VarDroid: Online Variability Emulation in Android/Linux Platforms

Pietro Mercati
SEELAB
UC San Diego
pimercat@eng.ucsd.edu

Francesco Paterna
Strategic CAD Labs
Intel Corporation
francesco.paterna@intel.com

Andrea Bartolini
Integrated System Laboratory
ETH Zurich
DEI, University of Bologna
barandre@iis.ee.ethz.ch

Mohsen Imani
SEELAB
UC San Diego
moimani@eng.ucsd.edu

Luca Benini
Integrated System Laboratory
ETH Zurich
DEI, University of Bologna
luca.benini@iis.ee.ethz.ch

Tajana Šimunić Rosing
SEELAB
UC San Diego
tajana@ucsd.edu

ABSTRACT

Variability is the real big challenge for integrated circuits. Today, simulators help to estimate the effect of variability, but fail to capture real workload dynamics and user interactions, which are fundamental to mobile devices. This paper presents VarDroid, a low-overhead tool to emulate power and performance variability on real platforms, running on top of the Android operating system. VarDroid enables analyzing the effect of variability in power and performance while capturing the complex interactions characteristic of mobile workloads, thus relating to user's quality of experience. The paper presents use cases to show the utility of VarDroid to test applications, device and OS robustness under the effects of variability. Our results show that a variability-agnostic OS can incur in a performance penalty of up to 60% and a power penalty of up to 20%.

1. INTRODUCTION

Variability in integrated circuits refers to the deviation of the actual value of a design parameter from its nominal value, such as transistor channel length and threshold voltage. Sources of variation are static and dynamic. Static variations are due to the intrinsic inaccuracy of the fabrication process. Dynamic sources, such as ambient temperature and system workload, change their impact over time.

In multicore processors, transistor-level static variability determines power and performance distributions across cores of the same model. Consequently, computing units have a power consumption and operating voltage/frequency which is different from nominal right after fabrication [7]. As the scaling of CMOS technology progresses and dimensions of transistors shrink, the impact of variations is more dramatic, up to the point in which counteracting with higher design margins is extremely costly [5].

Simulators have been developed to help designers make chips more robust to variations [24, 20]. These tools can be integrated into the design flow to estimate the resulting impact of variability. However, they cannot account for dynamic variations and cannot capture real workloads behavior. Moreover, they require a highly detailed architectural description, which may not be always available.

Given that dynamic variations play an important role, a key idea is to leverage sensors to expose variability to higher levels of the software stack (e.g. the operating system) to manage it at runtime. This strategy is referred to as Dynamic Variability Management (DVM). DVM can leverage common hardware sensors such as performance counters and temperature sensors, or more sophisticated ones, such as degradation sensors and frequency sensors (which are devices capable of monitoring path delays) [14]. The latter, however, are only available on prototypes and research platforms. The community needs a tool to evaluate the impact of variability and to test DVM strategies on real commercial devices.

This is particularly urgent for mobile devices, as they are growing in popularity and work in a variety of environments, from mountains to deserts. Mobiles are characterized by interactive workloads, which are hard to simulate and predict, as they strongly depend on users [8]. Moreover, the goal of mobiles is to provide quality user experience, rather than high performance. Therefore, it is crucial to be able to estimate the impact of variability on user experience, which is not possible with simulators.

Most of mobile devices today adopt the Android operating system. Android builds on top of the Linux kernel, which offers access to hardware resources through drivers. It also offers the possibility to switch operating conditions dynamically to implement DVM policies. Finally, Android implements interfaces between user and kernel space that enable cross-layer synchronization. Flexibility and open-source make Android a perfect candidate for our purpose.

In this work, we develop and implement VarDroid, a low-overhead framework to emulate the effect of performance and power variability on Android devices. The emulation on real devices enables capturing the effects of environmental changes and interaction with users, which is not possible with simulators. Thanks to this, VarDroid can estimate the impact of variability on power and quality of user experience. Also, the Variability-induced Power Breakdown (VIP) allows estimating the ratio of dynamic and leakage power under variability. We present the assumptions of our work, the details of the implementation, and use-cases to demonstrate the feasibility and ease of use of our framework. We also make VarDroid available online for download [3]. Figure 1 better explains the motivation and goal of VarDroid. At

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

GLSVLSI '16, May 18-20, 2016, Boston, MA, USA

© 2016 ACM. ISBN 978-1-4503-4274-2/16/05…$15.00

DOI: http://dx.doi.org/10.1145/2902961.2902971

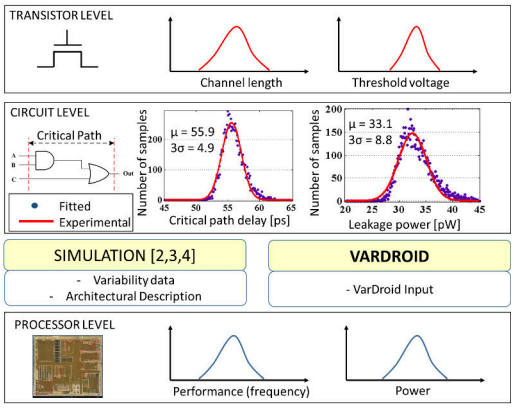

Figure 1: VarDroid context and motivation

the transistor level, variability mainly affects channel length and threshold voltage. At the circuit level, this has an impact on path delay and power consumption. As a preliminary study, we implement a circuit consisting of AND and OR gates using 32-nm technology with HSPICE simulator. For each transistor, we consider a variation in both channel length and threshold voltage which follows a Gaussian distribution with 3σ equal to 10% of the original value, similarly as in [6]. A Monte Carlo simulation with 5000 iteration produces the distributions of path delay (which affects operating frequency) and leakage power. The figure shows that such distributions are also well fitted by Gaussian curves. Finally, at the processor level, previous work [7, 24, 20] (on the left) focuses on simulating the effect of variability on a multiprocessor starting form detailed data on transistor parameters and propagates it to the processor level. In the end, it achieves Gaussian distributions for power and performance (i.e. operating frequency) for a large number of processors. In our work, instead, we emulate variability on a real device, by injecting perturbations in the Operating System (denoted as *VarDroid input*). Thanks to this, we do not require detailed low-level information about transistor variability. Nevertheless, we will show that we still obtain Gaussian distributions for power and performance, thus matching the results of state-of-the-art simulator and low level models. To clarify this, in the paper we describe models for power and frequency as a reference.

In our experiments, we use VarDroid to investigate the robustness of applications with respect to variability, the impact of variability on user experience and the tolerance of current OS migration policy. Our results show that performance of applications may drop by up to 35% in presence of variability and that a variability-agnostic OS can incur in a performance penalty of up to 60% and a power penalty of up to 20%.

The rest of the paper is organized as follows: Section 2 presents the related work, Section 3 describes the variability models, Section 4 describes the framework implementation and Section 5 presents experimental results. Finally, Section 6 concludes this paper.

2. RELATED WORK

In the last decade, variability has been the subject of numerous publications, from device-level simulation up to approximate computing for applications.

Architectural-level variability simulators have been developed, such as VARIUS [24], VAM [20], and VARIUS-NTV [11]. These tools can be integrated with simulators such as GEM5, but they require detailed architectural description. Garg et al. [9] propose a framework to analyze the impact of variability at the system level to help designers determining the tradeoff between performance and clock domain granularity. These techniques extract random values for architectural parameters and inject variability in the configuration of the simulator. The main drawback of such an approach is that simulations are very slow, as a few seconds of clock time for a real device could take hours.

An alternative approach in represented by variability emulation, which consists in injecting variations on a real device. In ERSA [15], errors are injected in architecturally visible registers. Similarly, in [23] a large fault injection campaign is conducted in the openSPARC T1 core logic. In [16], a combined hardware and software solution aims at identifying anomalous software behavior in presence of hardware faults. However, solutions based on hardware prototypes and FPGAs present low flexibility and portability. Work in reference [13] proposes VESPA, a variability emulation framework for SoC performance analysis. The proposed solution addresses the challenge of translating component-level characteristics into system-level performance distribution. The approach is evaluated for a 802.11 MAC processor and an MPEG encoder. Moreover, VESPA is used to assess the performance distribution of processors design prior to fabrication, given the design synthesis. Therefore is not effective in capturing real workload and the effects of real environment. Work in [25] proposes VarEmu. Implemented as an extension of the QEMU operating system, this can be used to evaluate variability-aware software techniques. VarEmu extracts timing and cycle count from the emulated code. This information is fed into a variability model with configurable parameters that determines energy consumption and fault variations in the virtual machine. VarEmu relies on models for power estimation, which have to be adapted to different platforms and may present accuracy problems. None of the previous work on variability emulation addresses mobiles and the crucial need of capturing interactive workload on variability-affected devices. Our approach is different in that we inject faults directly in the native OS of real mobiles in the form of a VarDroid input.

Recent publications on Dynamic Variability Management (DVM) propose scheduling and DVFS algorithms for embedded platforms [21] and for Android-based devices [19]. However, these solutions do not account for real workload and real user interaction.

Emulation in Android has been addressed before for reliability [18, 17]. Work in [18] implements a kernel module to virtualize the presence of degradation sensors to emulate the effect of degradation mechanisms. Our work is orthogonal to [18], in that we focus on performance and power variability, while the technique in [18] aims at emulating degradation. At this time, no publication presents power and performance variability emulation capable of running on real mobiles.

In this work, we develop and implement a framework for emulating the effects of performance and power variability on real Android devices, while capturing real mobile workloads. Finally, we show how to use it to test system robustness against variability.

3. VARIABILITY MODELS

Variability in integrated circuits manufacturing has two components: D2D (die-to-die) and WID (within-die) [24]. WID variations can be further separated into systematic and random variations. At the microarchitectural level, the elements of key importance for variability are the transistor threshold voltage V_{th} and the effective channel length L_{eff} of transistors. More recently, C2C (core-to-core) variations have been considered [7]. Following this model, low-level variations in V_{th} and L_{eff} result in operating frequencies and power consumption which are distributed among cores. Since V_{th} and L_{eff} can be considered normally distributed with good approximation, power consumption and critical path delay (thus, operating frequency) can be considered

normally distributed among cores as well [24]. Power is the sum of dynamic and leakage contributions. Dynamic power has a well-known dependency on frequency and voltage [12]. Leakage power, instead, depends on leakage current, which can be modeled as in Equation 1.

$$I_{leak} = b\left(\frac{kT}{q}\right)^2 \exp\left(\frac{q(V_{off} - V_{th})}{\eta kT}\right) \quad (1)$$

Where q is the electron charge and k is the Boltzmann constant. The terms b, η and V_{off} are empirically determined parameters. As shown in [24], the knowledge of V_{th} variance can be exploited to assess the impact on chipÂĄ leakage power. As a result, the value of the ratio between perturbed and nominal leakage can be expressed as in Equation 2.

$$\frac{P_{leak}}{P_{leak_0}} = \frac{V_{leak}}{V_{leak_0}} = \exp\left(\frac{1}{2}\left(\frac{q\sigma}{\eta kT}\right)^2\right) \quad (2)$$

Where P_{leak_0} and I_{leak_0} are the nominal values of leakage power and current. This indicates that the variation in leakage depends on the standard deviation σ of V_{th}. As for critical path delay, this is related to the switching delay of an inverter gate, which is expressed by Equation 3.

$$T_g = a\frac{L_{eff}V}{\mu^*(V - V_{th})^\alpha} \quad (3)$$

Where a is an empirically determined parameter, α is typically 1.3 and μ^* is the mobility of carriers. Given the critical path delay T_g, which depends on gate switching delays, the maximum frequency at which a processor can be clocked is given by its inverse $f_{MAX} = 1/T_g$.

In this work, we refer to Equations 1, 2 and 3 to characterize VarDroid configurations of the target platform. This is done assuming a diverse impact of variability in L_{eff} and V_{th}, expressed as the ratio μ/σ of the underlying Gaussian distribution. Finally, at the processor level, variability results in Gaussian distributions of maximum frequency (e.g. performance) and power. VarDroid naturally captures dynamic variations, such as workload and environmental conditions (such as SoC and ambient temperature), given that it is implemented on real devices.

4. VARDROID ARCHITECTURE

Figure 2 shows the block diagram of the proposed implementation. It has two main components: the **VarDroid Engine** and the **VarDroid Monitor**. The **VarDroid**

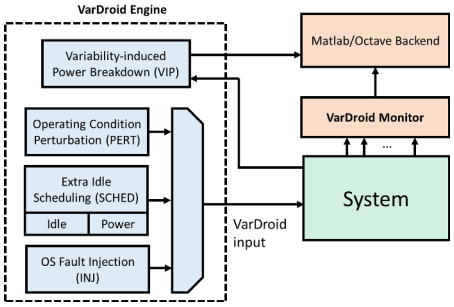

Figure 2: VarDroid block diagram

Engine is the software element which perturbs the system to emulate the effect of variability and has four components: the input configurations PERT, SCHED and INJ, and the Variability-Induced Power breakdown emulator (VIP). The components are described into details in the following.

Operating conditions perturbation (PERT): The operating frequencies of cores are limited by PERT to mimic

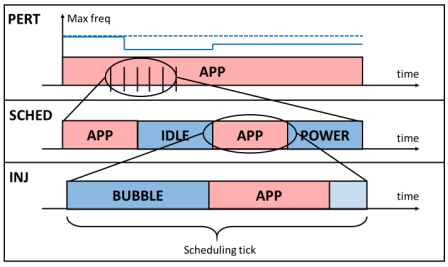

Figure 3: VarDroid input configuration

the effect of performance variability. We generate a Gaussian distribution for f_{MAX}, extract and assign randomly one value to each core. In this way, operating conditions are limited as if the multicore was affected by variations on performance. This emulates a scenario in which L_{eff} and V_{th} of transistors have variability, which reflects in f_{MAX} variability (i.e. performance), as described by Equation 3. The operating condition perturbation is implemented at the userspace level, and exploits the *sysfs* fields exposed by the *cpufreq* driver that allows to set an upper bound on operating frequency. The selected f_{MAX} represents the input of PERT to the system. Figure 3 shows that PERT perturbs the behavior at a high level (i.e. system level), by selecting an upper bound for operating conditions.

Extra scheduling (SCHED): SCHED forces the scheduling of extra periods to interleave the normal execution of applications. It is implemented as a program that alternates periods of execution and sleep (implemented with *nanosleep*). The effect of SCHED is varied by tuning the *nanosleep* duration, which affects the SCHED duty cycle. SCHED can be configured with an *Idle* application for performance variation (a program executing a long sequence of NOPs) and a *Power* application for power consumption variation (a sequence of cpu-intensive operations). In the first case, the emulated scenario is similar to the one achieved with PERT, in which L_{eff} and V_th variations result in frequency variations. In the second case, the scenario includes power variability, both dynamic and leakage. Figure 3 shows that SCHED occupies some scheduling intervals with spurious applications. Both PERT and SCHED are implemented as C programs and cross-compiled with Android NDK toolchain to run on the target platform.

OS fault injection (INJ): INJ is implemented by inserting in the OS kernel source code execution *bubbles* that perturb performance. The *bubble* is a fixed sequence of instruction which is inlined in the kernel source code. A single bubble executes a sequence of 20 NOPs (for the current design of VarDroid) at the beginning of each scheduling tick. The impact of variability can be tuned by choosing the number of bubbles (referred to as *bubble length*). INJ also emulates the scenario in which variability in L_{eff} and V_{th} results in performance variability. The fault injector is implemented as an inlined function which is invoked in the *scheduling_tick()* function in the scheduler source code (*core.c*). Figure 3 shows that INJ occupies part of the scheduling interval itself with spurious instructions.

We denote the VarDroid input ν in the three configurations respectively as the percentage of frequency degradation over f_{MAX} for PERT, the nanosleep duration for SCHED and the bubble length for INJ.

The three input configurations have complementary benefits. PERT and INJ can emulate performance variations, but are not effective in emulating power variations. INJ emulates performance variations at a finer granularity compared to PERT, but requires recompiling the Linux kernel. PERT and SCHED, instead, can be more easily ported, as they are userspace programs. Note that the three input config-

urations inject variability at different layers of the software stack, and consequently at different time granularity, as Figure 3 shows.

Variability-induced Power Breakdown (VIP): Beside the three input configurations, VarDroid features VIP, a framework to estimate the proportion of dynamic to leakage power depending on input variation on performance. The main motivation for the design of VIP is that on a real platform, we can only measure the total power consumption, but we cannot distinguish the dynamic and leakage contributions. This is important, because the variation on parameters L_{eff} and V_th determines a variation in the ratio between dynamic and leakage power. It is well known that dynamic power P_{dyn} is directly proportional to frequency, and VarDroid establishes a relation between the input variability ν and f_{MAX}. Since f_{MAX} depends on V_{th}, this can be derived from ν as well. The leakage power P_{leak} depends exponentially on V_{th}. Overall, this allows deriving the ratio P_{dyn}/P_{leak} as ν varies, as expressed by Equation 4.

$$\frac{P_{dyn}}{P_{leak}} \approx \sigma \frac{f(\nu)}{\exp(g^{-1}(\beta f(\nu)))} \quad (4)$$

Where σ, β are fitting parameters. The function g can be derived from Equation 3, and it is so that $g(V_{th}) = f(\nu)$, while $f(\nu)$ can be fitted experimentally. As ν increases, the ratio should also increase, as this reflects a scenario in which V_{th} is higher than nominal. VIP computes a ratio, thus it does not depend on which power is actually consumed and measured. We will better characterize VIP in the results section.

The **VarDroid Monitor** infrastructure has been implemented by modifying the one described in [18]. It has a kernel sampling function, a monitor driver to transfer data to the user space and a monitor daemon to regulate the data transfer. The monitor infrastructure is able to get one sample per each scheduling tick (10ms in our setup) and transfers data to the userspace with low overhead. Finally, the Matlab backend allows parsing and plotting results.

5. EXPERIMENTAL EVALUATION

We implemented VarDroid on the Odroid XU3 board. This board has a Samsung Exynos 5422 Octa core based on ARM big.LITTLE architecture, with a Cortex^{TM}-A15 2.0Ghz quad core cluster and Cortex^{TM}-A7 quad core cluster [1]. The platform has a 2MB L2 cache and a 2GB RAM. The OS is Android 4.4.4 with Linux kernel 3.10.9.

5.1 VarDroid Validation

The key idea for validating Android is to show that when varying the VarDroid input (for each one of the three configuration) we obtain a Gaussian distribution of frequency and power consumption, similar as in [7]. This proves that a variation of the VarDroid input reflects a scenario in which transistor parameters L_{eff} and V_{th} are subject to variability.

For validating VarDroid, we conduct a series of experiments using single-threaded microbenchmarks. The *cpu-bound* executes a series of ALU operations without accessing memory, the *L2-bound* executes a series of loads from the L2 memory and the *mem-bound* similarly executes a series of loads from the main memory. To avoid the effect of migrations, microbenchmarks exploit the *set affinity* mechanism to run on a specific core.

Performance Variability: In the following experiments, we keep only one LITTLE core active, running at fixed maximum frequency. Analogous results can be obtained for big cores. Figure 4 shows the normalized execution time of the three microbenchmarks for the three VarDroid configurations. The execution time of the benchmarks increases as the VarDroid input increases. This means that VarDroid effectively makes the processor behave as if it had a maximum frequency lower than the nominal. To further validate

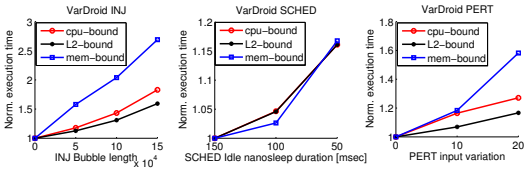

Figure 4: Microbenchmarks execution times

this behavior, we run the *cpu-bound* benchmark with random VarDroid input extracted from a normal distribution, in the three configurations. In Figure 5, the plots on the left show the histogram of VarDroid input. Then we collect the execution times and build a histogram, shown in the plots on the right. In the case of SCHED, we employed a smaller number of samples and a version of the *cpu-bound* benchmark of longer duration. The resulting shape match with the low-level models discussed from Section 3 and the example of Figure 1. In this way VarDroid emulation shows results as if the benchmarks were executed by different instances of the same processor, affected by performance variability. No variation in execution time is present when VarDroid is disabled. **Power Variability**: Next, we verified the ef-

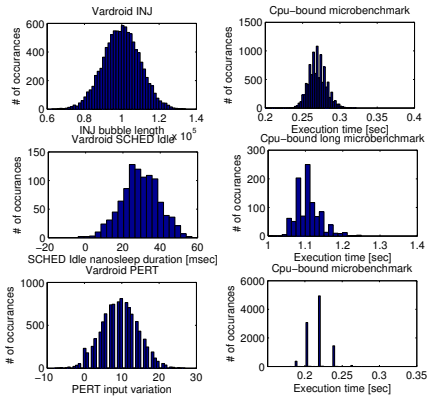

Figure 5: VarDroid performance distribution

fect of VarDroid on power variability. In Figure 6 we show the histogram of random input for the SCHED Power configuration (e.g. *nanosleep* duration) and the corresponding histogram of average power consumption. In this experiment, we are measuring only the power consumption of the LITTLE cluster, but a similar result can be obtained for the big cluster. One sample of power consumption is computed by averaging the power over a period of 120 seconds, with a given SCHED power input. Also in this case, the histograms match the behavior expected from low-level models of Section 3. In Figure 7, we perform the same measurement while executing a sequence of *cpu-bound* benchmarks. The second plot shows the power variability, while the third plot shows the execution time variability. The variation in the execution times is very low (in the 2% of the mean value). This means that the SCHED power setting can be effectively used to decouple power and performance variability effects. Finally, Figure 8 shows the qualitative behavior of the VIP ratio. For simplicity, the relations involved in Equation 4 are assumed linear. Dynamic and leakage power are scaled in the [0,1] interval and the final ratio is normalized with respect to the minimum. The final behavior is that of a log-normal distribution. As reported in [12] the log-normal shape is generated by the exponential increase in leakage current, thus matching with the model described by Equations 2 and 3. Note that the distribution of the VarDroid

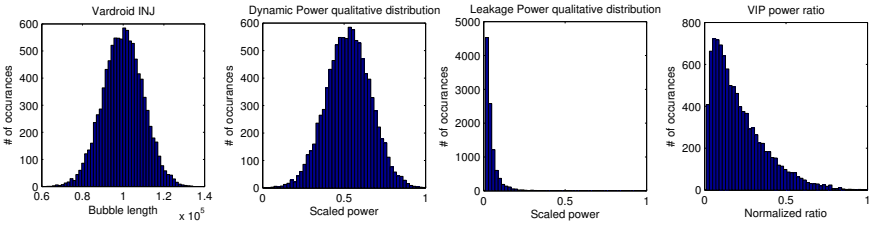

Figure 8: Variability-induced Power breakdown distribution

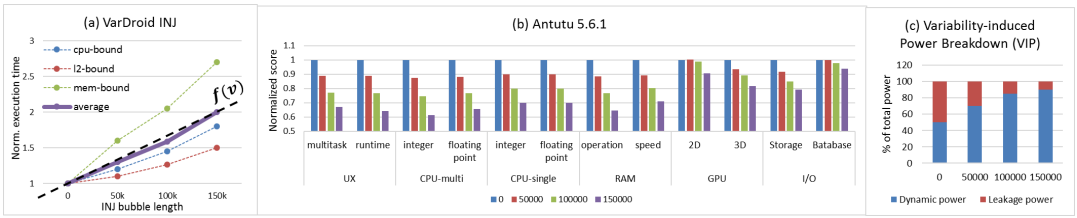

Figure 9: Performance degradation and VIP in Antutu benchmark

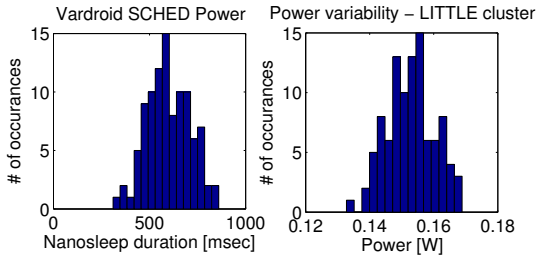

Figure 6: SCHED power variability of little cluster

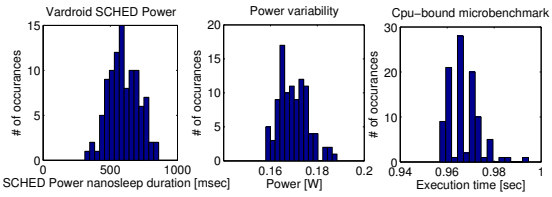

Figure 7: Power variability with cpu-bound benchmark

input can be tuned to achieve a desired power and performance variation.

5.2 Use Cases

In the following experiments, we present use cases in which we can exploit VarDroid.

Application Robustness: VarDroid can be used to investigate the impact of variability on real application performance and power consumption. In Figure 9b, we present the scores obtained with the mobile benchmark Antutu v5.6.1 [4] with increasing impact of INJ. Scores of mobile benchmarks are a common metric to compare the quality of execution of different devices. As expected, all the scores decrease as the INJ input increases, by up to 35%, also in accordance with the results obtained on microbenchmarks. Therefore, VarDroid can be used by application developers to test the performance of their apps in a real device affected by variability. Figure 9c shows the ratio between dynamic and leakage power obtained with VIP. The base case is assumed to have 50% of dynamic and 50% of leakage power, in accordance with ITRS [5]. In this case, the function f is derived from the results in Figure 3 from INJ microbenchmarking, also shown in Figure 9a. As expected, the ratio P_{dyn}/P_{leak} increases with the VarDroid input.

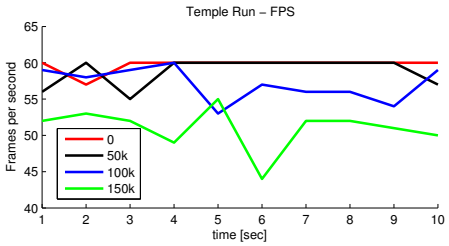

Figure 10: VarDroid impact on FPS

User Experience: Since VarDroid is implemented on a real device, it can capture real workload dynamics and user interaction, therefore it can be used to assess the impact of variability on user experience. For this experiment, we record and replay a 10 seconds-long touch event trace of a popular Android 3D game (Temple Run [2]), in order to reproduce the same workload [10]. Then, we execute the trace with different INJ configurations, and measure Frame per Seconds (FPS) by monitoring the SurfaceLinger service. Indeed, it is well recognized that FPS in 3D gaming is a good metric for quality of experience [22]. In particular, the closer to 60fps (which is the maximum allowed by Android), the better. The results in Figure 10 show the different FPS traces. As expected, the FPS lowers as the INJ input increases. In particular, in a scenario of severe variability impact (bubble length of 150k), user experience is heavily affected, as FPS drops below 50.

Variability Tolerance of the OS: VarDroid can be used to detect limitations and bugs of existing OS design. In this experiment, we focus on the workload migration policy employed in ARM big.LITTLE architectures [1]. In such architecture, a hysteresis mechanism regulates the task migration between LITTLE and big cores, with a high a and a low utilization threshold. When the high threshold is exceeded, the task is migrated to the big core, and is returned to the

LITTLE core if utilization decreases below the low threshold. The limitation of such mechanisms is that migration thresholds are fixed. Because of this, the execution can incur significant performance penalties. To show this, we im-

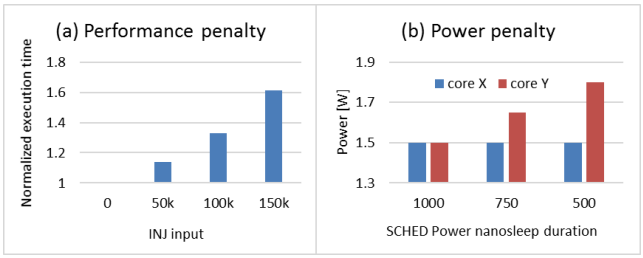

Figure 11: Migration (a) performance and (b) power penalty

plemented a single threaded cpu-bound synthetic workload that periodically alternates between three execution phases, respectively with low, medium and high utilization. Then, we execute it with only one LITTLE core and one big core active. The workload is initially allocated on the LITTLE core and execute low and medium phases. The high phase instead triggers the migration to the big core. We execute the program with increasing INJ input, and measure execution times. Figure 11a shows the normalized execution times of the medium phase. Results show a penalty of up to 60%. A lower migration threshold in this case would have avoided such penalty. The variability tolerance of the migration policy in this case could be improved by changing the migration threshold dynamically. In a scenario in which platforms can actually detect the variability at runtime thanks to dedicated sensors [14], thresholds should be adapted dynamically.

As a final example, we make the case in which the LITTLE core can choose to migrate the workload between two big cores, a core X of the big cluster with nominal power consumption (which is 1.5W from our measurements) and a core Y with higher power consumption due to variability. Results in Figure 11b show that a variation-agnostic migration policy can incur in a power penalty of up to 20%.

6. CONCLUSION

This paper presents VarDroid, the first tool for power and performance emulation for mobiles, implemented on top of the Android operating system. VarDroid is validated with microbenchmarks for performance and power variability. We presented use cases to show how VarDroid can be used to analyze the robustness of existing applications against variability, evaluate the impact of variability on user experience and highlight the limitations of current OS design on heterogeneous architectures.

7. ACKNOWLEDGMENTS

This work was supported in part by NSF grant 1527034.

8. REFERENCES

[1] Arm white paper. 2014. big.little technology: The future of mobile.
[2] https://en.wikipedia.org/wiki/temple_run.
[3] https://github.com/pietromercati/vardroid.
[4] http://www.antutu.com/en/ranking.shtml.
[5] Itrs reports, december 2014 [online] http://www.itrs.net. 2014.
[6] A. Agarwal, D. Blaauw, and V. Zolotov. Statistical timing analysis for intra-die process variations with spatial correlations. In *Computer Aided Design, 2003. ICCAD-2003. International Conference on*, pages 900–907, Nov 2003.
[7] K. A. Bowman, A. R. Alameldeen, S. T. Srinivasan, and C. B. Wilkerson. Impact of die-to-die and within-die parameter variations on the clock frequency and throughput of multi-core processors. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 17(12):1679–1690, Dec 2009.
[8] H. Falaki, R. Mahajan, S. Kandula, D. Lymberopoulos, R. Govindan, and D. Estrin. Diversity in smartphone usage. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, MobiSys '10, pages 179–194, New York, NY, USA, 2010. ACM.
[9] S. Garg and D. Marculescu. System-level throughput analysis for process variation aware multiple voltage-frequency island designs. *ACM Trans. Des. Autom. Electron. Syst.*, 13(4):59:1–59:25, Oct. 2008.
[10] L. Gomez, I. Neamtiu, T. Azim, and T. Millstein. Reran: Timing- and touch-sensitive record and replay for android. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 72–81, Piscataway, NJ, USA, 2013. IEEE Press.
[11] U. R. Karpuzcu, K. B. Kolluru, N. S. Kim, and J. Torrellas. Varius-ntv: A microarchitectural model to capture the increased sensitivity of manycores to process variations at near-threshold voltages. In *Dependable Systems and Networks (DSN), 2012 42nd Annual IEEE/IFIP International Conference on*, pages 1–11, June 2012.
[12] N. S. Kim, T. Austin, D. Baauw, T. Mudge, K. Flautner, J. S. Hu, M. J. Irwin, M. Kandemir, and V. Narayanan. Leakage current: Moore's law meets static power. *Computer*, 36(12):68–75, Dec 2003.
[13] V. J. Kozhikkottu, R. Venkatesan, A. Raghunathan, and S. Dey. Vespa: Variability emulation for system-on-chip performance analysis. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, pages 1–6, March 2011.
[14] L. Lai, V. Chandra, R. C. Aitken, and P. Gupta. Slackprobe: A flexible and efficient in situ timing slack monitoring methodology. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 33(8):1168–1179, Aug 2014.
[15] L. Leem, H. Cho, J. Bau, Q. A. Jacobson, and S. Mitra. Ersa: Error resilient system architecture for probabilistic applications. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, pages 1560–1565, March 2010.
[16] M.-L. Li, P. Ramachandran, S. K. Sahoo, S. V. Adve, V. S. Adve, and Y. Zhou. Understanding the propagation of hard errors to software and implications for resilient system design. In *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XIII, pages 265–276, New York, NY, USA, 2008. ACM.
[17] P. Mercati, A. Bartolini, F. Paterna, L. Benini, and T. S. Rosing. An on-line reliability emulation framework. In *Proceedings of the 2014 12th IEEE International Conference on Embedded and Ubiquitous Computing*, EUC '14, pages 334–339, Washington, DC, USA, 2014. IEEE Computer Society.
[18] P. Mercati, A. Bartolini, F. Paterna, T. S. Rosing, and L. Benini. A linux-governor based dynamic reliability manager for android mobile devices. In *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*, pages 1–4, March 2014.
[19] P. Mercati, F. Paterna, A. Bartolini, L. Benini, and T. S. Rosing. Dynamic variability management in mobile multicore processors under lifetime constraints. In *Computer Design (ICCD), 2014 32nd IEEE International Conference on*, pages 448–455, Oct 2014.
[20] M. Miranda, B. Dierickx, P. Zuber, P. Dobrovoln, F. Kutscherauer, P. Roussel, and P. Poliakov. Variability aware modeling of socs: From device variations to manufactured system yield. In *Quality of Electronic Design, 2009. ISQED 2009. Quality Electronic Design*, pages 547–553, March 2009.
[21] F. Paterna, A. Acquaviva, and L. Benini. Aging-aware energy-efficient workload allocation for mobile multimedia platforms. *IEEE Transactions on Parallel and Distributed Systems*, 24(8):1489–1499, Aug 2013.
[22] A. Pathania, Q. Jiao, A. Prakash, and T. Mitra. Integrated cpu-gpu power management for 3d mobile games. In *Proceedings of the 51st Annual Design Automation Conference*, DAC '14, pages 40:1–40:6, New York, NY, USA, 2014. ACM.
[23] A. Pellegrini, R. Smolinski, L. Chen, X. Fu, S. K. S. Hari, J. Jiang, S. V. Adve, T. Austin, and V. Bertacco. Crashtest'ing swat: Accurate, gate-level evaluation of symptom-based resiliency solutions. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, pages 1106–1109, March 2012.
[24] S. R. Sarangi, B. Greskamp, R. Teodorescu, J. Nakano, A. Tiwari, and J. Torrellas. Varius: A model of process variation and resulting timing errors for microarchitects. *IEEE Transactions on Semiconductor Manufacturing*, 21(1):3–13, Feb 2008.
[25] L. Wanner, S. Elmalaki, L. Lai, P. Gupta, and M. Srivastava. Varemu: An emulation testbed for variability-aware software. In *Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2013 International Conference on*, pages 1–10, Sept 2013.