Special Issue

Lucas Wanner*, Liangzhen Lai, Abbas Rahimi, Mark Gottscho, Pietro Mercati, Chu-Hsiang Huang, Frederic Sala, Yuvraj Agarwal, Lara Dolecek, Nikil Dutt, Puneet Gupta, Rajesh Gupta, Ranjit Jhala, Rakesh Kumar, Sorin Lerner, Subhasish Mitra, Alexandru Nicolau, Tajana Simunic Rosing, Mani B. Srivastava, Steve Swanson, Dennis Sylvester, and Yuanyuan Zhou

NSF expedition on variability-aware software: Recent results and contributions

Abstract: In this paper we summarize recent results and contributions from the NSF Expedition on Variability-Aware Software, a five year, multi-university effort to tackle the problem of hardware variations and its implications and opportunities in software. The Expedition has made contributions in characterization and online monitoring of variations (particularly in microprocessors and flash memories), proposed new coding techniques for variability-tolerant storage, provided tools and platforms for the development of variability-aware software, and created new runtime support systems for variability-aware task-scheduling and execution.

Keywords: Hardware variations, variability-aware software, survey.

***Corresponding author: Lucas Wanner,** Electrical Engineering Department, University of California, Los Angeles, e-mail: wanner@ucla.edu
Liangzhen Lai, Mark Gottscho, Chu-Hsiang Huang, Frederic Sala, Lara Dolecek, Puneet Gupta, Mani B. Srivastava: Electrical Engineering Department, University of California, Los Angeles
Abbas Rahimi, Pietro Mercati, Rajesh Gupta, Ranjit Jhala, Sorin Lerner, Tajana Simunic Rosing, Steve Swanson, Yuanyuan Zhou: Computer Science and Engineering Department, University of California, San Diego
Yuvraj Agarwal: School of Computer Science, Carnegie Mellon University
Nikil Dutt: Department of Computer Science, University of California, Irvine
Rakesh Kumar: Electrical and Computer Engineering Department, University of Illinois at Urbana Champaign
Subhasish Mitra: Department of Electrical Engineering, Stanford University
Alexandru Nicolau: Center for Embedded Computer Systems, University of California, Irvine
Dennis Sylvester: Department of Electrical Engineering and Computer Science, University of Michigan

ACM CCS: Hardware → Robustness → Hardware reliability → Process, voltage and temperature variations

DOI 10.1515/itit-2014-1085
Received December 19, 2014; revised April 6, 2015; accepted April 8, 2015

1 Introduction

The National Science Foundation (NSF) Expedition on Variability-Aware Software for Efficient Computing with Nanoscale Devices is a five year, multi-university project started in 2010 to tackle a major challenge facing the computing industry: the over-engineering of systems by chip designers, with wide error tolerances and guardbands, in order to accommodate variations in manufactured components. Variation has multiple causes, including: imperfections in manufacturing and materials; burn-in and lifetime degradation; temperature, altitude and other operating environment changes; etc. These sources of variability become more acute as the size of microelectronic components decline, use of silicon becomes increasingly problematic, and therefore new materials are adopted by the semiconductor industry. Given these trends in variability, chip and system designers – if they want to retain the same deterministic interface with software – are forced to engineer systems with orders-of-magnitude greater levels of error tolerance beyond what is ideal. Ultimately this variability poses a critical challenge to future advances in computing machines, by counteracting improvement in component capabilities that the semiconductor and other computing industries have come to expect.

The Variability Expedition envisions a computing world where system components – led by proactive software – routinely monitor, predict and adapt to the variability of manufactured systems. The Variability Expedi-

tion proposes a new class of computing machines that are adaptive but highly energy efficient. They will continue working while using components that vary in performance or grow less reliable over time and across technology generations. A fluid software-hardware interface will mitigate the variability of manufactured systems and make machines robust, reliable and responsive to changing operating conditions – offering the best hope for perpetuating the fundamental gains in computing performance at lower cost of the past 40 years.

The expedition has made contributions across the gamut of the variability problem, from quantifying and monitoring variations, to coding schemes for variability-tolerant systems, to runtime support systems, to tools and testbeds. Research in the expedition in organized into five main thrusts: 1) measurement and modeling, 2) design tools and testing methodologies, 3) architectural mechanisms, 4) runtime support, and 5) applications and testbeds. This paper presents a survey of recent results and contributions of the Variability Expedition, organized along our major research thrusts.

2 Measurement and modeling

In this thrust we aim to quantify variation in contemporary hardware, and provide means to measure and model it, both online through circuit mechanisms and software and mathematical models. Our results show the magnitude of power variation in embedded and desktop processors, memories, and error variations in flash memories.

2.1 Sleep power variations in embedded class processors

Power consumed in an embedded class microprocessor chip is broadly classified into active mode and sleep mode. In one of our first efforts in variability characterization [47], we studied the temperature dependence of sleep mode power consumption in embedded processors. With shrinking geometries the ratio of sleep mode power to active mode power has been increasing (as high as 40% in chips fabricated using 65 nm technology) [39]. This is due to the inability to turn devices "off" effectively as device dimensions continue to shrink. Manufacturing spread in transistor parameters can cause up to 20× variation in sleep mode power [6] in addition to substantial variation with supply voltage and temperature. Similar sleep power spread (about 7×) is also observed in our relatively small sample set of 15 testchips fabricated using 45 nm technology. Specifically in context of embedded sensor platforms, which often are deployed in extreme ambient conditions, the variation in leakage power during the lifetime of a device may be substantial.

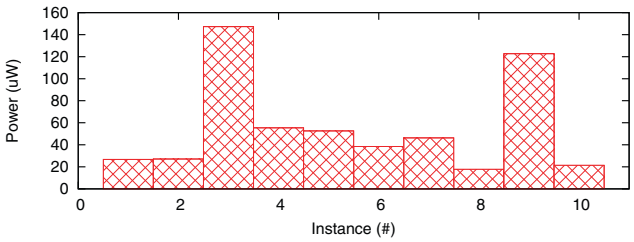

Figure 1: Cortex-M3 (Atmel SAM3U) sleep power at room temperature.

To illustrate the magnitude of sleep power variation in contemporary embedded processors, we measured sleep as a function of temperature across several instances of Atmel Cortex-M3 processors. The class of low-end 32-bit processors represented by the Cortex M3 is suitable for embedded applications where nodes perform data collection, aggregation, and inferences in a duty-cycled fashion. The variations we observed with the SAM3U are comparable to those found in other similar embedded processors [7].

For our measurements, we used ten identical SAM3U-EK development boards. These boards feature jumpers that allow power measurements for different components. We measured current and voltage on going into the SAM3U core, with all peripherals except for the real time clock disabled. We used a TestEquity 115F temperature chamber allowing control of ambient temperature with ±0.5 ˚C accuracy.

Figure 1 shows that the variation in sleep power across ten instances of SAM3U at room temperature is approximately 8×. Figure 2 shows the experimental data for sleep power consumption of the SAM3U instances across a temperature range fitted to an analytic model described in [47], using minimum mean square error criterion. As expected, individual processor instances exhibit large sleep power variations over the temperature range. While change in sleep power for any individual processor is monotonic, the magnitudes of variations are different so that relative rankings of different processors change over temperature. Over a temperature range of 20–60 ˚C, which is representative of the temperatures that embedded sensors deployed under unregulated and extreme ambient conditions often face (e. g. in factories, desert, etc.), total variation across all instances is 14×.

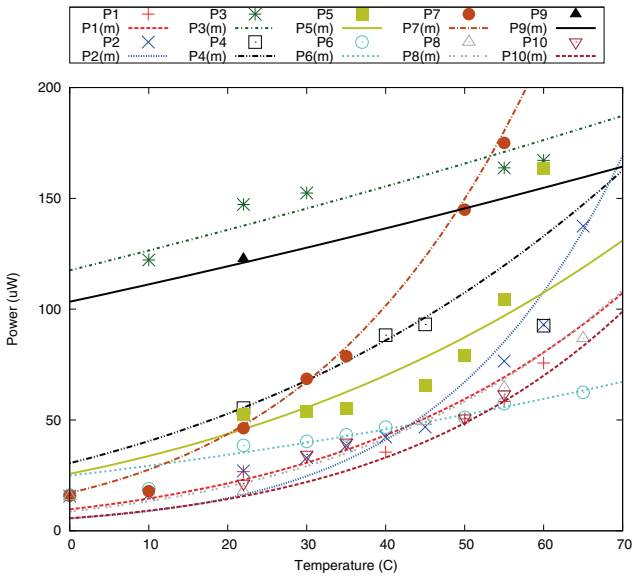

Figure 2: Cortex-M3 (Atmel SAM3U) measured and modeled variability of sleep power with temperature.

2.2 Active power variation in general purpose processors

We continued our efforts in power variation characterization with a study of active power consumption in Intel Core processors [2]. We measured power consumption for different benchmarks with six identical dual-core Intel Core i5-540M parts that feature both Hyper-Threading and Turbo Boost technologies. The processor utilizes the Nehalem architecture (32 nm), supports six sleep states (C-states) and ten frequencies (P-states) ranging from 1.33 GHz to 2.53 GHz and has maximum thermal design power (TDP) of 35 W. Our test setup used two identical development platforms, called Calpella, from Intel Labs which are highly instrumented with over fifty current sense resistors to isolate subsystem power. To isolate the CPU power we combined the measurements from three independent supply lines feeding different parts of the processor. We ran 19 SPEC CPU 2006 benchmarks on the six test processors.

Figure 3 plots the mean power consumption of each processor for the different benchmarks. We observe that for the benchmarks with high variations (bzip2, povray, soplex) the standard deviation across runs is also high, and thus, the actual process variation may be lower than the measurements indicate. While variation across runs accounts for significant variation across a single instance of a processor, we observe that the different processors perform consistently across benchmarks – e. g. processors P2, P5 and P6 have relatively high power consumption and P3 has a lower than average power consumption. Across all processors, variation ranges from 12% to 17%.

2.3 Power variation in memory subsystems

In addition to processors, variations in power consumption are also present in memory subsystems. We tested 22 DDR3 memory modules DIMMs, and found that power usage in DRAMs varies with operation type (write, read, and idle), data (with ones in the data typically costing more than zeros), and significantly across instances of the same model and across vendors in models with the same specification [19]. To perform our characterization, we used a standard PC platform augmented with a small 0.02 Ω resistor inserted on the V_{dd} line in between the DIMM and the motherboard. A thermal chamber was used to control temperature, and an Agilent 34411A digital multimeter sam-

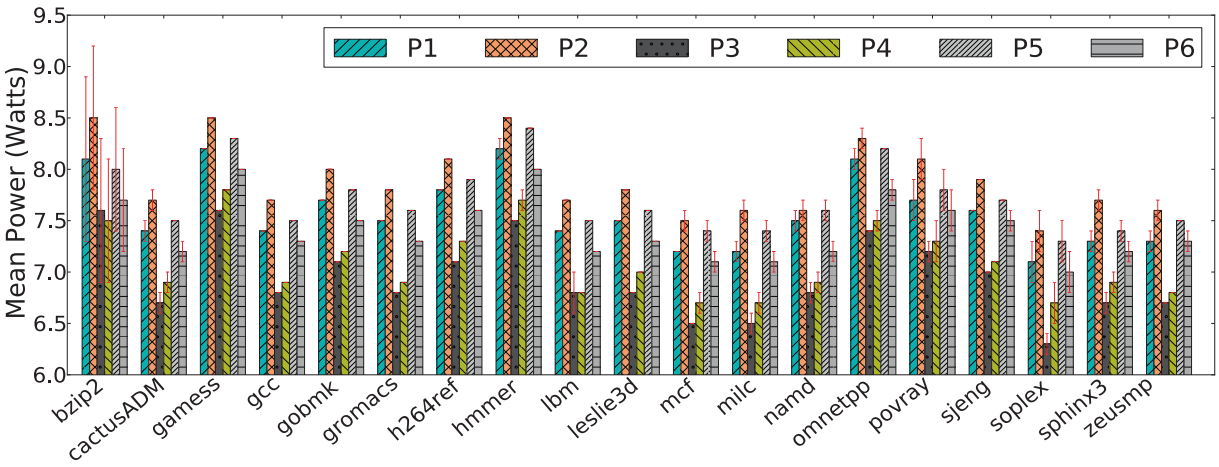

Figure 3: Power consumption of six Intel Core i5-540M processors for SPEC CPU 2006 benchmarks with Turbo Boost and Hyper-Threading disabled, C-States enabled at 2.53 GHz.

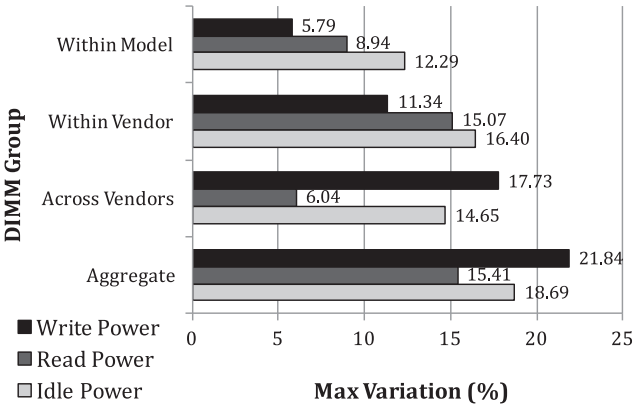

Figure 4: Maximum variations in read, write, and idle power in DDR3 memory DIMMs.

pled the voltage across the resistor at 10 000 samples/sec to derive power consumption.

We modified Memtest86+, a memory testing utility, to perform controlled experiments with memory accesses. We created a write function which wrote memory sequentially with a specified bit pattern, but never read it back. Similarly, a read function was created which only read memory sequentially without writing back. Each word location in memory could be initialized with an arbitrary pattern before executing the read test. The bit fade test, which was originally designed to detect bit errors over a period of DRAM inactivity, was modified to serve as an idle power test, with minimal memory usage. As our intent was primarily to measure power variability between different modules, we used sequential access patterns to avoid the effects of caches and row buffers. A summary of the variation found in our experiments is presented in Figure 4. The figure shows variations for write, read, and idle operation within DIMMs of the same model, across different models (with identical specification) of a same vendor, across different vendors, and overall across all of our DIMMs. Variations were up to 12.29% and 16.40% for idle power within a single model and for different models from the same vendor, respectively. In the scope of all tested modules, deviations were up to 21.84% in write power. Unlike in our measurements with processors, temperature had little effect (1–3%) across the −50 °C to 50 °C range [19].

2.4 Intracell variability in flash memories

As flash technology scales and the storage density increases, data errors become more prevalent. Flash memories fail in a variety of ways, particularly due to wearout with use, where after multiple erase and program cycles individual cells become unreliable due to charge trapping in the gate oxide [20]. Flash devices typically offer only vague guarantees on reliability, e. g., indicating correct operation from 10 000 to 100 000 erase cycles under specific assumption such as a ten-year "shelf life" for the data, random access patterns, and a loosely-specified error correction scheme. Correct operation of flash devices is further complicated by the use of multi-level cell technology. In early generations, each flash memory cell could represent two voltage levels and thus store a single bit (SLC). The demand for increased storage capacity has created the need to store more than a single bit per cell by simply representing more than two voltage levels. TLC (Triple Level Cell) technologies, for example, can store three bits per cell.

Adequate characterization and understanding of error patterns (and in intracell variability) in flash memories, can enable novel correction codes. In [15] we report on the observed errors measured from a TLC chip provided by an anonymous vendor. The errors were measured from sixteen blocks evenly divided across two planes. The following testing procedure was repeatedly performed. On the first cycle of every 100 program/erase (P/E) cycles, a block was erased, and random data was then written and finally read back for errors. On the other 99 cycles, the block was simply erased and the memory was programmed to simulate the aging of the device. In Figure 5, the Bit Error Rate (BER) is illustrated for the TLC chip tested over the course of its lifetime. It can be seen that over time, the BER increases dramatically but at different rates depending on which bit is programmed. The dominant trend from our results is that the "Symbol Error Rate" appears to be roughly the sum of the individual BERs of the MSB, CSB, and LSB. This suggests that whenever a cell-error occurs, with high probability only one of the three bits in the cell errs. More

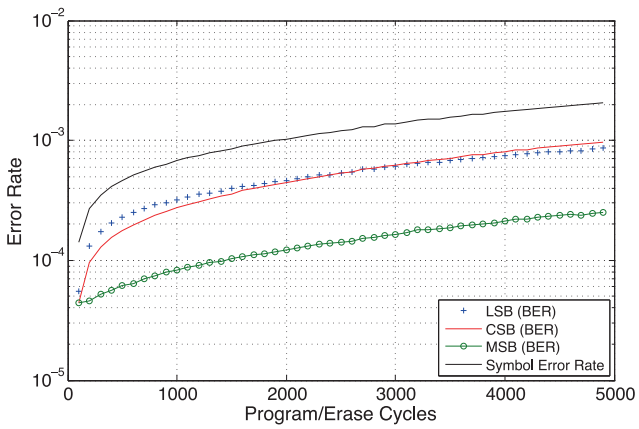

Figure 5: Error rates measured from a TLC flash device.

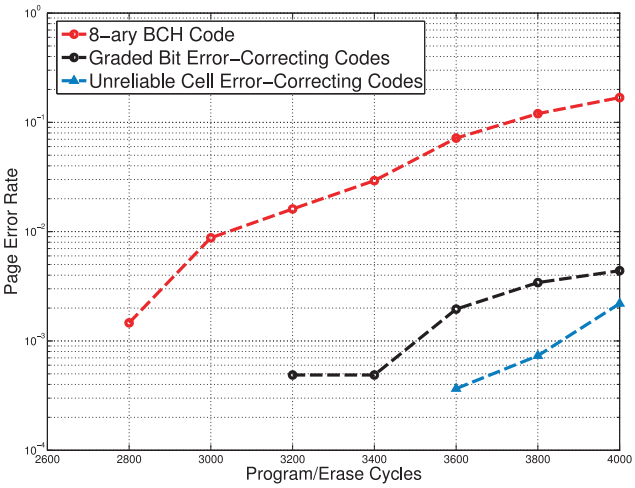

Figure 6: Page error rates in flash with different coding schemes.

specifically, 96.17% of cell-errors only had a single bit in error. In [15] we introduce new new codes to correct errors that mostly affect a single bit within each cell-error. These new codes also have the special property that they can correct the remaining few cell-errors with two or three bit-errors.

We also observed other trends in flash error patterns. For example, a small subset of cells in a block cause disproportionately many errors during the lifetimes of flash devices. Such errors take place when these unreliable cells are programmed to high levels. In [14], we modified the codes from [15] to avoid high levels in unreliable cells, further improving the lifetime of flash devices.

2.5 Variability-aware iterative graph-based information processing algorithms

A promising alternative to overdesigning and guardbanding is to adopt an *algorithmic error-tolerance approach*. In this approach, based on appropriate hardware models (e. g., [21, 22, 46]), we first establish closed-form fundamental performance limits of inference algorithms implemented on noisy hardware, and then offer a principled design methodology for the algorithm implementation that minimizes the effects of hardware variability. In particular, we quantify the effects on different operations within the algorithm have on the overall quality of inference and guide the implementation choice accordingly.

We first investigated the performance of popular iterative decoders for broadly deployed low-density parity check (LDPC) codes implemented on noisy hardware. Through a recursive analysis, we proved that different components of an iterative decoder have different effects on the error performance. Specifically, we showed theoretically and confirmed experimentally on the ERSA platform [30] that the decoder output error rate is dominated by the errors in the output messages at the variable nodes [24, 25, 45]. These findings enabled us to explore a new dimension in system design. We proposed an optimal resource allocation scheme applicable when computational units with varying degrees of reliability (and, naturally, cost) are available, a scenario that is appropriate when the variable nodes and checks nodes in an LDPC code have non-uniform degrees. Lower decoder error rate under the same implementation cost can therefore be achieved under the informed resource allocation [44].

Moreover, our theoretical analysis also revealed the inherent robustness of the iterative decoders: as long as the hardware error rate is small enough, the iterative decoders are still able to correct most of the errors from the communication channel such that the residual errors are due to unreliable hardware only. By observing that the checksum constraints in LDPC codes can detect both errors from the communication channel and from unreliable hardware, we proposed a scheme to detect permanent errors in the memory cells that store intermediate computations be-

Figure 7: Image denoising via BP: (a) original image (b) contaminated image (c) recovered image by noise-free BP (d) recovered image by nominal BP (e) recovered image by averaging BP.

tween successive decoding iterations. The proposed detection scheme utilizes the decoder structure (check-sum constraints) without adding any redundant components [25].

With the initial success of the analysis of LDPC decoders implemented on noisy hardware, we then broadened the scope of our analysis to include general belief propagation (BP) algorithms for inference over probabilistic graphical models. In [26], we characterized the BP algorithm on noisy hardware and proposed robust implementations of BP with mathematical guarantees. In particular, we introduced averaging BP, in which the effects of computation noise are reduced by averaging messages computed by BP over all up-to-date iterations. Theoretical analysis of averaging BP shows that the accuracy of noise-free BP can be achieved by averaging BP on noisy hardware. In the application example of belief propagation for image denoising, we demonstrated the effectiveness of mitigating computation noise by averaging BP, as shown in Figure 7.

3 Design tools and testing methodologies

This thrust includes the development of design and testing techniques for better variability monitoring, aging mitigation, failure detection and system resilience. SlackProbe is a flexible and efficient methodology for in situ performance monitoring. BTI-Gater is a cross-layer methodology to mitigate N/PBTI-induced clock skew on clock networks with clock gating features. An early-life-failure (ELF) detection algorithm based on local outlier factor (LOF) is developed to achieve high detection accuracy under temperature and voltage variations. New low overhead techniques are proposed which utilize architectural features to achieve self-repair of uncore components of SoC.

3.1 SlackProbe performance monitoring methodology

SlackProbe [28] is a flexible and efficient design technique for inserting in situ delay monitors. The working principle of SlackProbe is illustrated in Figure 8. In this example, by allowing monitors inserted at intermediate nets along critical paths, the number of monitors can be reduced and less than the number of path endpoints, i. e., two monitors as compared to four path endpoints. The delay of the circuits before the monitors (i. e., the monitored part) can be captured by the monitors, while the part of the circuits after the monitors (i. e., the margined part) will need additional delay margin.

SlackProbe is also flexible in trading-off the benefit and cost of monitoring. For critical path selection, we introduce a term called “opportunism window”. As illustrated in Figure 9, opportunism window is used to define the typical operating clock period that the circuits will operate at if no monitor flag is detected. The paths whose worst-case delay falls within the opportunism window are the ones that require monitoring. The size of opportunism window dictates the potential benefit of monitoring and determines the number of paths that requires monitoring. The additional monitor delay margin defines the monitoring cost and affects the monitor location candidates.

Our experimental results on commercial microprocessors show that with 5% monitor delay margin, SlackProbe

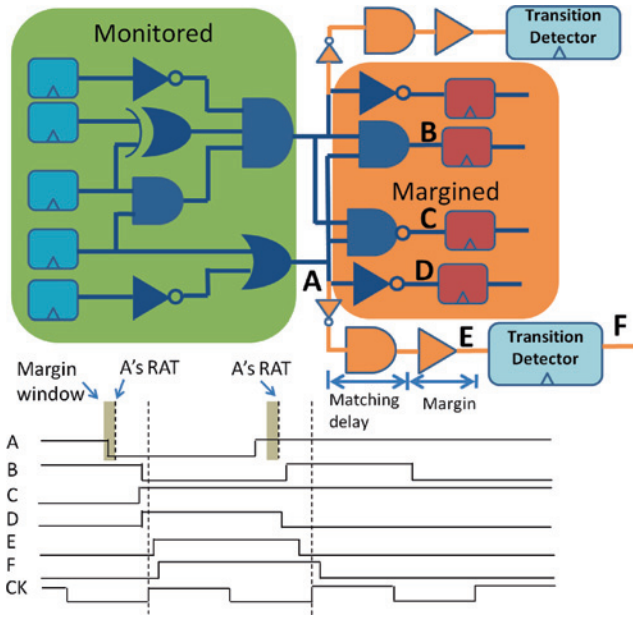

Figure 8: SlackProbe working principle.

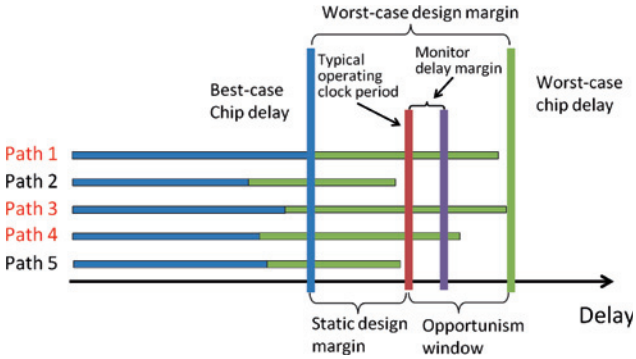

Figure 9: Opportunism window is the margin saving compared to worst-case design. Monitor delay margin refers to the delay margin of the delay matching chain.

can reduce the number of monitors by 12–16X as compared to the number of monitors inserted at path endpoint pins.

3.2 BTI-Gater clock gating methodology

BTI-Gater [29] is a cross-layer approach to mitigate aging effects due to N/PBTI on clock networks with clock gating features. Delay degradation induced by N/PBTI has strong dependence on workload, which can lead to different delay changes for clock branches under clock gating and thus additional clock skew. Since the imbalance is caused by the different signal patterns applied on each transistor, a circuit-level change is required for the integrated clock gating (ICG) cells to change the idle states of the clock branches. Cross-layer solutions from higher levels of system stacks are also necessary to mitigate this N/PBTI-induced skew because the clock gating implementation depends on the architecture and micro-architecture context and the actual usage will depend on the software behavior.

BTI-Gater proposes two ICG cell circuits that can alternate the idle state of the clock branches for each clock gating operation. The two ICG cells have different clock gating latencies, which can be matched to different architecture and micro-architecture requirements by a proposed architecture-level selection scheme. Experimental results show that BTI-Gater can balance the clock signal duto ratio on gated branches to close to the regular 50%, while guaranteeing a glitch-free clock signal with easy-to-verify timing contraints. Results on commercial processors show that BTI-Gater can effectively reduce N/PBTI-induced clock skew of up to 17 ps, which can be converted to up to 19.7% leakage power saving compared to pure design guardbanding.

3.3 LOF-based ELF detection

Early life failure (ELF), also known as infant mortality failure, occurs in defective integrated circuits that fail in the field before their expected product lifetime is up. Recent experimental results demonstrated that ELF defects cause a relative delay change (e. g., 20 ps) in a logic gate before the functional failures. Previous work assumed a quiescent situation, i. e., without temperature or voltage variations. We propose a new algorithm to detect ELF under temperature and voltage variations.

ELF detection is studied using the OpenSPARC T2 SoC design, which has a variety of components including 8 cores with 8 threads, memory subsystems and I/O components. A Boolean satisfiability (SAT)-based automatic test pattern generator (ATPG) is used to generate a special test pattern for the ELF detection. A simulation environment is developed that traces the test pattern and generates spice netlists for testing under temperature and voltage variations. Using the simulation platform, a local outlier factor (LOF)is investigated based on ELF detection algorithm in this project. LOF is a density-based outlier detection algorithm. The ELF-induced change shifted a propagation delay in a defected gate so that the ELF signature is detected by the LOF function [43].

Results from ELF detection simulations under temperature and voltage variations show that the LOF-based ELF detection algorithm has high detection probability and low false alarm probability. The simulation results show that 99.0% ELF defects are detected by LOF-based detection algorithm under $20\,^{\circ}\mathrm{C} \sim 70\,^{\circ}\mathrm{C}$ and $1.00\,\mathrm{V} \sim 1.05\,\mathrm{V}$ variations. In addition, the false alarm probability is less than 0.00060% under $-20\,^{\circ}\mathrm{C} \sim 100\,^{\circ}\mathrm{C}$ and $1.00\,\mathrm{V} \sim 1.05\,\mathrm{V}$ [43].

3.4 Self-repair of uncore components in robust system-on-chips

As technology scaling continues, large-scale systems-on-chip (SoCs) are becoming increasingly vulnerable to soft errors. The effects of soft errors in processor cores have been widely researched, but little is known about how soft errors affect uncore components, such as memory subsystem or I/O controllers. We set out to investigate uncore components, which account for a large footprint and power consumption in many SoCs. The two-fold goal of this project is to characterize the unique properties of soft errors on uncore components, while building efficient error resilience solutions for uncore components.

We studied soft errors on uncore components using the OpenSPARC T2 SoC design, which has a variety of uncore components including memory subsystems and I/O components [32]. To accurately model the error effects on a large-scale SoC, we needed an accurate and efficient simulation platform. We developed a mixed-mode simulation environment that combines a low-level RTL simulation for accurate error modeling and a high-level architectural simulator for a fast execution of real-world applications. Results from soft-error injection simulations showed that uncore errors have significant effect on system-level reliability (e. g., more than 1.4% of soft errors affect application outcome). Existing error detection and recovery mechanisms developed for processor cores, however, are not effective for uncore soft errors due to the unique prop-

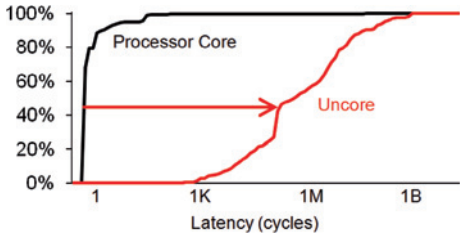

Figure 10: Comparison of error propagation latency between the soft errors in processor cores and the soft errors in uncore components.

erties of uncore errors such as long error propagation latency, up to 1 billion cycles (see Figure 10).

To provide efficient soft error resilience for uncore components and avoid the output commit problem, we developed an error detection and quick recovery technique that combines flip-flop hardening and logic parity soft error detection and recovery mechanism. The new solution eliminates 97% erroneous outcomes resulting from uncore soft errors at less than 2% of power and 2% of area overhead at chip-level [32].

4 Architectural mechanisms

In this section, we describe architectural mechanisms to enhance resiliency against device aging. NBTI-aware power-gating exploits the sleep state where a circuit is intrinsically immune to aging. However, the benefit of power-gating is strongly dependent on the fraction of time that a circuit spends in sleep mode. In practice, high power-gating factors are accompanied by significant performance degradation. We show how GPGPUs can instead arrange instructions and memory allocations to utilize the power-gating factor without sacrificing performance. We first propose an NBTI-aware very long instruction word (VLIW) assignment scheme that uniformly distributes the stress of instructions with the aim of minimizing aging of processing elements in GPUs [40]. ARGO [38], as a GPGPU register file management strategy, combats aging by leveling register banks though power-gating of stressful banks. Both of these techniques were applied the AMD Evergreen GPGPU architecture with general-purpose applications written in OpenCL. Finally, we discuss two of our recent contributions in energy-efficient reliability for cache and off-chip memories.

4.1 Adaptive VLIW assignment for GPGPU

This technique leverages a compiler-directed scheme that uniformly distributes the stress of instructions throughout various VLIW resource slots, results in a *healthy* code generation that keeps the underlying processing elements (PEs) healthy [40]. The key idea of an aging-aware compilation is to assign independent instructions uniformly to all slots: idling a *fatigued* PE and reassigning its instructions to a *young* PE through swapping the corresponding slots during the VLIW bundle code generation. This basically exposes the inherent idleness in VLIW slots and guides its distribution that does matter for aging. Thus, the job of dynamic binary optimizer, for K-independent instructions, is to find K-young slots, representing K-young PEs, among all available N slots, and then assign instructions to those slots. Therefore, the generated code is a "healthy" code that balances workload distribution through various slots maximizing the life time of all PEs. This adaptation flow is illustrated in Figure 11 that has four main steps: 1) Reading the aging sensors; 2) Disassembling the kernel, static code analysis, and calibration of predictions; 3) Uniform slot assignment; 4) Healthy code generation. Compared to the native kernels, the execution of healthy kernels not only imposes 0% throughput penalty but also reduces NBTI-induced voltage threshold shift: up to 49% and on average 34% in the presence of architectural power-

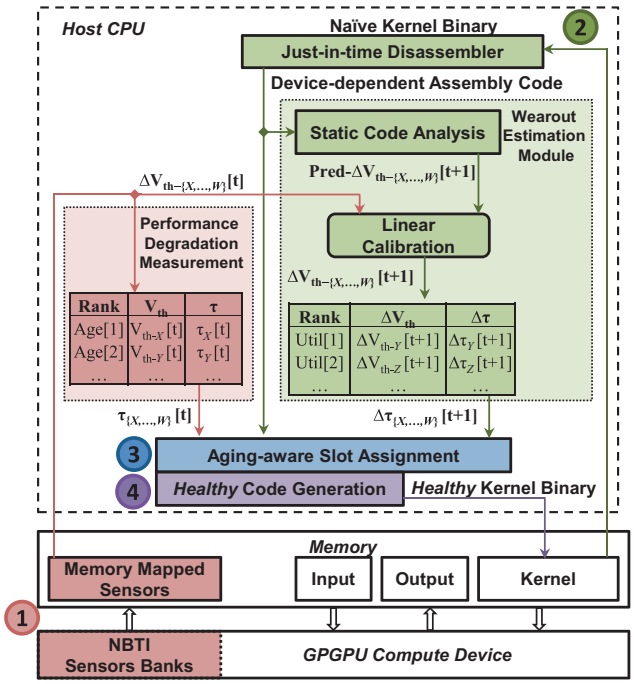

Figure 11: Aging-aware kernel adaptation flow.

gating supports. On average, the total execution time of the proposed adaption process is 13 millisecond.

4.2 Adaptive GPGPU register file allocation

ARGO (Aging-aware GPGPU register file allocation) [38] is an adaptive architectural technique for GPGPU register file (RF) allocation that exploits imbalanced RF utilization to ameliorate lifetime degradation by uniformly distributing the stress throughout the register files, without performance penalty. ARGO proactively and opportunistically exploits the underutilized portion of RF by proper leveling, accomplished through light-weight virtual sensing in conjunction with deliberated power-gating of stressful banks.

Experimental results for fifteen general-purpose kernels show the efficacy of ARGO through deliberated power-gating without throughput penalty: ARGO improves NBTI-induced threshold voltage degradation by up to 43% (on average 27%), improves the static noise margin in register files by up to 46% (on average 30%), and estimates a 54% reduction in leakage power.

4.3 Power/capacity scaling in fault-tolerant SRAM caches

Memories have historically been a major factor behind poor system-level energy proportionality [3]. This is partly because they contribute a relatively high portion of static leakage power due to the large number of transistors involved. In order to reduce SRAM supply voltage further, bit errors caused by collapsed noise margins must be tolerated for correct operation. There has been significant progress in this area over the past decade. Our approach [11, 17] leverages the insight that min-VDD is not the best metric for comparison of these fault-tolerant voltage-scalable SRAM caches. This is because the power, performance, and area overheads of the fault tolerance mechanism must be accounted for. In our scheme, we use a very simple mechanism where cache blocks are individually power gated as they become faulty at low voltage.

This capability introduces a new power/performance knob for improved energy efficiency and proportionality. Using the "fault inclusion property" [17] observed on our 45 nm SOI test chips [1], we allow for compact representation of fault maps for multiple runtime voltage levels. We use this mechanism for dynamic power/capacity scaling of the caches, illustrated in Figure 12. As voltage is reduced, some blocks become faulty and are disabled, which temporarily trades off cache capacity for power reduction in response to varying workload behavior. Our approach is able to achieve up to 75% average cache energy savings with only 2% performance and 5% area penalties across a suite of 16 SPEC CPU2006 benchmarks.

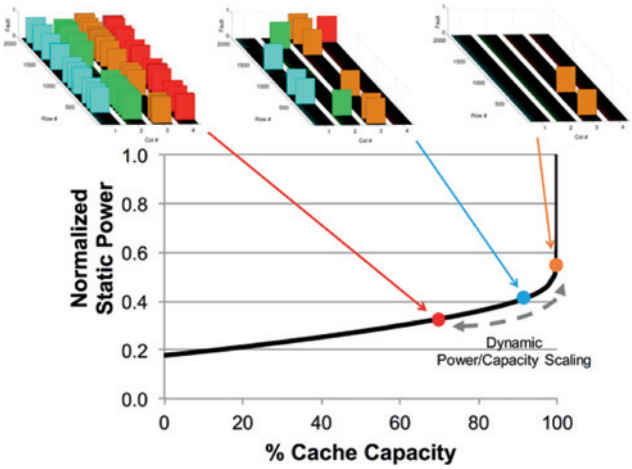

Figure 12: Dynamic Power/Capacity Scaling for SRAM caches.

4.4 Low-power chipkill correct

Chipkill correct is a type of memory-error correction that significantly improves memory reliability compared to the well-known single-error correction/double-error detection (SECDED) by providing correct memory accesses even when a DRAM device has failed completely. However, existing chipkill-correct solutions incur high power or storage overheads, or both because they use dedicated error-correction resources per codeword to per- form error correction. This requires high overhead for correction and results in high overhead for error detection.

Recent work on chipkill correct proposes reducing its memory power consumption of chipkill correct at the expense of increased storage overhead by reducing the number of data symbols per codeword.

In [27] we propose a novel chipkill-correct solution, multi-line error correction, that uses resources shared across multiple lines in memory for error correction to reduce the overhead of both error detection and correction. In this type of solution, more physical memory is required to provide the same amount of usable memory. Instead of simply reducing the number of data symbols per codeword, we explore reducing the number of check symbols per codeword while maintaining similar rates of detectable uncorrectable errors and silent data corruption. This allows the rank size to be reduced without having to increase storage overhead. Our evaluations in [27]

show that the proposed solution reduces memory power by a mean of 27%, and up to 38% with respect to commercial solutions, at a cost of 0.4% increase in storage overhead and minimal impact on reliability.

5 Compilers and runtime support

In the compilers and runtime support thrust of our project our objective is to build operating system and programming language constructs that help applications and system software adapt to (and exploit) variations in hardware. We highlight seven of our efforts in this space: the VaRTOS embedded kernel, that adapts applications and system duty cycle to meet lifetime goals with hardware suffering from power variations; the ViRUS framework for variability-aware algorithmic choice; the variation-aware OpenMP (VOMP) that reduces the cost of error correction for all work-sharing constructs in OpenMP V3.0 [41, 42]; the VaMV and ViPZonE memory allocators, which take hints expressing requirements applications allocating memory (e. g. low power or high reliability) and optimizes memory allocations and distribution accordingly; test amplification for verification of GPU kernels; and a Linux-governor based dynamic reliability manager for Android devices.

5.1 VaRTOS

Variations in power are particularly exaggerated for idle (or sleep) states, motivating the need to mitigate the effects of variability in systems whose operation is dominated by long idle states with periodic active states – that is, systems that are heavily duty-cycled. In systems where computation is severely limited by anemic energy reserves and where a long overall system lifetime is desired, maximizing the quality of a given application subject to these constraints is both challenging and an important step towards achieving high quality deployments.

VaRTOS [33] is an architecture and corresponding set of operating system abstractions that provide explicit treatment of both idle and active power variations for tasks running in real-time operating systems. Tasks in VaRTOS express elasticity by exposing individual *knobs* – shared variables that the operating system can tune to adjust task quality and correspondingly task power, maximizing application utility both on a per-task and system-wide basis. These knobs offer both a way for tasks to express elasticity in terms of utility and processing time and a way in which an operating system can fine-tune task energy consumption. Developers provide bounds on the values that each knob can assume and decide in what ways each knob is used, but optimization of these knob values is offloaded to the operating system and is done at runtime after accurate power and computational models have been constructed.

VaRTOS uses opportunistic sampling of temperature and instant power consumption to learn instance-specific sleep power, active power, and task-level power expenditure. Results on simulated hardware for several prototypical embedded sensing applications show that VaRTOS can reduce variability-induced deviations from expected battery lifetime from over 70% in many cases to under 2% in most cases and under 5% in the worst-case [33].

5.2 ViRUS

Task activation control mechanisms, such as those we designed with VaRTOS, are a valuable adaptation strategy for embedded sensing systems, where time spent in sleep mode may account for most of the energy dissipated by the system across its lifetime. For systems with significant variation in active power consumption, a *choice* of software to be executed provides further opportunities for optimization.

ViRUS (Virtual function Replacement Under Stress) [49] is a framework for variability and context-aware algorithmic choice. ViRUS is loosely related to polymorphic engines [50] in that it is used to transform sections of a program into different versions with alternate code paths that perform roughly the same functionality. Polymorphic engines are used to intercept and modify code transparently, typically for malicious purposes such as hiding malware functionality from anti-virus software. In ViRUS, the different code paths provide varying quality-of-service for different energy costs. Transformations are triggered by sensing and adapting to various sources of variability and energy stress. A block of code may be activated in ViRUS, for example, when processor temperature reaches a certain threshold. A second block may be activated when remaining battery capacity drops below a specified percentage. The different code blocks may be either standard library functions provided by the runtime system or alternative implementations provided by application programmers. Per-application configuration files determine when and under what circumstances transformations are triggered. The runtime system monitors sensors for energy stress and transparently triggers adaptation at appropriate times.

We demonstrated ViRUS with a framework for transparent function replacement in shared libraries and

a polymorphic version of the standard C math library in Linux. The ViRUS control framework uses less than 3 KB of RAM, and the polymorphic math library adds 10% memory overhead to its comparable single choice, high precision version. Application case studies using the polymorphic math library showed how ViRUS can tradeoff upwards of 4% degradation in application quality for a band of upwards of 50% savings in energy [49]. We are currently developing CaREDroid, an extended version of the ViRUS concept for the Android mobile operating system. In the future we also intend to explore enhanced cost/quality profilers that could automatically assign or suggest mutation rules for applications running under certain environmental and process variation conditions.

5.3 VOMP

We propose a variability-aware OpenMP (VOMP) programming environment, suitable for shared memory processor clusters, that relies upon modeling across the hardware/software interface [41]. VOMP is implemented as an extension to the OpenMP v3.0 programming model that covers various parallel constructs, including `task`, `sections`, and `for`. Using the notion of work-unit vulnerability (WUV) proposed here, we capture timing errors caused by circuit-level variability as high-level software knowledge. WUV consists of descriptive *metadata* to characterize the impact of variability on different work-unit types running on various cores. As such, WUV provides a useful abstraction of hardware variability to efficiently allocate a given work-unit to a suitable core for execution. VOMP enables hardware/software collaboration with online variability monitors in hardware and runtime scheduling in software. The hardware provides online per-core characterization of WUV metadata. This metadata is made available by carefully placing key data structures in a shared L1 memory and is used by VOMP schedulers. Our results show that VOMP greatly reduces the cost of timing error recovery compared to the baseline schedulers of OpenMP, yielding speedup of 3%–36% for tasks, and 26%–49% for sections. Further, VOMP reaches energy saving of 2%–46% and 15%–50% for tasks, and sections, respectively.

5.4 VaMV memory manager

VaMV (Variability-aware Memory Virtualization) [4] is a variability-aware memory allocator that allows programmers to partition their application's virtual address space into regions and create mapping policies for each region. Each policy can be designed to meet different requirements (e. g., power, performance, fault-tolerance). These user-defined and programmer-driven policies are then exploited by a dynamic memory management module which adapts to the underlying hardware, prioritizes the memory resources according to their characteristics (e. g., power consumption), and selectively maps program data to the best-fitting memory resource (e. g., highly- utilized data to low-power memory space).

VaMV adapts memory allocation to the underlying hardware and virtualizes the memory hierarchy, while opportunistically exploiting techniques such as voltage scaling to reduce on-chip power consumption and power consumption variability present in off-the-shelf off-chip memories. Experimental results on embedded benchmarks show that, by exploiting SRAM voltage scaling, RAM power variability, and efficient dynamic policy-driven variability-aware memory allocation, VaMV is capable of reducing dynamic power consumption by 63% on average while reducing total execution time by an average of 34% [4].

5.5 ViPZonE prototype Linux memory manager

Motivated by our observations regarding power variations in commodity DRAMs as described by Section 2.3, we prototyped ViPZonE, a DRAM power variability-aware OS kernel [5, 12, 16, 18]. Based on Linux kernel 3.2, ViPZonE's modified physical page allocator allows applications to specify low or high power space when dynamically allocating memory from the heap. Using our instrumented testbed, we were able to achieve up to 27% main memory power reduction with no more than a modest 5% performance overhead for a set of PARSEC parallel computing benchmarks by exploiting the power variability present in the system's DIMMs. Figure 13 depicts the hardware/software stack for ViPZonE-enabled systems.

5.6 Test amplification

The CUDA (Compute Unified Device Architecture) programming model and platform has recently enjoyed tremendous success in parallel computing. Due to its massive parallelism, loose synchronization mechanism, and fine-grained memory sharing, however, CUDA programs are hard to get right, and are difficult to analyze with existing static or dynamic approaches. Static techniques

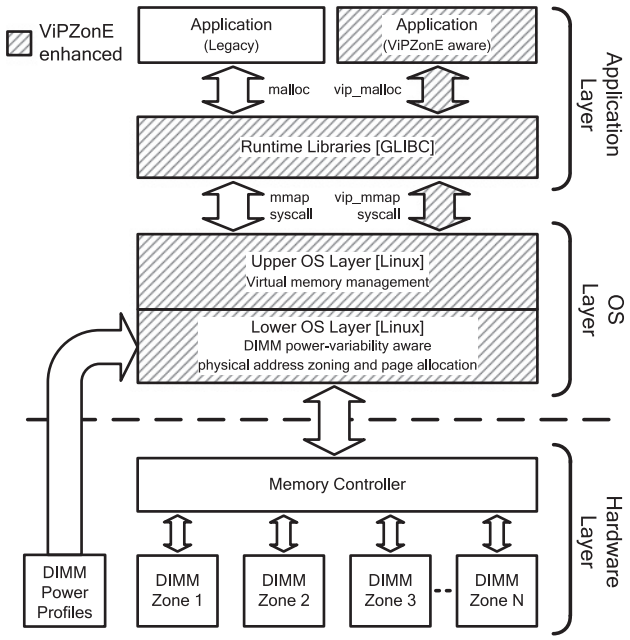

Figure 13: ViPZonE hardware/software architecture.

are thwarted by the complexity of the sharing patterns. A useful static analysis would have to find a succinct symbolic representation for the sets of addresses accessed by each thread. It is common for CUDA code to use modular arithmetic or bit-shifting to access memory indices according to complex linear or even non-linear patterns which makes such analyses difficult. Dynamic techniques are challenged by the combinatorial explosion of thread interleavings and space of possible data inputs: any reasonable number of tests would represent a small subset of the possible behaviors of the system.

To attack these challenges, we employ test amplification, a general notion wherein a single dynamic run can be used to learn much more information about a program's behavior than is directly exhibited by that particular execution. First, we run a dynamic analysis where we log the behavior of the kernel with some fixed test input and under a particular thread interleaving. Second, we use a static information flow analysis to compute the property-integrity inputs, namely, the input variables that actually flow-to, or affect the integrity of, the variables appearing in the property to be verified. Finally, we amplify the result of the test to hold over all the inputs that have the same values for the property-integrity inputs. In [31] we empirically demonstrate the effectiveness of test amplification for verifying race-freedom and determinism over a large number of standard GPU kernels, by showing that the result of verifying a single dynamic execution can be amplified over the massive space of possible data inputs and thread interleavings.

5.7 Dynamic Reliability Management in Linux

Dynamic Reliability Management (DRM) enables tradeoffs between processor degradation and performance at runtime. Reliability is periodically assessed, and processor operating conditions are controlled to limit the degradation source (i. e. temperature and voltage). The goal of DRM is maintain a predefined target reliability within a predefined target lifetime.

Implementation of DRM requires degradation sensors to be integrated on processors. However, these are available only on prototypes today. To enable the evaluation of DRM techniques on existing mobile devices, we developed an online reliability emulation framework and implemented it on a real Android device [34]. The framework is implemented at the level of the operating system and exploits existing hardware interfaces. It samples real voltage and temperature data and applies mathematical models to emulate the reliability degradation of the real platform.

Work in [36] introduces a Linux governor – i. e., a controller for operating conditions – for reliability management. The proposed governor implements the reliability *Borrowing Strategy* that we proposed in [35], which accounts for the execution of workload with varied quality requirements. We developed our DRM governor on a Snapdragon S4 mobile development tablet, which has an asynchronous quad Krait CPU with frequency from 380 MHz up 1.67 GHz and voltage from 0.95 V to 1.25 V. Cores have independent voltage/frequency (V/f) settings and fixed operating V/f points. Therefore, changing frequency automatically changes voltage.

Figure 14 shows our Reliability Governor for workload-aware DRM and the Debug and Monitor Infrastructure that exports data to the user space. The framework has a Long Term Controller (LTC) which activates every Long Interval (LI = days it takes for reliability to change) and a Short Term Controller (STC) which activates every Short Interval (SI = 1 jiffy). Tasks are divided in Highly critical (H) and Less critical (L). For providing good user experience, H tasks must be executed at maximum frequency. The goal of DRM is to let the core reliability R be above a target reliability Rt at the predefined target lifetime. The controller acts by monitoring temperature and adjusting voltage and frequency to adjust to task load and reliability requirements. Results in [36] demonstrate how our reliability governor can achieve upwards of 100% improvement in life-

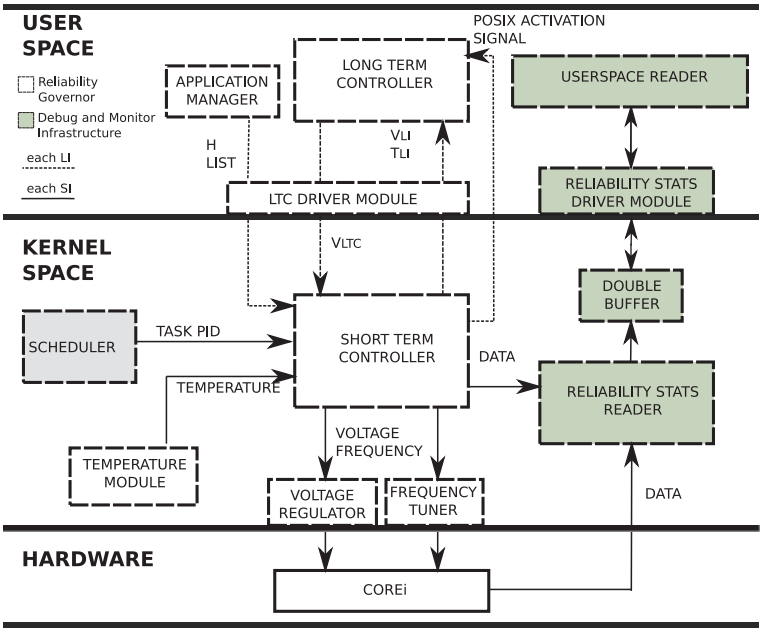

Figure 14: Block diagram for the Linux reliability governor.

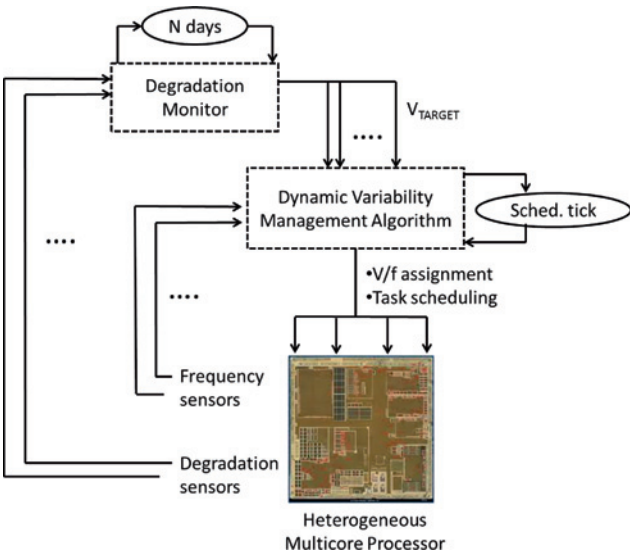

Figure 15: DVM Framework.

time compared to a traditional (performance-driven) governor while still providing high performance for critical tasks.

Our recent work also proposed a novel Dynamic Variability Management (DVM) strategy to improve performance of mobile multiprocessor subject to lifetime constraints [37]. The solution exploits sensors to monitor degradation and performance variability and adapts through a OS-level scheduling and frequency scaling algorithm. The runtime management comprehensively accounts for (1) variation on degradation rate; (2) variation in performance; (3) variation in application-specific quality requirements; (4) lifetime constraint and (5) ambient temperature variation.

Figure 15 shows the DVM framework. A degradation monitor activates periodically at a large time scale (in the order of days) and it samples sensors to determine the degradation status of each core. Based on this, it outputs a reference value of voltage V_{TARGET}. The constraint on lifetime is met if the average voltage over time is below V_{TARGET}. This is the input to the Dynamic Variability Management algorithm, which executes two subroutines at the rate of scheduling ticks: (1) T-boost extended DVFS and (2) Variability-aware task allocation. The proposed strategy has been implemented and tested on a real Android device and demonstrated to achieve up to 160% performance improvement relative to the state-of-the-art.

6 Applications and testbeds

In the applications and testbeds thrust of the Variability Expedition we aim to build tools and prototypes for the development and demonstration of variability aware and tolerant systems. In this section highlight two efforts from this thrust: the VarEMU variability emulator and the Ferrari testbed.

6.1 VarEMU

VarEMU [48] is an extension to the QEMU virtual machine monitor that serves as a framework for the evaluation of variability-aware software techniques. VarEMU provides users with the means to emulate variations in power consumption and in fault characteristics and to sense and adapt to these variations in software. Through the use (and dynamic change) of parameters in a power model, users can create virtual machines that feature both static and dynamic variations in power consumption. Faults may be injected before or after, or completely replace the execution of any instruction. Power consumption and susceptibility to faults are also subject to dynamic change according to an aging model. A software stack for VarEMU features precise control over faults and provides virtual energy monitors to the operating system and processes. This allows users to precisely quantify and evaluate the effects of variations on individual applications.

In VarEMU, timing and cycle count information is extracted from the code being emulated. This information is fed into a variability model, which takes configurable parameters to determine energy consumption and fault variations in the virtual machine. Energy consumption and susceptibility to faults are also subject to dynamic change according to an aging model. Control over faults and virtual energy sensors are exported as "variability registers" mapped into memory that is accessible to the software being emulated, closing the loop. This information is exposed through a variability driver in the operating system, which can be used to support software adaptation policies. Through the use of different variability emulation parameters that capture instance-to-instance, environmental, and age-related variation, VarEMU allows users to evaluate variability-aware software adaptation strategies across a statistically significant number of hardware samples and scenarios.

6.2 OrangeFerrari testbed

The goal for this project is to build an end-to-end UnO machine prototype, which can be used as a testbed for demonstrating and validating some of the expedition work. RedCooper [1] is our first attempt in building such testbed. It is based on a testchip taped-out using a 45 nm IBM SOI technology with dual-Vth libraries. The testchip contains an ARM Cortex-M3 microprocessor, our performance sensors [8] and some other on-chip leakage power sensors. A testbed board is built to drive the testchip and offers board-level power sensing and voltage/frequency scaling capabilities. A customized embedded operating system (OS) is ported based on CoOS [9]. With RedCooper testbed, we have successfully demonstrated a variability-aware applications running with the expedition duty cycling work [33].

OrangeFerrari testbed is our generation end-to-end variability testbed. The testchip is taped-out using the same 45 nm IBM SOI technology. The testchip block diagram and layout are shown in Figure 16. Compared to our earlier version of the testchip, this one is based on the same ARM Cortex-M3 microprocessor, but with larger SRAM and more peripherals supports. More varieties of on-chip sensors are also implemented, including DDRO performance sensors, leakage power sensors, temperature sensors and gate oxide breakdown sensors.

The testbed board block diagram is shown in Figure 17. Compared to RedCooper testbed, OrangeFerrari

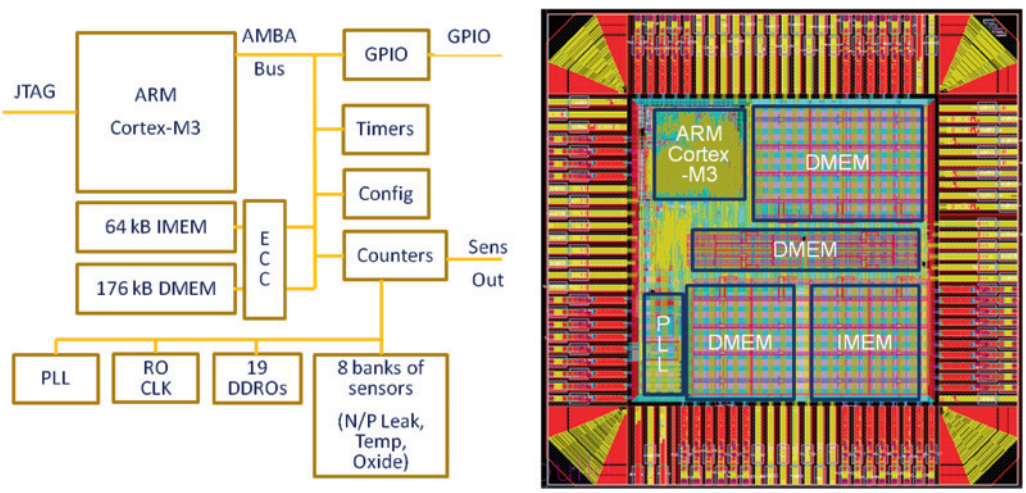

Figure 16: Ferrari testchip block diagram and layout illustration.

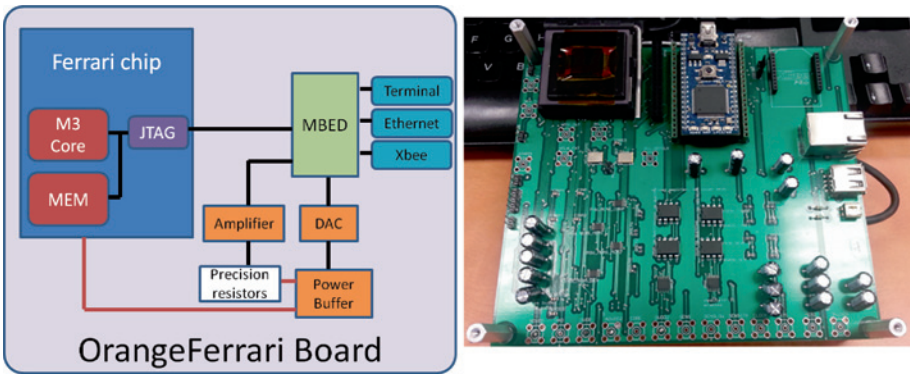

Figure 17: OrangeFerrari testbed block diagram and board photo.

testbed also includes additional network connection capabilities with both wired (ethernet) and wireless (Xbee [10]) options. The OS porting is based on FreeRTOS [13]. A customized routine is implemented to handle the communication with the on-board power sensors or through the network.

7 Conclusion

The NSF Variability Expedition started in 2010 with a promise of delivering holistic hardware/software solutions to the problem of process and environmental variability in semiconductor components. As our efforts evolved, there was a natural shift from variation characterization and modeling to variability-aware system design and implementation. Our initial characterization efforts showed that contemporary processors, memory, and storage systems already suffer from considerable variation in power and reliability. Our work in design tools and testing methodologies introduced low-cost variability sensing and mitigation mechanisms to hardware designs. Our architecture research introduced new adaptive memory controllers as well as error management and correction techniques. In runtime systems, we introduced new schedulers, runtimes, and memory management systems that can adapt to variations in the power and error characteristics of individual and distributed platforms. Finally, our work on testbeds, particularly the Ferrari test chip and board now provide a complete evaluation platform for embedded variability-aware software. Our recent efforts in international collaboration, particularly through the joint annual review and joint colloquium with the German Research Foundation (DFG) Priority Program SPP 1500 project [23] in 2014 provided opportunities for mutual awareness and nascent joint research. Moving forward, we expect to see a greater focus on variable quality and approximate software and hardware, as well as on the interplay between cost and reliability at different abstract layers of the system.

Funding: This material is based upon work supported by the National Science Foundation under Grant Nos. 1029030, 1028888, 1029783, 1028831, and 1029025. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

1. Y. Agarwal, A. Bishop, T.-B. Chan, M. Fotjik, P. Gupta, A. B. Kahng, L. Lai, P. Martin, M. Srivastava, D. Sylvester, L. Wanner, and B. Zhang. Redcooper: Hardware sensor enabled variability software testbed for lifetime energy constrained application. Technical report, NanoCAD Lab, University of California, Los Angeles, 2014.
2. B. Balaji, J. McCullough, R. K. Gupta, and Y. Agarwal. Accurate characterization of the variability in power consumption in modern mobile processors. In *Proceedings of the 2012 USENIX Conference on Power-Aware Computing and Systems*, HotPower'12, pages 8–8, Berkeley, CA, USA, 2012. USENIX Association.
3. L. A. Barroso and U. Hölzle. The Case for Energy-Proportional Computing. *IEEE Computer*, 40(12):33–37, December 2007.
4. L. A. D. Bathen, N. D. Dutt, A. Nicolau, and P. Gupta. Vamv: Variability-aware memory virtualization. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, pages 284–287, March 2012.
5. L. A. D. Bathen, M. Gottscho, N. Dutt, A. Nicolau, and P. Gupta. ViPZonE: OS-Level Memory Variability-Driven Physical Address Zoning for Energy Savings. In *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, pages 33–42. IEEE/ACM/IFIP, 2012.
6. S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De. Parameter variations and impact on circuits and

microarchitecture. In *Proc. Design Automation Conf.*, DAC '03, pages 338–342. ACM, 2003.
7. D. Bull, S. Das, K. Shivashankar, G. S. Dasika, K. Flautner, and D. Blaauw. A power-efficient 32 bit arm processor using timing-error detection and correction for transient-error tolerance and adaptation to pvt variation. *IEEE Journal of Solid-State Circuits*, 46(1):18–31, 2011.
8. T.-B. Chan, P. Gupta, A. B. Kahng, and L. Lai. Synthesis and analysis of design-dependent ring oscillator (ddro) performance monitors. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 22(10):2117–2130, Oct 2014.
9. CooCox Project. CoOS operating system. http://www.coocox.org/, 2014.
10. Digi International Inc. Xbee radio. http://www.digi.com/xbee/, 2014.
11. N. Dutt, P. Gupta, A. Nicolau, A. BanaiyanMofrad, M. Gottscho, and M. Shoushtari. Multi-Layer Memory Resiliency. In *Proceedings of the Design Automation Conference (DAC)*, San Francisco, CA, USA, June 2014. ACM Press.
12. N. Dutt, P. Gupta, A. Nicolau, L. A. D. Bathen, and M. Gottscho. Variability-Aware Memory Management for Nanoscale Computing. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC)*, Yokohama, Japan, 2013. IEEE.
13. FreeRTOS Project. Freertos. http://www.freertos.org, 2013.
14. R. Gabrys, F. Sala, and L. Dolecek. Coding for unreliable flash memory cells. *IEEE Communication Letters*, 18(9):1491–1494, July 2014.
15. R. Gabrys, E. Yaakobi, L. M. Grupp, S. Swanson, and L. Dolecek. Tackling intracell variability in tlc flash through tensor product codes. In *ISIT*, pages 1000–1004. IEEE, 2012.
16. M. Gottscho. ViPZonE: Exploiting DRAM Power Variability for Energy Savings in Linux x86-64. Technical report, University of California, Los Angeles, 2014.
17. M. Gottscho, A. BanaiyanMofrad, N. Dutt, A. Nicolau, and P. Gupta. Power / Capacity Scaling: Energy Savings With Simple Fault-Tolerant Caches. In *Proceedings of the Design Automation Conference (DAC)*, ACM, 2014.
18. M. Gottscho, L. A. D. Bathen, N. Dutt, A. Nicolau, and P. Gupta. ViPZonE: Hardware Power Variability-Aware Memory Management for Energy Savings. *IEEE Transactions on Computers*, 2014.
19. M. Gottscho, A. A. Kagalwalla, and P. Gupta. Power Variability in Contemporary DRAMs. *IEEE Embedded Systems Letters*, 4(2):37–40, June 2012.
20. L. M. Grupp, A. M. Caulfield, J. Coburn, S. Swanson, E. Yaakobi, P. H. Siegel, and J. K. Wolf. Characterizing flash memory: Anomalies, observations, and applications. In *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, pages 24–33, Dec 2009.
21. J. Han, J. Gao, P. Jonker, Y. Qi, and J. A. B. Fortes. Toward hardware-redundant, fault-tolerant logic for nanoelectronics. *IEEE Des. Test. Comput.*, 22(4):328–339, July 2005.
22. R. Hegde and N. R. Shanbhag. Soft digital signal processing. *IEEE Trans. VLSI Syst.*, 9(6):813–823, December 2001.
23. J. Henkel, L. Bauer, J. Becker, O. Bringmann, U. Brinkschulte, S. Chakraborty, M. Engel, R. Ernst, H. Hartig, L. Hedrich, A. Herkersdorf, R. Kapitza, D. Lohmann, P. Marwedel, M. Platzner, W. Rosenstiel, U. Schlichtmann, O. Spinczyk, M. Tahoori, J. Teich, N. When, and H. Wunderlich. Design and architectures for dependable embedded systems. In *9th International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 69–78, Oct 2011.
24. C.-H. Huang and L. Dolecek. Analysis of finite-alphabet iterative decoders under processing errors. In *IEEE ICCASP*, 2013.
25. C.-H. Huang, Y. Li, and L. Dolecek. Gallager B LDPC decoder with transient and permanent errors. *IEEE Trans. Commun.*, 62(1):15–28, January 2014.
26. C.-H. Huang, Y. Li, and L. Dolecek. Belief propagation algorithms on noisy hardware. *IEEE Trans. Commun.*, 63(1):11–24, Jan 2015.
27. X. Jian, H. Duwe, J. Sartori, V. Sridharan, and R. Kumar. Low-power, low-storage-overhead chipkill correct via multi-line error correction. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '13, pages 24:1–24:12, New York, NY, USA, 2013. ACM.
28. L. Lai, V. Chandra, R. Aitken, and P. Gupta. Slackprobe: A low overhead in situ on-line timing slack monitoring methodology. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '13, pages 282–287, San Jose, CA, USA, 2013. EDA Consortium.
29. L. Lai, V. Chandra, R. Aitken, and P. Gupta. Bti-gater: An aging-resilient clock gating methodology. *Emerging and Selected Topics in Circuits and Systems, IEEE Journal on*, 4(2):180–189, June 2014.
30. L. Leem, H. Cho, J. Bau, Q. A. Jacobson, and S. Mitra. ERSA: Error resilient system architecture for probabilistic applications. In *DATE*, 2010.
31. A. Leung, M. Gupta, Y. Agarwal, R. Gupta, R. Jhala, and S. Lerner. Verifying gpu kernels by test amplification. In *Proceedings of the 33rd ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '12, pages 383–394, New York, NY, USA, 2012. ACM.
32. Y. Li, E. Cheng, S. Makar, and S. Mitra. Self-repair of uncore components in robust system-on-chips: An opensparc t2 case study. In *2013 IEEE International Test Conference (ITC)*, pages 1–10, Sept 2013.
33. P. Martin, L. Wanner, and M. Srivastava. Runtime optimization of system utility with variable hardware. *ACM Trans. Embed. Comput. Syst.*, 14(2):24:1–24:25, February 2015.
34. P. Mercati, A. Bartolini, F. Paterna, L. Benini, and T. S. Rosing. An on-line reliability emulation framework. In *Embedded and Ubiquitous Computing (EUC), 2014 12th IEEE International Conference on*, pages 334–339, Aug 2014.
35. P. Mercati, A. Bartolini, F. Paterna, T. S. Rosing, and L. Benini. Workload and user experience-aware dynamic reliability management in multicore processors. In *Design Automation Conference (DAC), 2013 50th ACM / EDAC / IEEE*, pages 1–6, May 2013.
36. P. Mercati, A. Bartolini, F. Paterna, T. S. Rosing, and L. Benini. A linux-governor based dynamic reliability manager for android mobile devices. In *Proceedings of the Conference on Design, Automation & Test in Europe*, DATE '14, pages 104:1–104:4, 3001 Leuven, Belgium, Belgium, 2014. European Design and Automation Association.
37. 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.
38. M. Namaki-Shoushtari, A. Rahimi, N. Dutt, P. Gupta, and R. K. Gupta. Argo: Aging-aware gpgpu register file allocation. In *2013 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 1–9, Sept 2013.
39. R. Puri, L. Stok, and S. Bhattacharya. Keeping hot chips cool. In *Proceedings of the 42nd annual Design Automation Conference*, DAC '05, pages 285–288, New York, NY, USA, 2005. ACM.
40. A. Rahimi, L. Benini, and R. K. Gupta. Aging-aware compiler-directed vliw assignment for gpgpu architectures. In *Proceedings of the 50th Annual Design Automation Conference*, DAC '13, pages 16:1–16:6, New York, NY, USA, 2013. ACM.
41. A. Rahimi, D. Cesarini, A. Marongiu, R. K. Gupta, and L. Benini. Improving resilience to timing errors by exposing variability effects to software in tightly-coupled processor clusters. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 4(2):216–229, June 2014.
42. A. Rahimi, A. Marongiu, P. Burgio, R. K. Gupta, and L. Benini. Variation-tolerant openmp tasking on tightly-coupled processor clusters. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, pages 541–546, March 2013.
43. M. Sauer, Young Moon Kim, Jun Seomun, Hyung-Ock Kim, Kyung-Tae Do, Jung Yun Choi, Kee Sup Kim, S. Mitra, and B. Becker. Early-life-failure detection using sat-based atpg. In *Test Conference (ITC), 2013 IEEE International*, pages 1–10, Sept 2013.
44. S. M. S. Tabatabaei, C.-H. Huang, and L. Dolecek. Optimal design of a Gallager B noisy decoder for irregular LDPC codes. *IEEE Commun. Lett.*, 16(12):2052–2055, December 2012.
45. Y. Tabatabaei, S. M. Sadegh, H. Cho, and L. Dolecek. Gallager B decoder on noisy hardware. *IEEE Trans. Commun.*, 61(5):1660–1673, May 2013.
46. J. von Neumann. Probabilistic logic and the synthesis of reliable organisms from unreliable components. *Automata Studies*, pages 43–98, 1956.
47. L. Wanner, R. Balani, S. Zahedi, C. Apte, P. Gupta, and M. Srivastava. Variability-aware duty cycle scheduling in long running embedded sensing systems. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, pages 1–6, March 2011.
48. 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.
49. L. Wanner and M. Srivastava. ViRUS: Virtual function replacement under stress. In *6th Workshop on Power-Aware Computing and Systems (HotPower '14)*, Broomfield, CO, October 2014. USENIX Association.
50. T. Yetiser. Polymorphic viruses: Implementation, detection, and protection. Technical report, VDS Advanced Research Group, 1993.

Bionotes

Lucas Wanner
Electrical Engineering Department, University of California, Los Angeles
wanner@ucla.edu

Liangzhen Lai
Electrical Engineering Department, University of California, Los Angeles

Abbas Rahimi
Computer Science and Engineering Department, University of California, San Diego

Mark Gottscho
Electrical Engineering Department, University of California, Los Angeles

Pietro Mercati
Computer Science and Engineering Department, University of California, San Diego

Chu-Hsiang Huang
Electrical Engineering Department, University of California, Los Angeles

Frederic Sala
Electrical Engineering Department, University of California, Los Angeles

Yuvraj Agarwal
School of Computer Science, Carnegie Mellon University

Lara Dolecek
Electrical Engineering Department, University of California, Los Angeles

Nikil Dutt
Department of Computer Science, University of California, Irvine

Puneet Gupta
Electrical Engineering Department, University of California, Los Angeles

Rajesh Gupta
Computer Science and Engineering Department, University of California, San Diego

Ranjit Jhala
Computer Science and Engineering Department, University of California, San Diego

Rakesh Kumar
Electrical and Computer Engineering Department, University of Illinois at Urbana Champaign

Sorin Lerner
Computer Science and Engineering Department, University of California, San Diego

Subhasish Mitra
Department of Electrical Engineering, Stanford University

Alexandru Nicolau
Center for Embedded Computer Systems, University of California, Irvine

Tajana Simunic Rosing
Computer Science and Engineering Department, University of California, San Diego

Mani B. Srivastava
Electrical Engineering Department, University of California, Los Angeles

Steve Swanson
Computer Science and Engineering Department, University of California, San Diego

Dennis Sylvester
Department of Electrical Engineering and Computer Science, University of Michigan

Yuanyuan Zhou
Computer Science and Engineering Department, University of California, San Diego