Cong J, Duwe H, Kumar R *et al.* Better-than-worst-case design: Progress and opportunities. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 29(4): 656–663 July 2014. DOI 10.1007/s11390-014-1457-2

Better-Than-Worst-Case Design: Progress and Opportunities

Jason Cong[1] (丛京生), *Fellow*, *ACM*, *IEEE*, Henry Duwe[2], Rakesh Kumar[2], *Member*, *ACM*, *IEEE* and Sen Li[1] (李　森)

[1] *Computer Science Department, University of California, Los Angeles, CA 90095-1596, U.S.A.*

[2] *Electrical and Computer Engineering Department, University of Illinois at Urbana-Champaign, Urbana, IL 61801, U.S.A.*

E-mail: cong@cs.ucla.edu; {duweiii2, rakeshk}@illinois.edu; senli@cs.ucla.edu

Received April 3, 2014; revised June 3, 2014.

Abstract Today, designers are forced to reduce performance and increase power requirements in order to reserve larger margins that are required due to the greater variability introduced by smaller feature sizes and manufacturing variations of modern IC designs. The better-than-worst-case design can both address the variability problem and achieve higher performance/energy efficiency than the worst-case design. This paper surveys the progress to date, provides a snapshot of the most representative methods in this field, and discusses the future research directions of the better-than-worst-case design.

Keywords better-than-worst-case, error resilience, variability

1 Introduction

As transistor sizes scale down to the nanometer range, a number of variability sources are introduced, including process variation and environmental variation. Process variation is caused by dopant density variation, stress occurring during manufacturing, and edge geometry variation. The sources of environmental variation, on the other hand, can be overheating and voltage fluctuations. As device sizes further shrink, the impact of variation becomes greatly amplified. Conventional design techniques conservatively introduce guardbands to address the variability problem, but these can result in significant circuit performance/energy overhead.

There is a common consensus that the relative magnitude of variation is going to get much worse as we approach the limit of CMOS scaling and start exploring different nanotechnologies as alternatives to CMOS. The better-than-worst-case (BTWC) design paradigm has been proposed to account for this problem. The goal of BTWC design is to remove some (or all) design guardbands and run the circuit at a higher clock frequency to improve performance, or at a lower supply voltage to increase energy efficiency. Also, a recent goal has been to devote only as much hardware to a problem as it is required for a "good enough" solution.

Broadly speaking, there are two categories in BTWC design. The first category includes designs with error detection and correction circuits[1-7]. This kind of design relies on some form of scaling of operational parameters (such as the supply voltage), combined with architectural/circuit-level techniques for efficient error detection and correction, to achieve higher performance/energy efficiency while maintaining correctness of output. The second category of BTWC design takes advantage of approximate computing, which allows its output to have certain errors as long as the output quality is acceptable by the user[8-10]. It is based on either scaling of operational parameters or modifying/redesigning the hardware architecture.

Another challenge in BTWC design is the ability to estimate error probabilities. The existing techniques include tagged probabilistic simulations[11], Monte Carlo timing simulations[12], binary-decision-diagram-based techniques[13], and LUT-based regression[14].

In the remainder of this paper, we first discuss some background and definitions used throughout the paper (Section 2). We then discuss the category of BTWC design with error detection and correction circuits in Section 3. Section 4 describes approximate computing BTWC designs in detail. Section 5 discusses the techniques to estimate error probability in BTWC designs. We discuss some future research challenges in Section 6, and conclude the paper in Section 7.

2 Background

For conventional architectures, the performance of the circuit is only dependant on the worst-case delay, or

Survey
The research is partially supported by the National Science Foundation of USA under Grant No. CCF-0903541.

the static longest path in the logic network. However, (generally speaking) the performance of the BTWC architectures is related not only to the worst-case delay, but also to the possible timing error rate. There are three main performance metrics of BTWC architecture: error probability[1], error significance[14], and expected delay[2].

• Error probability: a determination of the performance of circuit C that will be implemented in an error recovery scheme. For example, the Razor architecture[4] uses extra clock cycles when it sees a timing error. Specifically, referring to Fig.1, let P^{clk}_{wrong} be the probability of circuit switching at time $d = 1/clk$ or later. In this case the error probability on a BTWC architecture is $P^{clk\_main}_{\text{wrong}}$ where *clk_main* is the fast clock period in Fig.1. $P^{clk\_main}_{\text{wrong}}$ is defined as:

$$P^{clk\_main}_{\text{wrong}} = Pr(C(x)_{clk\_main} \neq C(x)_{clk\_shadow} | x \in \text{inputs}).$$

• Expected delay: an estimate of the performance of a BTWC circuit. This can be easily computed by dividing the total processing time by the amount of data processed, since different input sequences can take different amounts of time to propagate to the outputs. The expected delay of a BTWC circuit with the target clock period d can be computed as follows:

$$ExpDelay(d) = d \times (1 - P^d_{\text{wrong}}) + (d + \tau_{\text{error}}) \times P^d_{\text{wrong}},$$

where τ_{error} is the time to detect and recover from an error.

• Error significance: a determination of the magnitude of errors. We more precisely define error significance as the signed average difference between correct and erroneous results.

3 Designs with Error Correction Circuits

The key concept of BTWC design is to remove some (or most) design guardbands and run the circuit at a higher clock frequency to improve performance, or, at a lower supply voltage, to increase energy efficiency. As a consequence, error detection and correction circuits are introduced in order to guarantee correctness of a program. In this section we begin by introducing the Razor architecture[4] which is a classic representative of this field, and discuss an efficient logic synthesis methodology[1] for BTWC designs.

3.1 Razor Architecture

The Razor architecture[4] was originally developed for microprocessors with a well-defined pipeline architecture. It takes advantage of dynamic voltage scaling (DVS) to obtain power savings, and introduces error detection and correction circuits to eliminate timing errors introduced by the sub-critical supply voltage. Razor tunes the supply voltage by monitoring the error rate during operation. It represents a trade-off between the power savings from operating at a lower supply voltage and the power penalty from the error detection and correction circuits.

The exact Razor pipeline stage is shown in Fig.1. Each flip-flop is accompanied with a shadow latch. The shadow latch is clocked with a delayed clock. In this way, it can catch any errors caused by the main flip-flop being clocked early. Specifically, if the values of the main flip-flop and the shadow latch are different during comparison, then an error is detected and an error signal is generated. In the subsequent cycle, the valid data in the shadow latch is restored into the main flip-flop, during which all the pipeline stages are stalled. Therefore, if an instruction fails in pipeline stage L1, it will get the correct value from the shadow latch in one cycle and keep executing in pipeline stage L2. In other words, it guarantees the forward progress of a failing instruction with a small performance penalty.

Razor uses a voltage control system to maintain a constant error rate Err_{ref}. During runtime it samples the current error rate Err_{cur} at regular intervals, and adjusts the supply voltage based on $Err_{\text{diff}} = Err_{\text{ref}} -$

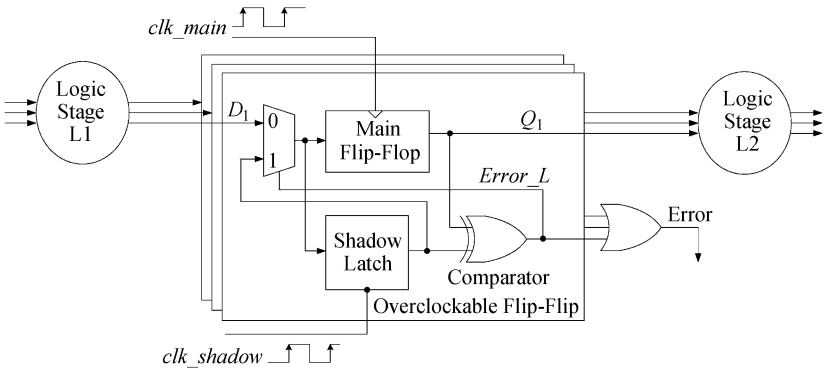

Fig.1. Razor stage with shadow latches and control lines[4].

Err_{cur}. Razor implements a proportional control system that adjusts supply voltage in proportion to Err_{diff}.

3.2 Logic Synthesis for BTWC Architecture

The Razor architecture was originally designed for pipelined microprocessor architecture. Later work[1-2] generalized this concept to synchronous circuits controlled by a finite state machine (FSM) in order to improve the circuit performance. The modified FSM is called BTWC FSM. Fig.2 shows the difference between a standard FSM and a BTWC FSM. All the registers in a standard FSM architecture are converted into Razor registers — ones with shadow registers as in Fig.1. The input registers are modified in a way that can be buffered and stalled by the timing error signal. An additional Razor stabilization stage is added at the end of the BTWC FSM architecture.

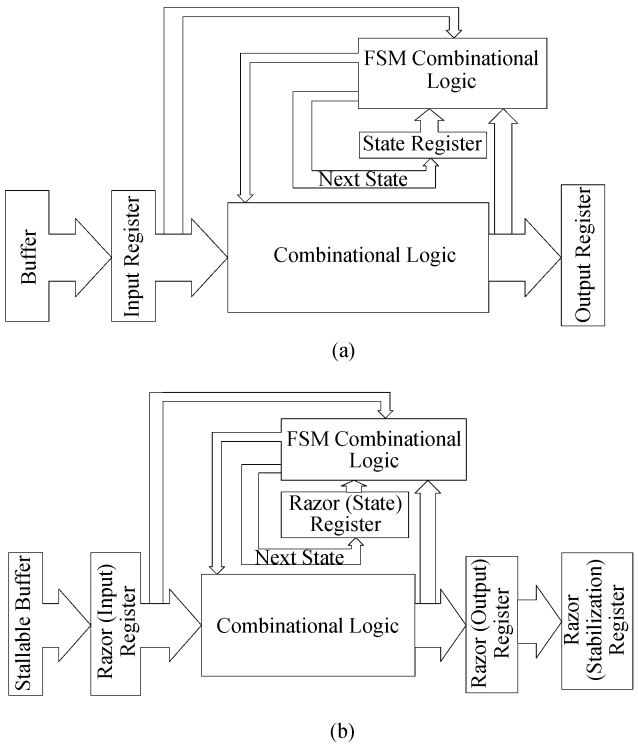

Fig.2. Standard and BTWC FSMs[1]. (a) Standard FSM architecture. (b) Stallable FSM architecture.

Given a specific performance metric, circuit optimization can be divided into two categories. The first category includes techniques that transform the circuit into another one of the same representation while optimizing the performance metric. The operations in this category can be further divided into three groups: operations that substitute nodes in a logic network, operations that collapse nodes, and operations that decompose nodes. Substitute operations are less suitable for BTWC architecture since they can create a significant amount of changes, and as a result, their impact is very difficult to evaluate. Collapse operations are mainly used for creating a better environment for decomposition operations. [1] adapts *balance* decomposition algorithms for the BTWC architecture and can achieve a reduction in timing error probability by 2.3X with only a 4% area overhead.

The second category includes techniques that convert a circuit into an alternative representation (e.g., technology mapping) while optimizing the performance metric. For example, the work in [2] develops an efficient technology mapping algorithm *BTWMap* for general synchronous designs that can operate better than the worst-case delay. *BTWMap* minimizes the probability of an error while assuming the circuit is clocked at the maximum possible clock speed (2X the worst-case delay). It then minimizes the area with the user-assigned target clock period and area/performance trade-offs. From the experiment results on MCNC benchmarks, *BTWMap* can reduce the expected delay by 13% with a 26% area overhead. Its modified version, *BTWMap+area*, reduces the expected delay by 11% with only 10% area overhead.

Later work proposes making frequent local changes to and-inverter graphs (AIGs)[5]. Since the effect on the error probability from each change must be estimated, using timing simulations to derive switching probabilities is prohibitively expensive. Instead, switching probabilities are estimated using PI switching probabilities and the results are derived by traversing the circuit network in a topological order. The following equations estimate the switching probabilities of gate i:

$$S_i = Tog_i \times Pr(g_1 = \text{NCV}) \times Pr(g_2 = \text{NCV}) \times \cdots \times Pr(g_k = \text{NCV}),$$

where Tog_i is the switching probability of path i's inputs, g_j is a side input, and NCV stands for non-controlling value (e.g., logic "1" for an AND gate). The switching probabilities are combined with a Gaussian distribution, for gate delay variation estimation, to estimate the error probability for each path. AIG supergates and their subtrees are optimized in descending order of potential error probability reduction. When applied to the ISCAS 1989 benchmark circuits, these optimizations increased throughput by 17.51% on average for a 1.13% increase in area.

3.3 Slack Redistribution for BTWC Architectures

One approach to improving the performance of Razor-like architectures is to manipulate path error dis-

tributions to reduce the error rate at a given operation point. [7] confirms the existence of a "wall" of slack postulated by the critical operating point (COP) hypothesis in the OpenSPARC T1 processor. The COP hypothesis states that a critical operating voltage, Vc, exists; below this, massive path error rates overwhelm recovery methods such as those in Razor. [7] proposes a power-aware slack redistribution to allow a reduction in voltage beyond Vc to gradually increase error rates. Fig.3 shows the slack wall and proposed slack redistribution. Slack redistribution attempts to increase slack for the most frequently exercised paths, since such paths are most likely to increase the error rate. By iteratively decreasing the target voltage and optimizing frequently exercised paths using cell sizing swaps, slack redistribution provides a lower operating voltage at a given error rate constraint. Path slack optimizations which increase the power are rejected.

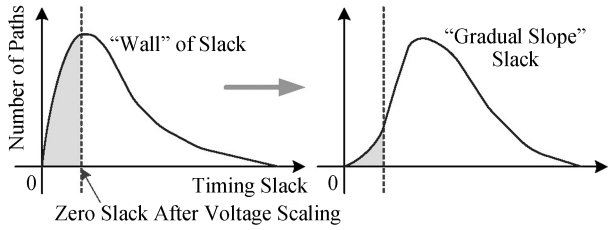

Fig.3. Slack wall.

The work in [7] demonstrates a 23% power savings over the baseline design for an error rate of 1%. Power-aware slack redistribution also has better power savings over other techniques which reduce the error rate at a given voltage. This is because power-aware slack redistribution makes cost effective optimizations where the reduction in voltage outweighs the power increase from cell sizing. Power-aware slack redistribution thus increases the range over which Razor can be applied. When coupled with Razor, power-aware slack redistribution improves these power benefits and also allows an increase in the throughput for a target voltage due to a reduction in error rates. The observed power and throughput benefits of slack redistribution cannot be replicated by simply tightening timing constraints.

The work in [6] proposes CAD methodologies designed specifically for use with hardware error recovery mechanisms such as Razor. [6] recognizes that if a recovery mechanism is being used, paths with a negative slack can be further downsized to reduce power. Thus recovery-driven design allows infrequently-exercised paths to violate timing and produce errors even at nominal voltage. In a manner similar to [7], [6] iteratively reduces the target voltage and upsizes cells on frequently exercised paths which have negative slack. Each iteration recovery-driven design selects infrequently exercised paths with a positive slack and allows them to have a negative slack. Once the paths are partitioned into positive and negative slack sets, [6] downsizes cells while respecting the partitioning. This process continues until a minimum power netlist is achieved.

The work in [6] uses functional simulation to identify activated paths for consideration during optimization. Gate-level simulation produces the number of cycles in which each activated path is toggled. Dividing the number of simulation cycles in which negative slack paths were toggled, by the total cycles simulated, results in an accurate error rate estimate during optimization. Recovery-driven design can achieve a power savings of up to 24% against traditional place-and-route at an error rate of 1%. Recovery-driven design shows a 21% reduction in power compared to the power-aware slack redistribution when both are used with Razor.

4 Approximate Computing Designs

Approximate computing designs improve power and performance (from removing guardbands) by allowing infrequent errors during execution. Many applications can tolerate such non-zero error rates. Three approaches to approximate design are the manipulation of the path error distribution[8], intelligent redesign of the entire architecture[9], and re-architecting hardware modules to produce a desirable path error distribution[10].

4.1 Error Distribution Manipulation

Path error distributions can be manipulated to allow larger power savings for a given error rate. [8] investigates the dynamic use of two adders with different error distributions. A simple ripple-carry adder (RCA) provides a graceful degradation in error rate as voltage is lowered beyond the critical voltage, Vc, for a given cycle time. A Kogge-Stone adder (KSA) has a lower Vc for the same cycle time, but its error rates explode to 1.0 once its voltage is lowered beyond Vc. Depending on application error-rate requirements, the adder with the lower power is selected at runtime. Applications that cannot tolerate errors or can only tolerate very low error rates use KSAs, while applications that can tolerate higher, but not catastrophic, error rates use RCAs. When applied to the SAD kernel of H.264, the dynamic solution uses 20% to 60% less power than a static adder.

4.2 Probabilistic Pruning

There are several ways to realize approximate computing design. One way is to rely on some form of

scaling of operational parameters like techniques discussed in Subsection 4.1. Another way is probabilistic pruning[9] which prunes portions of circuits that have a lower probability of being active.

The overall flow of this probabilistic tuning technique is as follows. Given an initial circuit design, it first estimates the probability of being active for each path based on simulation or mathematical model. Then, based on whether the application requires weighted circuits or not, it performs weighted probabilistic pruning/uniform probabilistic pruning, and computes an error rate in the pruned circuit using either mathematical estimation or functional simulation. If the error rate is higher than the target error rate, it then undoes the last pruning, and the resulting design is the final pruned design; otherwise it continues the pruning process.

The gain achieved by probabilistic pruning is additive in that it can be combined with either standard techniques that achieve higher energy/performance efficiency or techniques that use voltage scaling to achieve additional gains.

4.3 Reconfigurable Approximate Adder

Another approach to approximate computing is to re-architect processor modules (e.g., adders or multipliers) to produce acceptable error distributions. [10] details a reconfigurable, gracefully degrading adder (GDA). The GDA adjusts its approximation error rate during runtime to meet the varying quality of result and throughput demands of an application. A GDA gracefully degrades by predicting carry-ins to the most significant bits using carry-out information from a variable number of preceding bits. A reduction in the use of carry-out information reduces computational effort (i.e., lower power and less delay). The GDA gracefully degrades the error rate with respect to computational effort because the most significant bits are least affected by the farthest input bits. Such graceful degradation results in a significantly higher throughput than traditional approximate adders for a given error rate, as shown in Fig.4. [10] uses GDA in the discrete cosine transform (DCT) of a JPEG encoding. The GDA results in a 21% increase in throughput and an increase of over 3 dB of peak signal-to-noise ratio (PSNR) compared to the next best approximate adder.

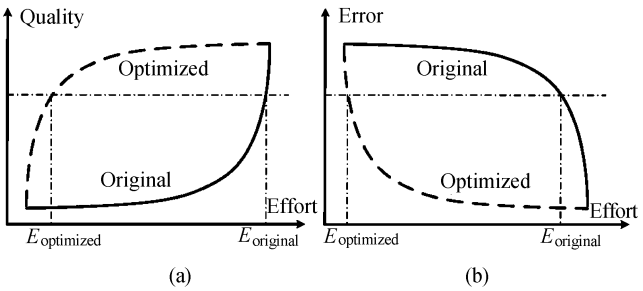

Fig.4. Quality-effort and error-effort trade-off curves.

5 Error Probability Estimation

5.1 Monte Carlo Timing Simulations

The work in [12] models circuit delays that are due, not only to static variations, but also to dynamic variations such as temperature. Each gate is modeled as a collection of identical transistors which have a Gaussian distribution for delay. The distribution's parameter is broken into a nominal value, an inter-die value, and systematic and random intra-die variations. Monte Carlo simulations are performed. Each sample involves the selection of these parameters for each gate and then a timing simulation of the circuit. While providing accuracy within 5% of HSPICE simulations, this methodology decreases the execution time by five orders of magnitude.

5.2 Binary-Decision-Diagram-Based Techniques

Another method of error estimation uses the timed characteristic function (TCF) for each node[13]. $TCF(n, t-)$ contains all input vectors which make the output n stable no later that t. The TCF for a node (and each primary output) is calculated recursively using gate-specific equations such as the following for an AND gate ($c = a \wedge b$):

$$\begin{aligned} TCF(c, t-) = & CF(a') \cdot CF(b) \cdot \\ & TCF(a, t-d)-) + CF(a) \cdot CF(b') \cdot \\ & TCF(b, (t-d)-) + CF(a') \cdot CF(b') \cdot \\ & [TCF(a, (t-d)-) + TCF(b, (t-d)-)] + \\ & CF(a) \cdot CF(b) \cdot TCF(a, (t-d)-) \cdot \\ & TCF(b, (t-d)-). \end{aligned}$$

$CF(a')$ includes all input vectors resulting in a final value of logic "0". Binary-decision diagrams (BDDs) are used to calculate CF for each node. The probability of error for a given cycle time is calculated by the fraction of input vectors for which each primary output is met over the total number of possible input vectors. Unfortunately, binary-decision diagrams require exponential space complexity in the worst case, which may limit their utility for large circuits.

5.3 Tagged Probabilistic Simulations

Binary-decision diagrams require exponential space complexity, and timing simulations require many test inputs. Tagged probabilistic simulations (TPS) provide

an efficient solution by building a signal and transition probability waveform at each node[11]. These waveforms represent the instantaneous probability that each node is a logic "1", or transitioning to or from a logic "1". Each node's probability waveform is partitioned into sets of signals that start at value x and end at value y. Such tagged probability waveforms can be propagated from primary inputs to primary outputs using propagation equations and delays for each gate type (AND, OR, etc.). Error probability for a primary output is estimated by taking the difference of the primary output probabilities between the minimum safe cycle time, Tp, and a cycle time, Tp', less than Tp. TPS is good for tree-structured circuits, but additional heuristic adjustments are required for reconvergent paths.

5.4 LUT-Based Regression

Many previous studies on error probability estimation have focused on the problem of estimating a single approximate module's error rate in isolation. In order to effectively use approximate hardware modules, CAD tools must be able to efficiently estimate the output quality of a design composed of approximate modules. A recent work proposes using look-up tables (LUTs) coupled with regression to estimate error composition[14]. The proposed tool characterizes the error rate and error significance responses of each module in a library relative to a given input distribution. The value distribution for each module is also characterized based on the input distribution. The value distribution at each node within a design is determined by a single topological pass. The intrinsic error rate (ER_{in}) and error significance (ES_{in}) estimates for individual nodes can be directly looked up based on the input distributions at each node. The output estimates (ER_z and ES_z) can be generated based on the input estimates (ER_a, ER_b, ES_a, and ES_b) using the following regression equations where $\alpha_{\{\text{in},\,c,\,p\}}$ are regression coefficients:

$$\begin{aligned} ER_z =& 1 - 10^{\alpha_c} \times (1 - ER_{\text{in}})^{\alpha_{\text{in}}} \times \\ & ((1 - ER_a) \times (1 - ER_b))^{\alpha_p}, \\ ES_z =& \alpha_{\text{in}} \times ES_{\text{in}} + \alpha_p \times (ES_a + ES_b) + \alpha_c. \end{aligned}$$

The LUT-based regression technique improves the accuracy of error estimation by 3.75x over interval-based error composition estimation techniques, while running 8.4X faster for a set of MAC circuits.

6 Challenges and Opportunities for BTWC Design

Several considerations ultimately dictate the efficacy and practicality of BTWC designs. First, it is critical that a small amount of reduction in reliability can be traded for significant power benefits. Also, hardware errors arising from running at a reduced reliability need to be tolerated by applications in the context of approximate designs. Often, a large fraction of hardware components must be redesigned to allow meaningful power/reliability trade-offs[5-7,15]. Furthermore, the error distribution needs to be carefully managed in order for the applications to tolerate errors[8,10], even for ASICs. It is critical that effective and general hardware design methodologies first be developed that allow controlled power/reliability trade-offs. The resulting hardware should also generate errors from a distribution with known properties. For example, hardware can be designed such that the error distribution is (largely) input independent[16]. Programming models and runtime systems, etc., that exploit these trade-offs can then follow.

Algorithmic transformation is another frontier for BTWC design. Algorithm selection for an application has been conventionally dictated by concerns such as performance, and more recently, energy. However, transforming applications into algorithmic forms that are significantly more resilient can allow greater hardware approximation. Examples include transforming applications into iterative problems[17] or sampling problems[18], where many hardware errors simply increase the runtime instead of affecting output quality[19]. Many of these algorithms also lead to highly parallel domain-specific hardware[19]. Unfortunately, the algorithmic transformations are often application-specific, and may not be easily generalizable. The transformation is also largely manual — thus research is needed on automatic transformations. Finally, many transformed implementations do not provide the same guarantees as the baseline implementations. Research is needed to provide bounds on output quality for the alternate algorithmic implementations.

Opportunities also exist to support hardware optimizations that make certain controlled software approximations practical. For example, hardware support for branch herding[20] — forcing all copies of a program executing on an SIMD processor to branch in the same direction for certain branches in order to eliminate the performance overhead of control divergence — leads to significantly higher performance than performing branch herding on commodity hardware. Such hardware optimizations need to be co-designed with software, and the compiler. The approximate system co-design must address several challenges: modeling the effect of an approximation on QoS, modeling the effect of composing approximations, applying approximations automatically, guaranteeing QoS, etc. A promising di-

rection is working on analyzing critical instructions in a program[21]: given an error tolerance threshold of an application, it develops an automated analysis technique in the compilation time to analyze which instructions are critical and which are not (in the sense that its failure will not violate the error tolerance threshold). Note that these challenges exist for hardware approximation as well. Recent work[14], for example, shows that error composition may become an intractable problem for large hardware designs.

Furthermore, general-purpose programmable processors (GPPs) face additional challenges. The difficulty of exactly modeling the manifestation of faults as software errors, except in case of limited functional under-design (e.g., reduced precision), makes hardware approximation difficult. Other challenges include fault containment, expression of exact reliability requirements for a given instruction/data, and exact mapping between programmer intent and operational parameters. Furthermore, the additional layers of abstraction make some of the challenges mentioned earlier even worse (e.g., modeling the effect of an approximation on QoS, modeling the effect of composing approximation, applying approximations automatically, guaranteeing QoS). The challenges are particularly worse when errors are due to parametric under-design. Many of these challenges that need to be addressed before a credible hardware approximation-based system can be built for GPPs.

Finally, as we enter the era of dark silicon, we believe that future computer architectures will be rich in accelerators. Recent studies address the issues of accelerator sharing, composition, scheduling, management, and virtualization[22-24]. How to introduce imprecise accelerators and manage error prorogation in accelerator-rich architectures is another important research direction.

7 Conclusions

BTWC design is one of the architecture design trends in the future because of the greater variability introduced by the smaller feature size of modern IC designs. With this in mind, this survey paper discusses the current literature in BTWC design, ranging from designs with error detection/correction circuits and approximate computing designs, to error probability estimation techniques. Our paper outlines the future research challenges and opportunities for BTWC designs.

References

[1] Cong J, Minkovich K. Logic synthesis for better than worst-case designs. In *Proc. Int. Symp. VLSI Design, Automation and Test*, April 2009, pp.166-169.

[2] Cong J, Minkovich K. Mapping for better than worst-case delays in LUT-based FPGA designs. In *Proc. the 16th FPGA*, Feb. 2008, pp.56-64.

[3] Austin T, Bertacco V, Blaauw D, Mudge T. Opportunities and challenges for better than worst-case design. In *Proc. ASP-DAC*, Jan. 2005, pp.2-7.

[4] Ernst D, Kim N S, Das S *et al.* Razor: A low-power pipeline based on circuit-level timing speculation. In *Proc. the 36th MICRO*, Dec. 2003, pp.7-18.

[5] Liu Y, Ye R, Yuan F *et al.* On logic synthesis for timing speculation. In *Proc. ICCAD*, Nov. 2012, pp.591-596.

[6] Kahng A B, Kang S, Kumar R, Sartori J. Recovery-driven design: A power minimization methodology for error-tolerant processor modules. In *Proc. the 47th DAC*, June 2010, pp.825-830.

[7] Kahng A, Kang S, Kumar R, Sartori J. Designing a processor from the ground up to allow voltage/reliability tradeoffs. In *Proc. the 16th HPCA*, June 2010.

[8] Narayanan S, Sartori J, Kumar R, Jones D. Scalable stochastic processors. In *Proc. DATE*, Mar. 2010, pp.335-338.

[9] Lingamneni A, Enz C, Nagel J L *et al.* Energy parsimonious circuit design through probabilistic pruning. In *Proc. DATE*, Mar. 2011.

[10] Ye R, Wang T, Yuan F *et al.* On reconfiguration-oriented approximate adder design and its application. In *Proc. ICCAD*, Nov. 2013, pp.48-54.

[11] Tosson A, Garg S, Anis M. Tagged probabilistic simulation based error probability estimation for better-than-worst case circuit design. In *Proc. the 21st VLSI-SoC*, Oct. 2013, pp.368-373.

[12] Ganapathy S, Canal R, Gonzalez A, Rubio A. Circuit propagation delay estimation through multivariate regression-based modeling under spatio-temporal variability. In *Proc. DATE*, Mar. 2010, pp.417-422.

[13] Wan L, Chen D. DynaTune: Circuit-level optimization for timing speculation considering dynamic path behavior. In *Proc. ICCAD*, Nov. 2009, pp.172-179.

[14] Chan W T, Kahng A B, Kang S *et al.* Statistical analysis and modeling for error composition in approximate computation circuits. In *Proc. the 31st ICCD*, Oct. 2013, pp.47-53.

[15] Kahng A, Kang S, Kumar R *et al.* Slack redistribution for graceful degradation under voltage overscaling. In *Proc. the 15th ASP-DAC*, Jan. 2010, pp.825-831.

[16] Abdallah R, Lee Y H, Shanbhag N R. Timing error statistics for energy-efficient robust DSP systems. In *Proc. DATE*, Mar. 2011.

[17] Sloan J, Kesler D, Kumar R, Rahimi A. A numerical optimization-based methodology for application robustification: Transforming applications for error tolerance. In *Proc. DSN*, July 2010, pp.161-170.

[18] Deka B, Birklykke A, Duwe H *et al.* Markov chain algorithms: A template for building future robust low power systems. In *Proc. Asilomar Conf. Signals, Systems and Computers*, Nov. 2013, pp.118-125.

[19] Kesler D, Deka B, Kumar R. A hardware acceleration technique for gradient descent and conjugate gradient. In *Proc. the 9th SASP*, June 2011, pp.94-101.

[20] Sartori J, Kumar R. Branch and data herding: Reducing control and memory divergence for error-tolerant GPU applications. *IEEE Transactions on Multimedia*, 2013, 15(2): 279-290.

[21] Cong J, Gururaj K. Assuring application-level correctness against soft errors. In *Proc. ICCAD*, Nov. 2011, pp.150-157.

[22] Cong J, Ghodrat M A, Gill M *et al.* Architecture support for accelerator-rich CMPs. In *Proc. the 49th DAC*, June 2012, pp.843-849.

[23] Cong J, Ghodrat M A, Gill M. CHARM: A composable heterogeneous accelerator-rich microprocessor. In *Proc. ISLPED*, July 30-August 1, 2012, pp.379-384.
[24] Cong J, Ercegovac M, Huang M *et al.* Energy-efficient computing using adaptive table lookup based on nonvolatile memories. In *Proc. ISLPED*, Sept. 2013, pp.280-285.

Jason Cong received his B.S. degree in computer science from Peking University in 1985, his M.S. and Ph.D. degrees in computer science from the University of Illinois at Urbana-Champaign in 1987 and 1990, respectively. Currently, he is a Chancellor's Professor at the Computer Science Department of University of California, Los Angeles, UCLA, the director of Center for Domain-Specific Computing (CDSC), co-director of UCLA/Peking University Joint Research Institute in Science and Engineering, and co-director of the VLSI CAD Laboratory. He served as the chair the UCLA Computer Science Department from 2005 to 2008. Dr. Cong is also a distinguished visiting professor at Peking University and the director of PKU Center for Energy-Efficient Computing and Applications (CECA). Dr. Cong's research interests include synthesis of VLSI circuits and systems, programmable systems, novel computer architectures, nano-systems, and highly scalable algorithms. He has over 400 publications in these areas, including 10 best paper awards, and the 2011 ACM/IEEE A. Richard Newton Technical Impact Award in Electric Design Automation. He was elected to an IEEE Fellow in 2000 and ACM Fellow in 2008. He is the recipient of the 2010 IEEE Circuits and System (CAS) Society Technical Achievement Award "For seminal contributions to electronic design automation, especially in FPGA synthesis, VLSI interconnect optimization, and physical design automation".

Henry Duwe received his B.S.-AMEP in 2010 from the University of Wisconsin–Madison and his M.S. degree in electrical and computer engineering (ECE) in 2013 from the University of Illinois at Urbana-Champaign. His M.S. work focused on characterizing and effectively exploiting the dynamic behavior of fault tolerance at the instruction level. He is currently a Ph.D. candidate in ECE at the University of Illinois at Urbana-Champaign. His research interests include the design of programmable hardware for unreliable substrates and the design of programmable stochastic accelerators.

Rakesh Kumar is an associate professor in the Electrical and Computer Engineering Department at the University of Illinois at Urbana Champaign. His current research interests are in computer architecture, stochastic and approximate computing, low power and error resilient computer systems, and architectures for inference and machine learning. His research recognitions include several best paper awards and best paper award nominations, ARO Young Investigator Award, Arnold O Beckman Research Award, FAA Creative Research Award, UCSD CSE Best Dissertation Award, and an IBM Ph.D. Fellowship. His teaching recognitions include appearance on UIUC's List of Teachers Ranked as Excellent. Advising recognitions include Engineering Council Outstanding Advisor Award. Rakesh has a B.S. degree from IIT Kharagpur and a Ph.D. degree in computer science (computer engineering) from University of California at San Diego.

Sen Li received his B.S. degree in computer science from Peking University in 2010, and his M.S. degree in computer science from the University of California, Los Angeles in 2014. His research interest includes approximate computing, static/dynamic program analysis, and error-resilient architecture.