance counter statistics collected over time for a given benchmark run. Figure 1(b) shows the thread usage of an application for a given benchmark run.

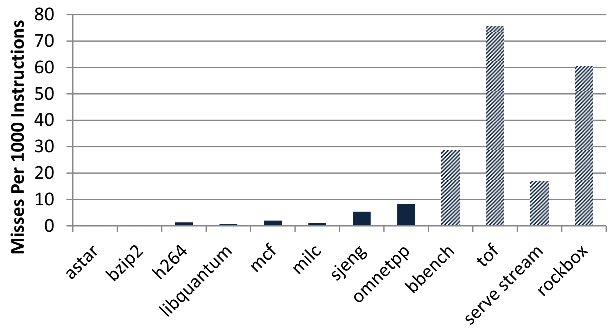
IV. Results

In this section, we explain our characterization results and highlight the most notable differences between the SPEC benchmarks and our interactive smartphone application suite. In particular, we note significantly higher instruction cache and instruction TLB miss rates, a greater amount of system and shared library usage, and higher branch misprediction rates in our suite. In all graphs, our suite are the four rightmost bars.

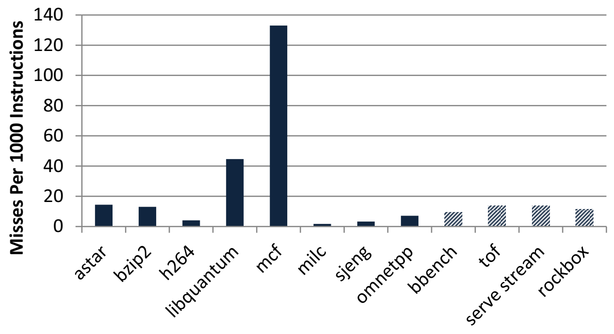
A. Branch Misprediction Rates

Branch prediction is one of the most important features utilized in modern microprocessors to ensure high performance, particularly to exploit instruction level parallelism. Applications that exhibit poor branch predictability can suffer greatly because they spend a significant portion of time executing intructions that are subsequently squashed.

We report branch performance in terms of the branch misprediction rate—the percentage of predictable branches that are predicted incorrectly. (Note that, in current ARM cores, certain less common classes of branch operations are specified as not predictable and are not included in the branch miss rate statistic, these are extremely rare however and are unlikely to affect the results). As can be seen in Figure 2 the branch misprediction rate for each of the interactive applications is worse than nearly all of the SPEC benchmarks evaluated in this study. The only SPEC benchmarks that have branch misprediction rates comparable to the interactive smartphone applications are astar and sjeng. These two applications perform back-tracking searches that require numerous data-dependant branches that are inherently non-predictable. In contrast, in the smartphone applications, the high misprediction rate is due to the massive code footprints that overwhelm the capacity of the embedded-class CPU's predictor. Note that, though branch misprediction

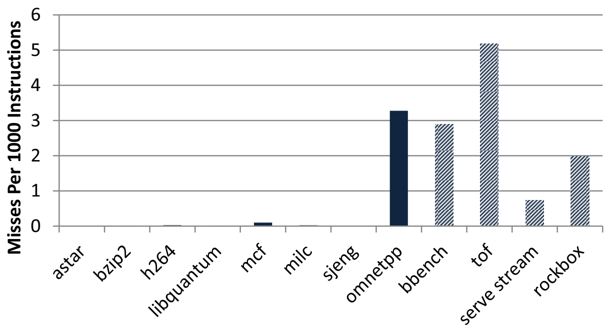


(a) ICache Miss Rate Per 1000 Instructions

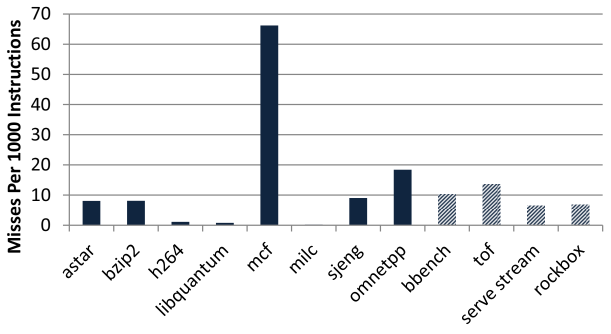


(b) DCache Miss Rate Per 1000 Instructions

Fig. 3: Cache Miss Rates. Although the data cache miss rates for SPEC are similar and in some cases much worse than the interactive applications, the SPEC instruction cache miss rates are far lower.



(a) ITLB Miss Rate Per 1000 Instructions



(b) DTLB Miss Rate Per 1000 Instructions

Fig. 4: TLB Miss Rates. The interactive applications have drastically worse instruction TLB performance as compared to SPEC as a result of their heavy reliance on calls to shared libraries, which tend to increase ITLB pressure.

rates are comparatively high, the penalty for a branch prediction is lower in embedded-class CPUs than server-class CPUs because of their far smaller instruction windows. Nevertheless, significant opportunities may exist to improve embedded CPU performance by integrating more capable branch predictors.

B. Instruction Cache Miss Rates

Although CPU caches have scaled with process technology, the application code footprints have also increased, and in many cases code size has increased faster than cache sizes. This code size increase leads to poor instruction cache miss rates in many modern interactive applications [31].

We report instruction cache performance in terms of the number of instruction cache misses per one thousand instructions. The instruction cache miss rate is the most striking difference we observe between the SPEC benchmarks and the interactive smartphone applications. As shown in Figure 3(a), the instruction cache miss rates for the interactive smartphone applications are several times higher than in any of the SPEC benchmarks in our subset. Because SPEC benchmarks have been explicitly designed for ease of portability, their code footprints tend to be far more compact, and rely on fewer operating system services, than complete interactive applications. Moreover, SPEC applications lack a user interface, which comprises a substantial fraction of the code footprints of the interactive applications. As a result, SPEC benchmarks place insignificant stress on instruction cache capacity. In contrast, the interactive applications overwhelm the instruction side of the memory system of the smartphone-class processor. Hence, there is substantial opportunity to improve smartphone performance through larger instruction caches and/or better instruction prefetching.

C. Instruction TLB Miss Rates

We report instruction TLB performance in terms of the number of instruction TLB misses per one thousand instructions. Figure 4(a) shows that, much like the instruction cache miss rates, the instruction TLB miss rates in the interactive smartphone applications are significantly higher than SPEC. Only omnetpp has an instruction TLB miss rate similar to any of the interactive applications.

These results indicate the smartphone applications' frequent calls to short functions and heavy reliance on shared libraries (which are typically aligned on page boundaries and result in frequent crossings of page boundaries). The SPEC benchmarks, on the other hand, spend much of their time in tight

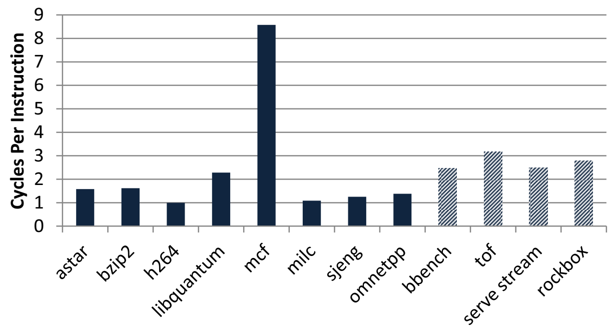


Fig. 5: Cycles Per Instruction. In general, the CPI of the interactive applications is worse than the SPEC CPU2006 applications.

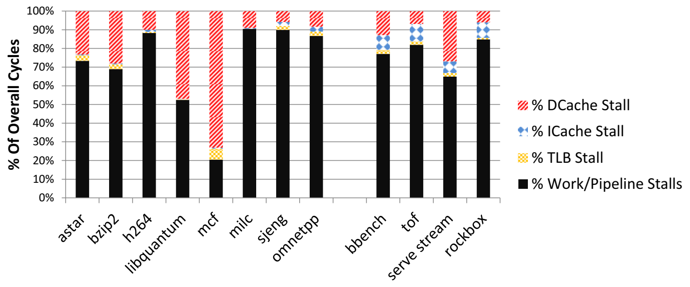


Fig. 6: Cycle Usage Breakdown. The SPEC benchmarks spend their time either doing useful work or waiting on data. The interactive applications have more varied cycle utilization.

loops.

It is worth noting that omnetpp is the only SPEC benchmark we evaluated that utilizes a variety of shared libraries and spends a significant amount of execution time inside these libraries, as shown in Table III. Although ServeStream spends a significant portion of execution time inside shared library calls, it relies on comparatively fewer libraries and hence incurs fewer page crossings (and TLB misses) than the other benchmarks. Rockbox spends only a small fraction of time inside shared library calls so it is not surprising that it has a better instruction TLB miss rate than both BBench and ToF.

D. Data Cache and TLB Miss Rates

Given the scale of interactive applications, we expected that they should have large data footprints and thus exhibit poor data cache and data TLB characteristics. We report data cache and data TLB performance in number misses per one thousand instructions in Figure 4(b) and Figure 3(b), respectively. Contrary to our expectations, we find their data cache and TLB performance surprisingly good; on par with the typical behavior of the SPEC applications and far better than the most memory-intensive applications. This result is interesting because, although the interactive smartphone applications seem to have larger and less predictable code paths than the SPEC bencharmks, their data access patterns exhibit good locality on the average. Our result here suggests that embedded CPU designers have struck good balances in selecting data cache and data TLB capacities.

E. System Interaction

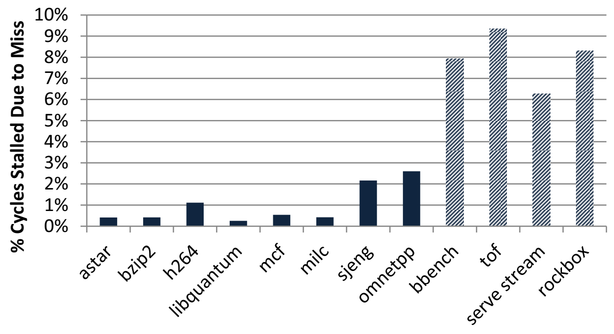
Much like interactive applications on the desktop, smartphone applications comprise a myriad of software layers including drivers, daemons, system calls, and shared libraries. A key result of our study is that these software layers play a prominent role in the execution time, and hence microarchitectural characteristics, of the smartphone applications. To demonstrate the importance of the various system layers, we report the fraction of execution time each benchmark spends in user code, in operating system code, and in shared libraries in Table III. These results demonstrate that the amount of time spent in system calls and shared libraries is significantly greater in the interactive smartphone applications as compared to the SPEC benchmarks. As previously noted, it is not surprising that SPEC benchmarks spend little time outside of user code, as they are designed to ensure high portability, which precludes extensive external dependencies. Nevertheless, it is worthy to note that increased code size [31] is as prevalent on smartphones as on more capable platforms. As with the desktop, the need to improve programmer productivity, increase functionality, and maximize maintainability drive the use of high-level languages, numerous software layers and rich libraries; these concerns trump worries over the size of application code footprints.

F. CPI and Cycle Usage Breakdown

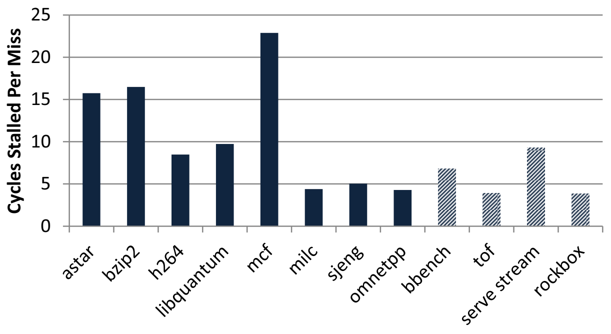
Given our previous branch, cache, and TLB characteristics, we would like to see how these characteristics affect application performance. We calculate and report the average cycles per instruction (CPI) for each application in Figure 5. (Recall that the instruction count used to derive these CPIs includes both committed instructions and wrong-path instructions that are renamed but ultimately squashed). In general, the interactive smartphone applications spend more of execution time

	Benchmark	User	System	Shared Lib
SPEC 2006	astar	99 %	<0.01 %	0.49 %
	bzip2	99 %	<0.01 %	0.25 %
	h264	99 %	<0.01 %	0.62 %
	libquantum	99 %	<0.01 %	0.83 %
	mcf	99 %	<0.01 %	0.20 %
	milc	99 %	<0.01 %	0.04 %
	omnetpp	56 %	<0.01 %	43.10 %
	sjeng	99 %	<0.01 %	0.51 %
Interactive	BBench	45 %	5 %	50 %
	Rockbox	88 %	9 %	3 %
	ServeStream	16 %	6 %	78 %
	ToF	43 %	15 %	42 %

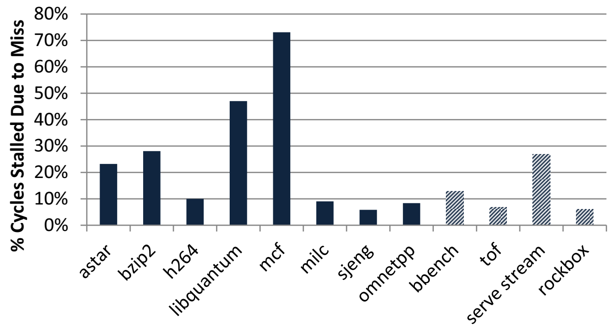
TABLE III: Shared Library and System Usage. By design, the SPEC benchmarks do not use many shared libraries or OS features. In contrast, the interactive applications make significant use of shared libraries and system calls.



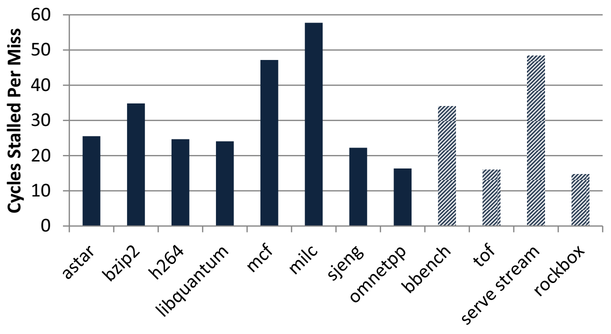
(a) % Stalled Due to ICache Misses



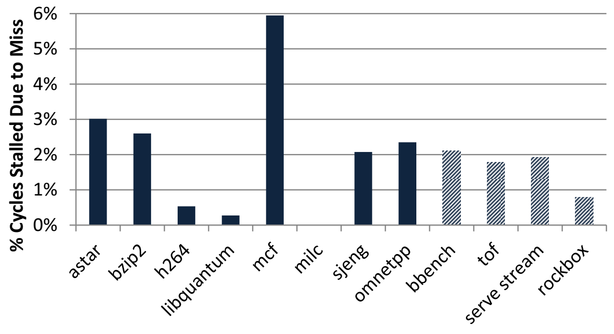
(a) Number of Stall Cycles Per ICache Miss



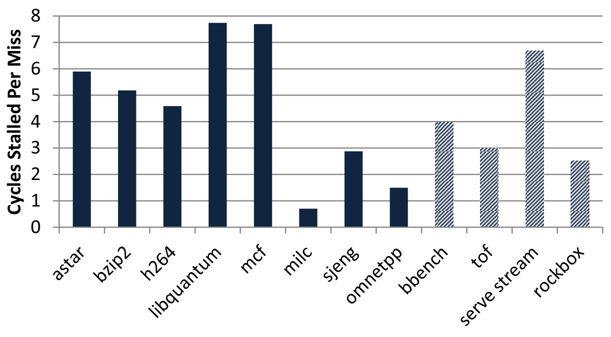
(b) % Stalled Due to DCache Misses



(b) Number of Stall Cycles Per DCache Miss



(c) % Stalled Due to TLB Misses



(c) Number of Stall Cycles Per TLB Miss

Fig. 7: % Cycles Spent Stalled Due to Misses. The SPEC benchmarks spend a significant portion of time waiting on data cache misses, whereas the interactive applications spend a larger portion of time waiting on instruction cache misses.

Fig. 8: Average Stall Penalty Per Miss. Whereas the SPEC benchmarks have lower miss rates in general, the average stall cycles incurred per miss is typically as bad or worse than the interactive benchmarks, indicating SPEC benchmarks suffer from few, but long-latency, misses (e.g., compulsory misses).

stalled than the SPEC benchmarks. The CPI for all of the interactive smartphone applications is close to 3, but with the exception of mcf and libquantum, the SPEC benchmarks exhibit CPIs close to 1.

To determine the cause of the higher CPIs, we construct time breakdowns for each application illustrating the relative fraction of execution time spent on TLB, instruction, and data cache stalls and useful execution, shown in Figure 6. (The available performance counters do not allow us to distinguish busy cycles from cycles with resource stalls or cycles lost due to the execution of wrong-path instructions). These results indicate that the SPEC benchmarks spend their time either doing useful work or stalled waiting for data, whereas the interactive applications are more diverse and spend a far greater portion of time on instruction and TLB misses, corroborating our earlier findings.

We provide greater detail on each of the stall categories of Figure 6 in Figure 7. Each graph shows the percentage of cycles spent stalled due to a particular miss event (note the difference in the scales of the vertical axes). Figure 7(b) shows the large fraction of data stalls in the SPEC benchmarks, while instruction stalls are far more prominent in the smartphone applications, as seen in Figure 7(a).

Figure 8 shows the average latency for the various miss events. It is interesting to note that the average number of stall cycles due to instruction cache misses is comparable, and in many cases worse, for the SPEC benchmarks when compared to the interactive smartphone applications used in this study. This result indicates that the few instruction misses that occur in a SPEC apps tend to fetch the instruction from main memory (likely compulsory misses), whereas the numerous instruction misses in the interactive smartphone applications are satisfied by the L2 cache. However, because instruction cache misses are far less frequent in SPEC, they contribute little to overall execution time.

V. Related Work

A. Benchmark Development

One of the most important aspects when evaluating a new CPU feature or architectural design is to have benchmarks that are representative of the real workloads that will eventually be run on these systems. For embedded systems, the most widely used benchmark suite in the research community is MiBench [15]. This benchmark suite, however, was developed during a different era of embedding computing. Embedded CPUs are no longer simple processors designed to execute straight-forward tasks inside devices such as microcontrollers. Instead, the latest CPUs found in advanced consumer devices, such as smartphones and tablets, are quite sophisticated. The applications run on these platforms have kept pace with the increasing computing power and are becoming more like desktop PC applications.

There are several existing benchmarks for web-browser performance that bear similarity to BBench (e.g., [10], [18]), however, these suffer from compatibility issues and/or do not stress the latest web broswer features and standards. Moreover, EEMBC [10] is proprietary and is not freely available to academics.

B. Real Workload Analysis

Researchers have always known about the importance of using benchmarks that are representative of the real workloads that will eventually be run on the systems they are evaluating. Because of this, extensive research has often been done to characterize the most widely used synthetic benchmarks and validate them against the real workloads they are meant to represent.

In [21] the authors characterized and validated the most popular CPU benchmarks of their era, the SPEC CPU95 benchmarks, on the most popular consumer platform at the time, a desktop PC running Windows NT. In their results, the authors concluded that the SPEC 95 benchmarks were representative of desktop/engineering workloads. Although that study noted many of the same differences between real workloads and the SPEC benchmarks that we have noted, their results did not indicate that these differences lead to diverging microarchitectural conclusions as we have found in our study. In [20], the authors studied embedded Java applications and compared them against the leading Java benchmarks, SPEC and Decapo. Their results are consistent with ours.

Several studies have looked at the thread-level parallelism present in interactive applications [8], [12]. These studies show that the available thread-level parallelism hasn't increased much in over ten years; nevertheless, modern interactive workloads can still effectively utilize several cores. We observe similar trends in our smartphone applications. As can be seen in Figure 1(b), BBench spawns numerous threads and effectively utilizes two cores.

VI. Conclusion

In this paper we have characterized interactive smart phone applications, and shown how they differ significantly from SPEC CPU2006 benchmarks. Smartphones have become the primary consumer computing device. As computer architects work to improve the efficiency and performance of such systems they need to consider benchmarks that better match typical use cases along with their library and system interactions. In particular, the interactive smart phone applications have significantly worse instruction cache, TLB miss statistics and branch misprediction rates, which we attribute to heavy use of high level software abstractions. To this end, we have developed an interactive smartphone benchmark suite that includes a web-browser benchmark, BBench, which is representative of the most ubiquitous smartphone application—the web-browser. BBench provides a fully contained, automated and repeatable web-browser benchmark that exercises not only the web-browser, but the underlying libraries and operating system. While there is a place for CPU-intensive portable benchmarks, computer architects and systems designers will be ill-served relying solely on these benchmarks for performance improvement and characterization in the mobile device arena.

Acknowledgements

The authors would like to thank the anonymous reviewers for their feedback. We would like to thank Geoff Blake for his advice on tools and background information. We would also like to thank Joseph Pusdesris for his advice on passing arguments through the webpage's URL, the sustenance he provided during many late hours in the lab, and for being an inspiration to us always. This work was supported by a grant from ARM.

References

[1] A. R. Alameldeen, M. M. K. Martin, C. J. Mauer, K. E. Moore, M. Xu, D. J. Sorin, M. D. Hill, and D. A. Wood. Simulating a $2M Commercial Server on a $2K PC. *IEEE Computer*, 36(2):50-57, 2003.
[2] Apple OS X Lion. http://www.apple.com/macosx/
[3] ARM Architecture Reference Manual: ARM v7-A and ARM v7-R Edition.
[4] ARM Cortex-A9. http://www.arm.com/products/processors/cortex-a/cortex-a9.php
[5] ARM Cortex-A15. http://www.arm.com/products/processors/cortex-a/cortex-a15.php

[6] ARM Streamline Performance Analyzer. http://www.arm.com/products/tools/software-tools/ds-5/streamline.php

[7] N. L. Binkert, R. G. Dreslinski, L. R. Hsu, K. T. Lim, A. G. Saidi, and S. K. Reinhardt. The m5 simulator: Modeling networked systems. *IEEE Micro*, 26(4):52-60, 2006.

[8] G. Blake, R. G. Dreslinksi, T. Mudge, and K. Flautner. Evolution of Thread-Level Parallelism in Desktop Applications. *Proceedings of the 37th Annual International Symposium on Computer Architecture*, pages 302-313, 2010.

[9] Cortex-A9 Technical Reference Manual.

[10] EDN Embedded Microprocessor Benchmark Consortium. Browsing-Bench. http://www.eembc.org/

[11] Firebug Firefox Plugin. http://getfirebug.com/

[12] K. Flautner, R. Uhlig, S. Reinhardt, and T. Mudge. Thread-Level Parallelism and Interactive Performance of Desktop Applications. *Workshop on Multi-Threaded Execution, Architecture, and Compilation*, 2000.

[13] Google's 1000 most-visited sites on the web. http://www.google.com/adplanner/static/top1000/

[14] Google Chrome OS. http://www.chromium.org/chromium-os

[15] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. MiBench: A free, commercially representative embedded benchmark suite. *IEEE 4th Annual Workshop on Workload Characterization*, pages 3-14, 2001.

[16] J. L. Henning. SPEC CPU2006 Benchmark Descriptions. *SIGARCH Computer Architecture News*, 34(4) :1-17, 2006.

[17] HTTrack Website Copier. http://www.httrack.com/

[18] iBench. ftp://ftp.pcmag.com/benchmarks/i-bench/

[19] IDC Worldwide Quarterly Mobile Phone Tracker 2011. http://www.idc.com/

[20] C. Isen, L. John, J. P. Choi, and H. J. Song. On the Representativeness of Embedded Java Benchmarks. *IEEE International Symposium on Workload Characteriztion*, pages 153-162, 2008.

[21] D. C. Lee, P. J. Crowley, J.-L. Baer, T. E. Anderson, and B. N. Bershad. Execution Characteristics of Desktop Applications on Windows NT. *SIGARCH Computer Architecture News*, 26(3):27-38, 1998.

[22] Nielsen. http://blog.nielsen.com/nielsenwire/?p=27418

[23] NVIDIA Tegra 2. http://www.nvidia.com/object/tegra.html

[24] Rockbox. http://www.rockbox.org/

[25] M. Rosenblum, S. A. Herrod, E. Witchel, and A. Gupta. Complete Computer System Simulation: The SimOS Approach. *IEEE Parallel and Distributed Technology: Systems and Applications*, 3(4):34-43, 1995.

[26] ServeStream. http://sourceforge.net/projects/servestream/

[27] SPEC CPU2006 benchmark suite. http://www.spec.org/cpu2006/

[28] C. Sudanthi, M. Ghosh, K. Welton, and N. Paver. Performance Analysis of Compressed Instruction Sets on Workloads Targeted at Mobile Internet Devices. *IEEE International SOC Conference*, pages 215-218, 2009.

[29] Taps of Fire. http://code.google.com/p/tapsoffire/

[30] Texas Instruments. http://www.ti.com/omap

[31] R. Uhlig, D. Nagle, T. Mudge, S. Sechrest, and J. Emer. Instruction Fetching: Coping with Code Bloat. *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, pages, 1995.

[32] T. F. Wenisch, R. E. Wunderlich, M. Ferdman, A. Ailamaki, B. Falsafi, and J. C. Hoe. SimFlex: Statistical Sampling of Computer System Simulation. *IEEE Micro*, 26(4):18-31, 2006.

[33] YouTube Mobile Player. m.google.com/youtube