A Structured Approach to the Simulation, Analysis and Characterization of Smartphone Applications

Dam Sunwoo, William Wang, Mrinmoy Ghosh, Chander Sudanthi,
Geoffrey Blake, Christopher D. Emmons, Nigel C. Paver
ARM R&D
{dam.sunwoo, william.wang, mrinmoy.ghosh, chander.sudanthi,
geoffrey.blake, chris.emmons, nigel.paver}@arm.com

***Abstract*—Full-system simulators are invaluable tools for designing new architectures due to their ability to simulate full applications as well as capture operating system behavior, virtual machine or hypervisor behavior, and interference between concurrently-running applications. However, the systems under investigation and applications under test have become increasingly complicated leading to prohibitively long simulation times for a single experiment. This problem is compounded when many permutations of system design parameters and workloads are tested to investigate system sensitivities and full-system effects with confidence.**

In this paper, we propose a methodology to tractably explore the processor design space and to characterize applications in a full-system simulation environment. We combine SimPoint, Principal Component Analysis and Fractional Factorial experimental designs to substantially reduce the simulation effort needed to characterize and analyze workloads. We also present a non-invasive user-interface automation tool to allow us to study all types of workloads in a simulation environment. While our methodology is generally applicable to many simulators and workloads, we demonstrate the application of our proposed flow on smartphone applications running on the Android operating system within the gem5 simulation environment.

I. Introduction

Computer architects are challenged to reason about the intertwined and obtuse behavior that emerges from full-system hardware and software interactions. Connecting these pieces together and understanding the interplay between them is essential for analyzing real systems [1] and proposing highly optimized micro-architectures. Simulators such as gem5 [2] which run full operating systems and software stacks are suitable platforms for early exploration of candidate architectures and their potential effects on software.

Although simulating full software stacks on detailed CPU models is insightful, a frequently encountered problem with detailed simulation is prohibitively long run times for each benchmark. Additionally, early modeling of candidate micro-architectures requires exploring an intractably large design space. These problems severely limit the amount and quality of data that can be generated from simulation. This bottleneck often forces architects to rely heavily on intuition and domain specific knowledge of workloads to guide design decisions instead of empirical data. Lack of tractable experiment and simulation methodology also often limits investigations to non-representative benchmarks.

There has been extensive work in tackling the problem of simulation length. Modeling methods that raise the level of abstraction to achieve more reasonable simulation times [3] is one example. However, this approach is limited; while it may allow one to more quickly simulate full workloads, it does not solve the problem of simulating that workload on a detailed model when an architect is negotiating finer details and trade-offs. A more favorable approach to increase analysis throughput is to use SimPoint [4] to distill out the important and often repetitive phases of benchmarks, experiment on those phases, and then project workload performance from that small subset of the workload. This approach allows an architect to simulate effectively full workloads on any type of micro-architecture model, regardless of detail, in a more tractable amount of time. This methodology has not been proven on real applications running on top of operating systems, and, in isolation, this technique does not address the large design space exploration problem for analyzing different hardware structures, algorithms, and arrangements. Exhaustive design space exploration with detailed models can be intractable for even modest numbers of workload SimPoints and experimental hardware designs.

Simulating relevant applications and workloads is also challenging due to user interaction that may need to be modeled especially for emerging smartphone applications. User interface automation tools are the obvious answer, but they have many shortcomings. One typical problem is that the tools may change the behavior of automated programs because they rely on timestamps to determine when to feed inputs to the system. In a simulated system where an architect is exploring different system parameters, a dependence on timing for automating interaction will not work because the systems will vary in performance. A user automation technique that is noninvasive and able to deal with time dilation is needed to enable the study of interactive applications.

We address these issues by demonstrating a new analysis flow using a combination of well-established methodologies and statistical tools in an effort to better characterize contemporary, full-system smartphone applications and behavior in a structured manner. Specifically, we contribute:

- A structured, tractable, and generic flow for detailed design space exploration of contemporary applications in a full-system simulation environment combining SimPoint, Principal Component Analysis and Fractional Factorial experimental design to substantially reduce simulation time
- A generic, noninvasive, and time-dilation-tolerant GUI automation tool for capturing and replaying GUI input
- Application of the flow to study emerging smartphone workloads

Section II describes our flow in detail. Section III and Section IV present a case study of applying our analysis

978-1-4799-055-3/13/$31.00 ©2013 IEEE

methodology to contemporary smartphone applications. We then present related work and conclude.

II. Structured Methodology

A. Overview

As stated, simulation is the preferred path to investigate new and interesting micro-architectures. Simulations provide the necessary flexibility in terms of modeling detail and the ability to run modern workloads. Modern workloads, however, require full system support and are often interactive making it difficult to repeat simulation in a deterministic fashion. They can also have long run times making it very expensive to exhaustively explore the design space especially when there are many system parameters to consider. A structured methodology that could simultaneously (i) ensure deterministic simulation on emerging workloads, (ii) reduce the required runtime of individual workloads, and (iii) systematically narrow down the design space parameters to explore is desirable.

This section outlines our methodology that provides all these desirable characteristics. The flow is comprised of four key components: GUI automation, SimPoints, principal component analysis, and fractional factorial designs. These components are all well-established methodologies on their own, and some are already widely used for a variety of research. The structured flow that combines these methodologies and the application of them to a contemporary workload suite is novel.

- **GUI Automation**: Our *AutoGUI* framework allows full-system interactive workloads to be simulated in a deterministic fashion.
- **SimPoint**: Once we enable deterministic simulation, we use SimPoint to identify short key phases of workloads that can be used to project full run behavior. We perform detailed simulation on just those phases to significantly reduce the simulation time for each workload. We show that SimPoint works well for our full-system smartphone workloads.
- **Principal Component Analysis (PCA)**: We apply PCA to contrast our workloads and workload phases from each other and from traditional benchmark suites. PCA exposes the characteristics that make them different.
- **Fractional Factorial Design**: Fractional Factorial Design significantly narrows down the number of experiments required to explore a large design space while maintaining statistical integrity.

Our proposed flow is summarized in Figure 1. Each methodology in the flow will be described in greater detail in the remainder of this section.

While our methodology is generic enough to be used with any simulator, we use gem5 [2] with the ARM architecture. gem5 has great flexibility in configuring various system parameters. It supports a full-system environment and peripherals so that unmodified operating systems can boot. The ARMv7 architecture models are stable and support the latest Linux and Android distributions. gem5 has a fast functional CPU model (atomic CPU) and a detailed out-of-order CPU model (O3 CPU).

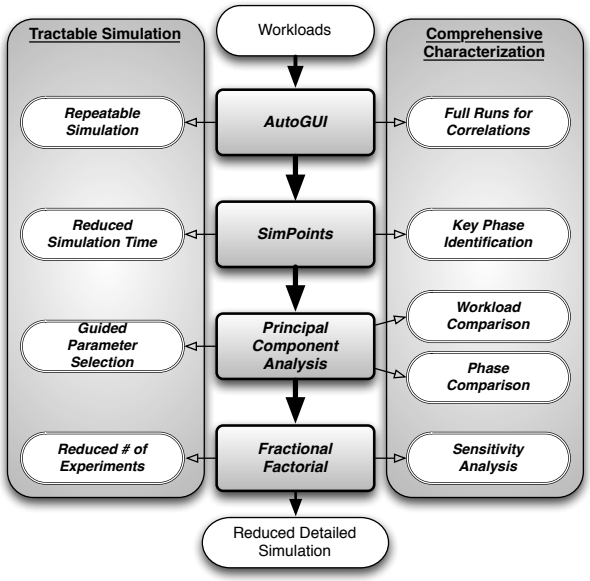

Fig. 1: Flowchart of our proposed methodology.

B. GUI Automation for Deterministic Replay

A major difficulty when analyzing Android applications is generating reproducible results without manual user input. This problem can be solved with software automation tools like Xnee [5], Robotium [6], and MonkeyRunner [7]. However, slow execution speeds make it impractical to test the automation in simulators. It is also desirable to have an automation framework that is portable across multiple applications, operating systems (OS's), boards, simulators, and emulators. Maintenance costs increase and ease of use decreases with each additional framework. When unique implementations are necessary for each application and OS, the time to develop a suite of automated applications also increases. The tool we developed and will make publicly available, called AutoGUI, addresses these problems.

Recording and replaying time-synchronized input on different machines, simulated or real, can be problematic. For example, the load screen for a game will appear later on a slower machine than on a faster one. When replaying time-synchronized input on these systems, the interaction with the game after the load screen will replay earlier on the slower machine than on the faster one. To address this issue, AutoGUI allows users to also synchronize on frame buffer images.

Xnee, an automation tool for Linux, records user input and the X11 windowing system events that occur before the user input. During replay, X11 events are used to synchronize user input playback. The use of X11 events is specific to the software stack running on top of the Linux kernel. Robotium and Google's MonkeyRunner are automation tools tied to the Android OS. Robotium is additionally tied to the application, requiring the user to write an automation script using knowledge of the application layout, nomenclature, and the Android API. MonkeyRunner is more general, replaying mouse and keyboard events rather than application-specific events, but uses time to synchronize input. Furthermore, manually writing automation files for Robotium and MonkeyRunner is a tedious process.

AutoGUI addresses the shortcomings of existing solutions

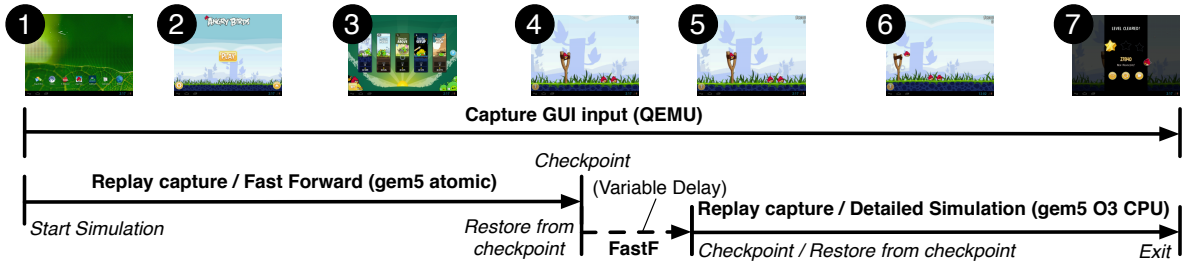

Fig. 2: Capture and Replay of Angry Birds with AutoGUI on gem5

and meets all of our requirements. AutoGUI is not dependent on the software platform because it operates on the Remote Framebuffer Protocol which many OS's support via Virtual Network Computing (VNC). Many emulators and simulators natively use VNC to allow users to interact with the emulated or simulated GUI. VNC is also easily installed onto devices. AutoGUI creates automation files on-the-fly by capturing touchscreen and keyboard events as the user interacts with an application without any specific software dependencies.

The AutoGUI methodology primarily consists of two parts, capture and replay. Though AutoGUI is portable across multiple platforms, for our studies we perform the automation capture on the Quick EMUlator (QEMU) [8] and perform the automation replay on gem5. We use QEMU because the emulation speed is much faster than gem5 atomic and O3 models. AutoGUI connects to a QEMU instance running the target OS. Users instruct AutoGUI when to start and stop recording and what inputs and reference points to capture as they interact with applications.

AutoGUI does require that both the replay and capture environment are similar. Specifically, we kept the display resolution and file systems the same in both gem5 and QEMU. Though not strictly necessary, using the same filesystem allowed us to install applications and to initialize them by running them once on the faster QEMU environment.

As shown in Figure 2, the popular game Angry Birds has been automated with AutoGUI. The steps involved with starting Angry Birds and interacting with it are shown in the figure: ❶ boot Android and click through to the Angry Birds application icon, ❷—❸ click through the initial screen and select a game level to play, ❹—❺ wait for the application to pan the level and rest on the screen ready for player input, ❻ pull back and release the angry bird, and, assuming only one bird fling is necessary to complete the game level, ❼ wait for the level to complete and display a score.

Streams captured from QEMU are fast forwarded on the gem5 atomic model to the critical portion of the workload and continue on the detailed O3 model. Most of the replay involves syncing on frame buffers and replaying recorded touchscreen inputs. However, in order to analyze run-to-run variability common in real applications, special events in the AutoGUI capture support arbitrary insertion of time delays. In Angry Birds, a checkpoint event recorded at ❹ notifies gem5 to checkpoint its state. The checkpoint is then resumed on multiple instances of the gem5 atomic model, each varying a runtime delay parameter. After idling for the prescribed time, another checkpoint is automatically taken. With this methodology, multiple post-delay checkpoints are created; a run from each post-delay checkpoint will exhibit slightly different behavior just as different test runs would in a real system. These checkpoints are then replayed on the gem5 O3 model at ❺. The replay continues until ❼ when a final AutoGUI exit event is replayed causing the simulator to stop.

C. SimPoints

Once we are able to automate our workloads, we generate SimPoints [9] for them. SimPoint is a widely-used methodology that gathers small, representative samples of a workload and uses them to project characteristics of the complete execution of the workload. SimPoint commonly uses Basic Block Vectors (BBV's) that keep track of the frequency of basic block execution to represent an interval. Once the BBV's are generated, SimPoint uses Euclidean distance to compare the vectors and applies the K-means clustering algorithm to group them. The Bayesian Information Criterion (BIC) is used to determine the quality of the clusters and, thus, the optimal number of SimPoints. Finally, one SimPoint per cluster is chosen to form the SimPoints that represent the full workload.

While SimPoint has been commonly used with traditional bare-metal benchmarks such as the SPEC benchmark suites, the use of it in a more complex full-system environment with operating systems and interactive applications has not been shown or verified. We generate SimPoints for our chosen smartphone applications and empirically show that they still have good accuracies for such workloads.

In addition, while SimPoints are traditionally used for projecting the full-run characteristics with shorter samples, we also use them to characterize the phase behavior of the workloads. Once the key phases are identified, we examine how distinct the phases are within the same workload and how similar some phases are across workloads.

D. Principal Component Analysis (PCA)

The next step in our methodology is to apply PCA on our workloads. PCA [10] is a mathematical procedure that is used on a large data set to distill out its *principal components*, a set of linearly uncorrelated variables. The principal components are generated so that the first principal component accounts for as much of the variability in the data as possible, and each succeeding component accounts for as much of the remaining variability as possible. It is thus useful for quickly extracting the similarities and dissimilarities of characteristics within the data set. PCA has been recently used in workload characterization of SPEC2000 [11], [12] and SPEC2006 [13] benchmark suites. In this work, we employ PCA (i) to contrast our smartphone workloads against traditional benchmark suites and also (ii) to contrast different phases in individual workloads using SimPoints.

As the principal components are sorted by variance, the first few components are generally enough to get an idea of how similar or dissimilar the data points are. An oft-used method of visualizing the result is to generate a scatter plot with the first two principal components as the X and Y axes, respectively. Sections IV-B and IV-C provide concrete examples of such plots and analyses. Each axis will be an orthogonal linear sum of the original observed variables. For example, if two data points on this chart have a similar X value but significantly different Y value, one can expect that the variables used in the X axis are similar while the difference lies in the variables of the Y axis. Thus, analysts can quickly get an overview of the large data set without having to compare the individual data points for all the variables.

E. Fractional Factorial Design

Even with a reduction in the number and length of benchmarks using PCA and SimPoints, exploring the design space of a modern microarchitecture is difficult due to the large number of parameters that need to be tested for the benchmarks of interest. Brute force testing of all possible parameter configurations, known as a full-factorial experiment, quickly leads to an intractable number of simulations to perform and analyze.

To reduce the number of simulations to perform, we utilize a rigorous design of experiments technique called *Fractional Factorials*. Fractional factorial designs [14] were created by Sir Ronald Fisher in the 1920's when researching statistical techniques for agricultural research. Fractional factorial designs use a small subset of the full-factorial design space. Using fewer experiments retains validity of the results by reformulating the question asked of the experiments. Instead of trying to determine directly "what parameters are important and how much improvement do they provide the micro-architecture?", the question is simplified to "what parameters are important?". Fractional factorials are used in our flow to answer this question first, prior to extensive simulation, by screening for the parameters that significantly impact performance.

Fractional factorials work by varying all parameters simultaneously in a fashion that is balanced, orthogonal, and unbiased. This allows strong conclusions to be drawn about whether a parameter makes a statistically significant impact on performance and should be investigated further. Figure 3 illustrates this property. The cube visualizes the full factorial set of experiments for three parameters which map to the axes of the cube. Each vertex represents each permutation of settings for each parameter. The permutations are represented by the tuple of $-$'s and $+$'s. These $+$'s and $-$'s represent the high and low settings for each variable being tested in the design. Fractional factorial designs are meant as screening experiments, so more than two settings per parameter is not needed. For example, in Figure 3 one might select 32kB as the $-$ setting and 128kB as the $+$ setting for the DL1 Size parameter to screen whether a benchmark sees DL1 Size as an important parameter. As shown, the fractional factorial experiment selection of parameter settings represented by the green-shaded polygon is balanced across the experiment space. In fact, each parameter is set to $-$ and $+$ an equal number of times. This balanced construction makes fractional factorials easy to analyze. Because each parameter is set an equal number of times to $-$ and $+$, we can use the following equation to

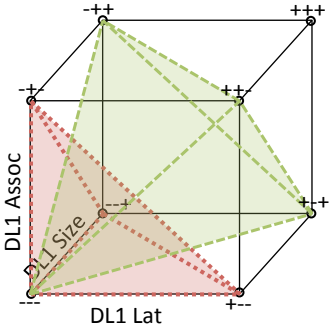

Fig. 3: Visualization of the distribution of experiments in a 3 dimensional design space using a common one-parameter-at-a-time approach (red) and a fractional-factorial design (green).[16]

estimate the *Importance* of each parameter:

$$|x_i| = \sum_{+} r_n - \sum_{-} r_n \tag{1}$$

This equation states, for each parameter x_i, sum up the responses of the system, r_n, when the parameter is set to $+$ and subtract the sum of the responses when the parameter is set to $-$ to get an estimate of the importance of that parameter. In contrast, the traditional vary-one-parameter-at-a-time experiment, represented by the red-shaded polygon in Figure 3, is heavily biased towards setting parameters as $-$ (a setting of $+$ is only selected once per parameter) and cannot be analyzed using equation 1.

The following caveats must be accounted for when using fractional factorials in our flow. First, a low $-$ and high $+$ setting for each parameter to be screened must be selected. One must be careful in the selection of the $+$ and $-$ settings as too narrow of a range may mask a parameter's importance while too large of a range may exaggerate a parameter's importance. We will see in Section IV-D an example of this hazard. Since only two parameter settings are used, fractional factorial experiments assume that moving from $-$ to $+$ will produce a generally monotonic response. For our experiments, this was seen as a valid assumption.

Second, fractional factorials assume that high-order effects, combinations of three or more parameter settings leading to a response greater than the sum of it parts, are small compared to main effects. Fractional factorials achieve a small number of experiments by *confounding* high-order effects with the main effects. More information on confounding can be found in [15]. If high-order effects are suspected, the only solution to determine their impact is to run more experiments. In our experience with working on fractional factorials for micro-architecture design space exploration, high-order effects between three or more parameters are in fact small.

In summary, the following steps are used to apply fractional factorial designs in our workload evaluation methodology:

1) Choose the desired parameters to test.
2) Choose the low ($-$) and high ($+$) settings for each parameter, being careful to pick an approriate range between low and high.
3) Pick the number of experiments to run, also known as the *Resolution* of the design [15], to account for any suspected higher-order effects.

TABLE I: Baseline system parameters and ranges for fractional factorial designs (for Section IV-D).

Parameters	Default	Low (−)	High (+)
Integer RegFile Size	128	80	196
FP RegFile Size	128	80	196
Inst Queue Size	32	8	128
ROB Size	40	20	196
ALU	Nominal	Nominal	Large
Load Queue Entries	16	8	64
Store Queue Entries	16	8	64
DTLB Size	64	8	128
ITLB Size	32	8	128
BP Sizes	Small	Small	Large
L1 I-Cache Size	32kB	8kB	64kB
L1 I-Cache Associativity	2	2	8
L1 D-Cache Size	32kB	8kB	64kB
L1 D-Cache Latency	2 cycle	4 cycles	2 cycles
L1 D-Cache Associativity	2	2	8
L2 Bus Width	16	32	64
L2 Cache Size	1MB	64kB	8MB
L2 Cache Latency	12 cycles	30 cycles	12 cycles
L2 Cache Associativity	16	16	32
L2 Prefetcher Type	Stride	None	Stride
Memory Bus Width	16	16	32
Memory Bus Clock	1GHz	1GHz	2GHz
Memory Latency	54ns	108ns	20ns

4) Perform the experiments.
5) Use equation 1 to estimate the impact of each parameter. Pick the top n parameters with the largest impact to use in further experiments.

After the top parameters are found, additional experiments can be conducted on these parameters to answer remaining questions asked in a design space exploration such as "how much performance gain is expected?" or "where are the inflection points for a parameter?".

III. Infrastructure and Workloads

A. Infrastructure

We modified the gem5 simulator to add the capability to generate basic block vector profiles for SimPoint generation. Basic block profiling and checkpoint generation are conducted in the atomic mode of gem5 which provides the fastest execution times. We then resume from the SimPoint checkpoints using the detailed CPU models for the analyses we want to carry out.

The default column of Table I shows the baseline configuration for the out-of-order (O3) ARMv7 CPU and platform we used as the baseline system. The values chosen are typical settings for a modern microprocessor architecture. For this work, we only investigate uni-processor system configurations.

B. Description of Workloads

To demonstrate our methodology, we use the flow to characterize emerging smartphone applications. Using AutoGUI as described in Section II-B, we created an Android Jelly Bean suite, shown in Table II, that contains a set of smartphone workloads including traditional benchmark applications as well as popular gaming and productivity applications. As the benchmark applications exercise specific components of the Android system, they are useful for in-depth study of specific components of the system, such as the Dalvik virtual machine or SQLite in Android, or components like the CPU, memory and storage in the hardware platform. Using popular applications for analysis helps ensure systems have good overall performance for typical use-cases on smartphones like web browsing, gaming and productivity.

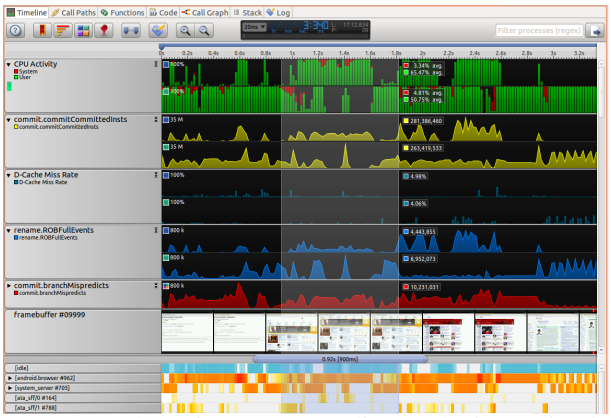

Fig. 4: Screenshot of ARM Streamline analyzing a simulated run of BBench from gem5.

Studying the behavior of popular applications on Android gives architects hints as to where to optimize to improve system efficiency. Analyzing the benchmark applications helps steer investigations of specific components inside the Android system. The workload suite can be enriched with more popular applications, such as social networking, entertainment, navigation or augmented reality applications, with the same GUI automation methodology. They are not included in the current workload suite due to the lack of camera, GPS or necessary network components in the gem5 simulator infrastructure. This peripheral development is an area for future work.

C. ARM Streamline

We also use the ARM Streamline Performance Analyzer [24] that is part of the ARM DS-5 suite. Streamline was originally developed for analyzing execution on real hardware platforms but has been recently adapted to view executions from the gem5 simulator as well [25]. Figure 4 shows an example of Streamline visualizing results of a BBench run generated from gem5.

Streamline visualizes the complex behavior of full-system applications and enhances one's capability to comprehend the temporal behavior of the benchmark. It displays the various gem5 statistics along with the Linux process and thread view at the bottom. It can display the various statistics broken down by process and thread. Streamline also shows frames buffer output in the timeline so that users can have insight into what is happening within a frame or, conversely, what is happening visually when interesting changes in statistics occur.

IV. Detailed Analyses on Workloads

A. Correlation of SimPoints

As our characterization flow relies on the use of SimPoints, we first need to make sure that the SimPoints correlate well with the full runs for our workloads and target platforms. We experimented with various SimPoint lengths, or *interval sizes*, ranging from 100K to 100M instructions and discovered that, for our workloads, 100M instruction intervals provide the best balance between accuracy and reasonable numbers of SimPoints. We designated a maximum of 50 SimPoints ($maxK$) and a 10M instruction warm-up period for each application to keep the storage requirement reasonable. An average of 28 SimPoints were generated per workload. Figure 5 compares the CPIs from the full runs of the benchmarks against those

TABLE II: Description of smartphone workloads.

Workload	Type	Scope	Description	Reported results
AndEBench [17]	Benchmark	Dalvik / CPU	A benchmark application from EEMBC[18] that includes the original CoreMark written in C language and a Java reimplementation of CoreMark. The internal algorithms concentrate on integer operations on linked lists, matrices, and state machines. The workload can be configured to run in single or multiple threads.	Two scores represent the C native and Java performance results in 'iterations per second'.
CaffeineMark [19]	Benchmark	Dalvik	A benchmark application that uses six tests to measure various aspects of Java virtual machine (VM) performance, including Sieve, Loop, Logic, String, Method and Float. Each test runs for approximately the same length of time. The score for each test is proportional to the number of times the test was executed divided by the time taken to execute the test.	Scores for each of its six constituent tests, as well as the overall score that is the geometric mean of the individual scores.
RL Benchmark [20]	Benchmark	SQLite	A synthetic benchmark application that generates a pre-defined number of various SQL queries to test SQLite performance in Android.	The overall score for all the tests as well as individual test scores.
BBench [21]	Benchmark	Browser	A browser benchmark that measures the browsing experience in terms of how fast browsers render web pages. BBench v2.0 loads nine web pages into the browser one by one, it includes the following popular web pages, Amazon, BBC, CNN, Craigslist, eBay, Google, MSN, Slashdot, Twitter, excluding ESPN and YouTube as in the full version.	The load time of each web page as well as the overall load time for all the web pages.
Angry Birds [22]	Gaming	System	A game application used to measure the gaming experience on the Android platform. The workload loads the Angry Birds application, pull the bird back, release a fling shot, hit the pigs, and complete the first level in one hit. Angry Birds is selected due to its popularity, and the application works offline and is free from Google play.	The run time of the workload is measured by reported gem5 statistics.
Kingsoft Office (WPS) [23]	Productivity	System	A productivity application used to measures the load time of office applications such as Word, Powerpoint and Spreadsheet. The workload loads the Kingsoft Office application, and creates Word, Spreadsheet and Powerpoint documents in turn using templates. Kingsoft Office application is selected due to the increasing importance of productivity applications on tablets/smartphones, and the application works offline and is free from Google play.	The run time of the workload is measured by reported gem5 statistics.

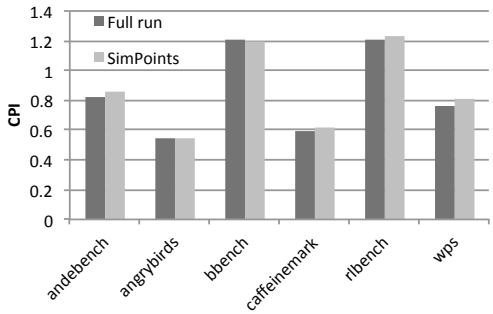

Fig. 5: Correlation of CPIs (Full runs vs. SimPoints).

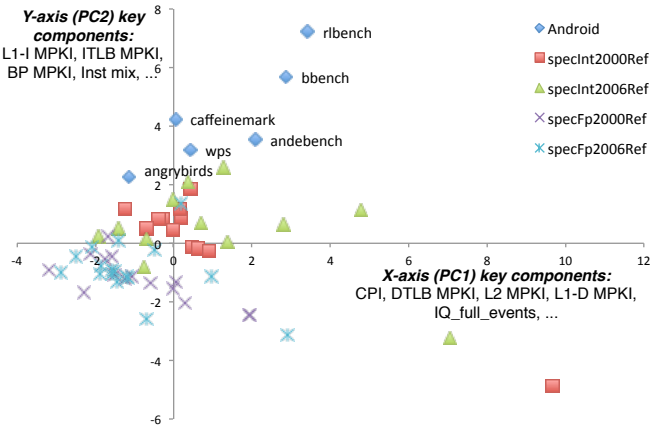

Fig. 6: Top two PCs (PC1 and PC2) from PCA on smartphone workloads and SPEC benchmarks. Each data point represents a benchmark from the corresponding benchmark suite.

projected by using SimPoints. The average error from the full runs is within 2.5%[1].

Previously, SimPoint has mostly been used with traditional bare-metal benchmarks. Its effectiveness has not been well-demonstrated for full-system and interactive benchmarks. We empirically show that SimPoint can be very accurate even for such benchmarks and platforms. Our interval study indicated longer intervals led to better accuracy. This might be due to the fact that longer intervals capture more full system effects (e.g., timer interrupts) and represent them more accurately.

B. PCA for Workload Comparison

After establishing our benchmark suite largely from intuition, we were interested in statistically understanding how our smartphone workloads are different from traditional workloads. We generated SimPoints for the SPEC2000 and SPEC2006 benchmark suites (both INT and FP suites) using the reference input sets. Once we had the detailed simulation results of these SimPoints, we fed the projected full-run gem5 statistics through PCA along with the same statistics from the smartphone workloads.

[1]It might be possible that shorter intervals lead to better accuracies, but this accuracy would come at the cost of significantly more SimPoints.

PCA produces several equations constructing the principal components. In Figure 6, we present the top two principal components, PC1 on the X-axis and PC2 on the Y-axis, grouped by benchmark suite. The closer two data points are on this chart, the more similar their characteristics are. The top factors on the X-axis include CPI, DTLB MPKI, L2 MPKI, and so forth. The top factors on the Y-axis include L1-I MPKI, ITLB MPKI and branch predictor MPKI. It is interesting to note that PCA identifies and separates the data-side and instruction-side statistics out to two separate axes automatically.

In Figure 6, the smartphone workloads are clearly distinguished from the rest of the workloads along the Y-axis. This implies that these workloads differ from the traditional SPEC benchmarks in instruction-side characteristics. This result is in-line with findings from similar studies [21]. The difference,

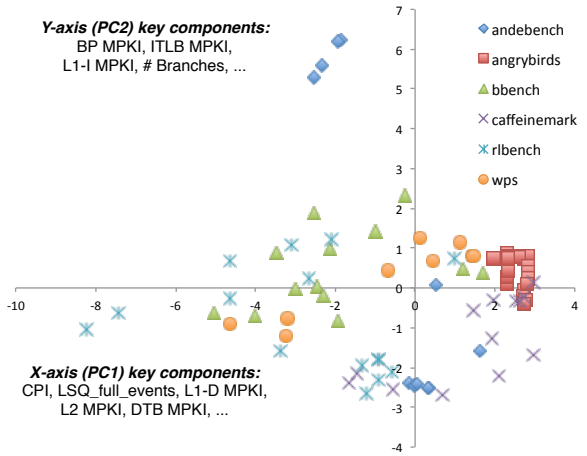

Fig. 7: Top two PCs (PC1 and PC2) from PCA on individual phases of smartphone workloads.

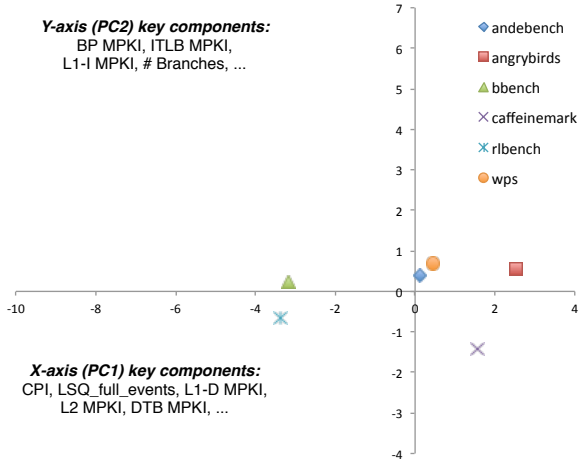

Fig. 8: PCA of projected full runs using weighted SimPoints using the same formula as in Figure 7.

however, is that we did not have to manually analyze and compare individual statistics to lead to this discovery; our methodology revealed this for us. The same methodology can be applied systematically to other emerging workload suites. Based on such analyses, one can probe further by comparing individual statistics identified by PCA or prune similar workloads to reduce simulation time.

C. PCA for Phase Analysis

Workloads often have very distinct phases. To analyze the phase behavior of our workloads, we also applied PCA on individual phases identified by SimPoint for each of the smartphone workloads. This is unlike previous approaches that used PCA for whole-workload characterization [11], [12], [13]. As SimPoint identifies many lightly-weighted phases, we only included those SimPoints with a weight of at least 1% in the analysis.

Figure 7 displays the top two principal components for individual phases in our smartphone workloads. Similar to Figure 6, the X-axis is composed of data-side statistics including CPI, LSQ full events, L1-D MPKI, and L2 MPKI, while the Y-axis is composed of instruction-side statistics such as branch predictor MPKI, ITLB MPKI, and L1-I MPKI. Some workloads like Angry Birds have most phases clustered closely together while other workloads have distinct phases spread across the chart. For example, RLBench phases are widely spread across the X-axis indicating variation in data-side characteristics while AndEBench phases are spread across the Y-axis thus indicating variation in instruction-side characteristics.

For comparison, Figure 8 shows the PCA chart for the projected full runs based on weighted SimPoints. This figure uses the same principal components as the axes generated by the per-phase PCA in Figure 7 and uses the same ranges for those axes. Naturally, each full run is represented by a single data point. Workloads that do not have many distinct phases, like Angry Birds, exhibit behavior that is statistically well-represented by the full-run. On the other hand, AndEBench, for example, has very few phases (in Figure 7) that are close to its projected full run data point (in Figure 8). In this case, analyzing the full-run (or projected full run if using SimPoints) without considering the distinct phases would not accurately capture the nature of the workload.

D. Sensitivity Studies based on Fractional Factorial Designs

For the sensitivity study to identify the most important system parameters for each workload, we use fractional factorial to generate the list of experiments. Table I shows the complete list of system parameters and their ranges (Low and High) we used for this sensitivity study. Figure 9 shows the result of the fractional factorial experiments. The chart shows the sensitivity of each parameter normalized by the maximum impact for each workload. The experiments are generated from SimPoints of each workload, but we also display results for the same experiment using full runs, in dashed lines, for some of the workloads. We can see that the study done using SimPoint tracks the full runs well[2]. This type of sensitivity study is critical for pruning the design space by filtering out the parameters that are not important. This pruning enables the designer to concentrate on only five or six important parameters for a given benchmark rather than work with the complex problem of looking at all the parameters shown in Table I.

To validate the effectiveness of our SimPoint-based fractional factorial tool, we perform detailed sensitivity studies on the BBench and Caffeinemark benchmarks for a select number of parameters with finer granularity. These two benchmarks were chosen because they differ considerably in the parameters they are sensitive to. These studies use the full runs to rule out any error introduced by SimPoint projection. From Figure 9, we see that BBench is most sensitive to L2 size and to memory latency while Caffeinemark is primarily sensitive to instruction queue (IQ) size. Therefore, for our sensitivity analysis, we choose to sweep these three parameters. We also sweep the branch predictor size parameter, which fractional factorial experiments determined was not significant for either of the benchmarks, to demonstrate that the parameter is indeed irrelevant.

Figure 10 plots the relative performance improvement of BBench and Caffeinemark when the L2 cache size changes

[2]The full runs missing from the chart did not finish execution for all configurations. The difficulty in managing long jobs and ensuring they run to completion without failure is one of the primary problems we are trying to solve with the run time and experiment reduction methodology proposed in this paper.

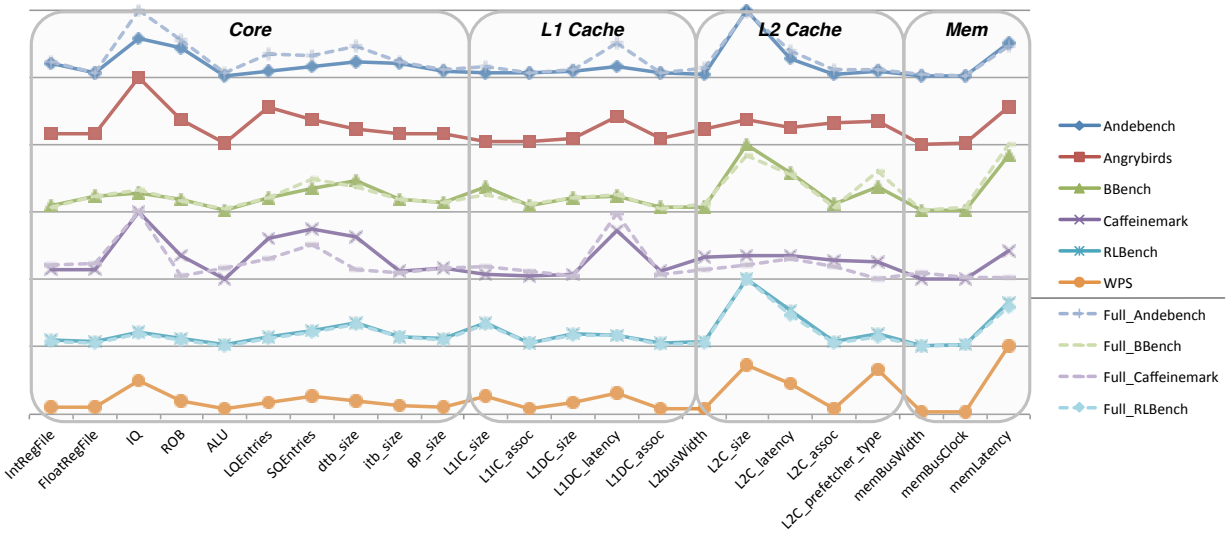

Fig. 9: Sensitivity of system parameters generated from fractional factorial designs with SimPoints. Dashed lines indicate the same studies based on available full runs to demonstrate the effectiveness of SimPoints on fractional factorial designs.

from 64kB to 8MB. The IPC improvement is plotted relative to its corresponding value for the lowest parameter value (64 KB). As predicted by the fractional factorial experiments, BBench performance is indeed very sensitive to L2 size. The difference in performance between the smallest and largest cache is greater than 22%. Caffeinemark performance, in contrast, only fluctuates about 1% over the large range of cache sizes.

Figure 11 reports the relative performance improvement of BBench and Caffeinemark when the memory latency changes from 108ns to 20ns. The results again demonstrate the effectiveness of our fractional factorial experiments. While BBench experiences a performance change of 30% between the fastest and slowest memories, Caffeinemark is found to be insensitive to memory latency as predicted in Figure 9.

Figure 12 lists the relative improvement of the two benchmarks under study when the IQ size is varied from 8 to 256. As noted by the fractional factorial experiments, the IQ size is the most sensitive parameter for Caffeinemark whose performance is affected by about 55%. However, BBench is found to be insensitive to this parameter. It should be noted from the figure that the results of fractional factorial experiments are heavily dependent on the ranges of values chosen for the parameters. In this example, even though the performance difference is huge for Caffeinemark, it can be seen that the performance saturates after the IQ size exceeds 32. Therefore, for the fractional factorial experiments, if the lowest value for IQ size were chosen to be 32 instead of 8, IQ size would not have emerged as an important parameter for Caffeinemark.

To conclusively demonstrate the effectiveness of the fractional factorial experiments, we choose to plot the sensitivity of a parameter that has been reported to be insensitive for both the benchmarks. The results of performance sensitivity of the benchmarks to the size of the branch predictor is shown in Figure 13. As expected, both benchmarks are relatively insensitive to branch predictor size with a maximum performance delta of 1.5%.

Figures 10—13 conclusively show the effectiveness of the fractional factorial experiments in pruning the design space.

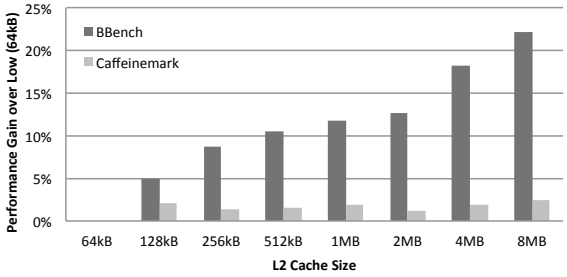

Fig. 10: L2 cache size sensitivity.

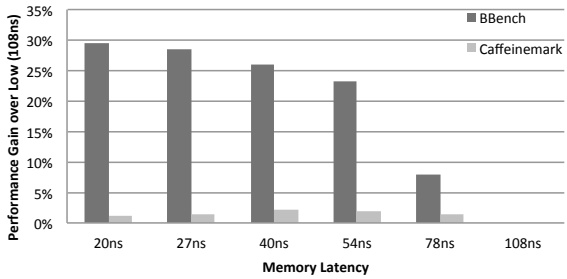

Fig. 11: Memory latency sensitivity.

The results also show that factors identified by the fractional factorial experiments are very dependent on the initial range of the parameters. A different maximum and minimum value of a parameter can completely change the results.

E. Discussion on Simulation Time

In our methodology, we have reduced simulation time in two dimensions. First, SimPoint reduces the simulation time per workload by selecting small, representative samples of each workload. By breaking down the serial workload into multiple pieces, the simulation becomes very easy to parallelize. The results are then stitched together with weights to project full workload behavior. There are a total of 169

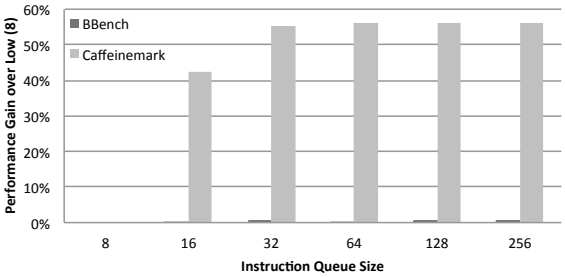

Fig. 12: Instruction Queue size sensitivity.

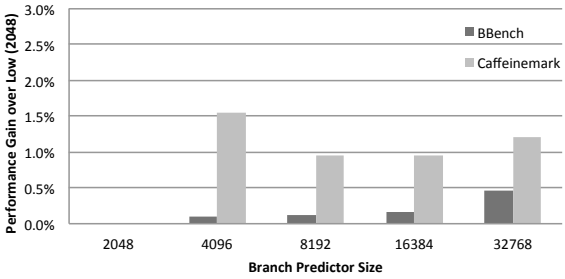

Fig. 13: Branch predictor size sensitivity.

SimPoints generated for the six smartphone workloads, and each takes about 10 minutes to run. Assuming a simulation farm of 100 slots, it would take 20 minutes to run all of the SimPoints. The full runs, however, take 50 hours to run on average and cannot take advantage of a large simulation farm, resulting in 150× overhead compared to the SimPoint approach. We also noticed that SimPoints of our workloads follow the Pareto principle where only 20∼30% of the SimPoints account for more than 80% of the total weights. We discovered that we were able to reach the same conclusions on sensitivity studies by just using those heavily-weighted SimPoints saving an additional 3∼4× of simulation time.

Second, fractional factorial designs significantly reduce the number of experiments to run, thus reducing simulation time. In our experiments in Section IV-D, we ran a sensitivity study on 23 system parameters. A brute-force exhaustive study would have needed 2^{23}, or 8 million, experiments. Fractional factorial reduces the number of experiments to 64. An alternative would be to do a simple one-parameter-at-a-time study (23 experiments) but at a cost of doing a significantly less-rigorous study. We demonstrate that the combination of SimPoint and fractional factorial design effectively reduces orders of magnitude of simulation time while maintaining statistical rigor in the analyses.

F. Full-system Statistics

ARM Streamline is capable of visualizing gem5 statistics and can attribute statistics to certain processes and threads using the Linux kernel scheduling trace. Analyzing the workloads using Streamline reveals many unforeseen aspects that could be useful for further optimizations.

For example, Streamline reveals that BBench has three primary threads: `android.browser`, `WebViewCoreThread`, and `SurfaceFlinger`. Despite having a total of 28 processes and 158 threads active

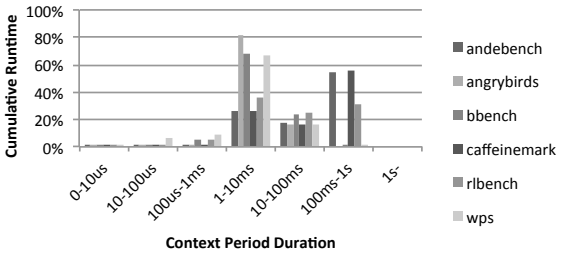

Fig. 14: Cumulative runtime distribution per context period duration lengths.

throughout the benchmark, the total run-time for the three primary threads accounts for more than 96% of the full BBench workload. Detailed per-thread analysis through Streamline shows that the behavior of these threads are very different. We are actively analyzing such thread-specific behavior for optimization opportunities as a part of ongoing heterogeneous systems research.

Also, as the workloads we simulate are full-system, there is a lot of context switching occurring throughout each experiment. Figure 14 shows the cumulative runtime distribution per context period duration. For most of the workloads, the runtimes are biased between the 1ms and 100ms range. Different threads have different average context period durations. These type of analyses are useful for studying Linux scheduler behavior and relevant optimizations.

V. Related Work

Other previous works have studied characteristics of mobile workloads. Gutierrez et al. introduced BBench and characterized a few other Android applications [21]. AM-Bench [26] focused on multimedia benchmarks. MobileBench [27] investigated the user experience while MEVBench [28] primarily examined mobile vision benchmarks. All of these works were based on real hardware platforms and did not have a systematic mechanism of handling deterministic replay of GUI inputs. The use of real hardware also prohibits thorough sensitivity studies and gathering intrinsic statistics.

PCA has been used in a series of works to contrast workload suites. The analysis focused primarily on SPEC benchmark suites [11], [12], [13]. We take this type of analysis a step further by applying it to contemporary mobile workloads and by contrasting them with traditional benchmarks. We also apply PCA on phases within workloads to analyze the phase behavior.

Statistically rigorous techniques to evaluate design spaces have been appearing more frequently in the systems research community. Researchers are realizing that systems are becoming increasingly complicated. Measurement noise from virtual machines, spurious interrupts, OS scheduling decisions and other sources make simple analysis techniques inadequate and instead require more statistical rigor to explain the effects on performance seen from experiments. Recent papers by De Oliveria et al. [29] and Curtsinger et al. [30] show that analyzing software and systems can no longer be adequately done by running a few benchmarks and taking a simple average of the results due to uncontrolled variables in a system. Yi et al. [31] used Plackett-Burman designs [32] to do similar design space explorations as this work but did not present

a comprehensive approach to reduce simulation time as this work has done. Another related statistical work is regression modeling done by Lee et al. [33]. The main goal of Lee et al. was to predict the system performance using regression-fitted models formed from sample points in the design space using a statistically sound method to produce those sample points. These regression models were then intended to characterize the full design space. One of the disadvantages of this work is that Lee et al. needed domain specific knowledge of their benchmark applications and micro-architecture to properly form their regression models. In contrast, the goal of our work is to characterize benchmarks and micro-architecture; predicting performance of different design points is a secondary goal satisfied by performing additional experiments.

VI. Conclusions

In this paper, we described a structured approach to analyzing real applications and exploring micro-architectural designs in a full-system simulation environment in a tractable amount of time. We described the new AutoGUI tool that allows us to run interactive workloads by replaying user inputs on a detailed simulator. We empirically showed that the SimPoint methodology, typically used as a tool to project run-times of bare-metal benchmarks, also worked well for projecting the performance and overall sensitivities of real, multi-threaded applications we tested running on Android. Using PCA, we demonstrated that analyzing these real applications over bare-metal benchmarks is important because they stress the underlying system differently. By combining the use of SimPoint with fractional factorial experimental design principles to discover the experimental variables of interest, we are able to effectively explore an otherwise intractable design space with a detailed simulator. While our proposed methodology is quite general, we demonstrated the flow on less-characterized smartphone applications and benchmarks.

We plan to apply this flow for further detailed analysis of real workloads. A current barrier for the analysis of many real smartphone applications is the lack of peripherals required to run them. We plan to build out our system model with components like a camera, network interface controller, and a GPU model to enable more meaningful simulation of applications like social networking, augmented reality, and gaming.

References

[1] J. C. Mogul, A. Baumann, T. Roscoe, and L. Soares, "Mind the gap: Reconnecting architecture and os research," in *Proceedings of the 13th USENIX Conference on Hot Topics in Operating Systems (HotOS)*, 2011, pp. 1–1.

[2] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti *et al.*, "The gem5 Simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, 2011.

[3] D. Genbrugge, S. Eyerman, and L. Eeckhout, "Interval simulation: raising the level of abstraction in architectural simulation," in *International Symposium on High-Performance Computer Architecture Proceedings*, 2010, pp. 307–318.

[4] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically characterizing large scale program behavior," in *ASPLOS-X*, 2002, pp. 45–57.

[5] "GNU Xnee webpage," http://www.gnu.org/software/xnee.

[6] "Robotium webpage," http://code.google.com/p/robotium.

[7] "MonkeyRunner webpage," http://developer.android.com/tools/help/MonkeyRunner.html.

[8] F. Bellard, "QEMU, a Fast and Portable Dynamic Translator," in *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, 2005, pp. 41–41.

[9] G. Hamerly, E. Perelman, J. Lau, and B. Calder, "SimPoint 3.0: Faster and More Flexible Program Phase Analysis," *Journal of Instruction Level Parallelism*, vol. 7, no. 4, pp. 1–28, 2005.

[10] G. H. Dunteman, *Principal Components Analysis*. Sage Publications, 1989.

[11] A. Joshi, A. Phansalkar, L. Eeckhout, and L. K. John, "Measuring benchmark similarity using inherent program characteristics," *Computers, IEEE Transactions on*, vol. 55, no. 6, pp. 769–782, 2006.

[12] K. Hoste, A. Phansalkar, L. Eeckhout, A. Georges, L. K. John, and K. De Bosschere, "Performance prediction based on inherent program similarity," in *Proceedings of the 15th International Conference on Parallel Architectures and Compilation Techniques*, 2006, pp. 114–122.

[13] A. Phansalkar, A. Joshi, and L. K. John, "Analysis of redundancy and application balance in the SPEC CPU2006 benchmark suite," in *Proceedings of the 34th Annual International Symposium on Computer Architecture (ISCA)*, 2007, pp. 412–423.

[14] R. Fisher, *The Design of Experiments*. Hafner, 1951.

[15] "NIST/SEMATECH e-Handbook of Statistical Methods," http://www.itl.nist.gov/div898/handbook/, 2012.

[16] J. K. Telford, "A brief introduction to design of experiments," *Johns Hopkins APL Technical Digest*, vol. 27, no. 3, p. 224, 2007.

[17] "AndEBench," https://play.google.com/store/apps/.

[18] "The Embedded Microprocessor Benchmark Consortium," http://www.eembc.org/.

[19] "The Embedded CaffeineMark," https://play.google.com/store/apps/.

[20] "RL Benchmark: SQLite," https://play.google.com/store/apps/.

[21] A. Gutierrez, R. G. Dreslinski, T. F. Wenisch, T. Mudge, A. Saidi, C. Emmons, and N. Paver, "Full-system analysis and characterization of interactive smartphone applications," in *Proceedings of the 2011 IEEE International Symposium on Workload Characterization (IISWC)*, 2011, pp. 81–90.

[22] "Angry Birds," https://play.google.com/store/apps/.

[23] "Kingsoft Office 5.3.4 (free)," https://play.google.com/store/apps/.

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

[25] D. Sunwoo, "Visualizing gem5 via ARM DS-5 Streamline," *The First Annual gem5 User Workshop in conjunction with MICRO-45*, 2012.

[26] C. Kim, L. Euna, and K. Hyesoon, "The AM-Bench: An Android Multimedia Benchmark Suite," Georgia Institue of Technology, Tech. Rep. GIT-CERCS-12-04, 2012.

[27] C. Kim, J.-H. Jung, T.-K. Ko, S. W. Lim, S. Kim, K. Lee, and W. Lee, "MobileBench: A Thorough Performance Evaluation Framework for Mobile Systems," *The First International Workshop on Parallelism in Mobile Platforms (PRISM-1), in conjunction with HPCA-19*, 2013.

[28] J. Clemons, H. Zhu, S. Savarese, and T. Austin, "MEVBench: A Mobile Computer Vision Benchmarking Suite," in *IEEE International Symposium on Workload Characterization (IISWC)*, 2011, pp. 91–102.

[29] A. B. de Oliveira, S. Fischmeister, A. Diwan, M. Hauswirth, and P. F. Sweeney, "Why you should care about quantile regression," in *Proceedings of the 18th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2013, pp. 207–218.

[30] C. Curtsinger and E. D. Berger, "Stabilizer: statistically sound performance evaluation," in *Proceedings of the 18th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2013, pp. 219–228.

[31] J. J. Yi, D. J. Lilja, and D. M. Hawkins, "A statistically rigorous approach for improving simulation methodology," in *Proceedings of the 9th International Symposium on High-Performance Computer Architecture (HPCA-9)*, 2003, pp. 281–291.

[32] R. L. Plackett and J. P. Burman, "The Design Of Optimum Multifactorial Experiments," *Biometrika*, vol. 33, no. 4, pp. 305–325, 1946.

[33] B. C. Lee and D. M. Brooks, "Accurate and efficient regression modeling for microarchitectural performance and power prediction," in *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2006, pp. 185–194.