Hardware/Software Codesign for Energy Efficiency and Robustness: From Error-Tolerant Computing to Approximate Computing

Abbas Rahimi and Rajesh K. Gupta

1 Introduction

Let us step back and first look at an ideal hardware where the entire software stack can be executed. In reality, however, the hardware underneath of computing is being challenged as CMOS scaling continues to nanometer dimensions [14, 41]. The hardware experiences different sources of variability over time, or across different parts (see Fig. 1a). These variations include: manufacturing process variability that causes static variations in critical dimension, channel length, and threshold voltage of devices [6]; temporal aging/wear out variability that causes slow degradation in devices [22]; and finally, dynamic variability in ambient condition that is caused by fluctuations in operating temperature and supply voltage [8, 25]. The way that designers typically combat with these sources of variability is to consider worst-case design by imposing a large margin in hardware to ensure the correct execution of the software stack. This conservative margin leads to a loss of energy efficiency. Further, the ever-increasing amount of variability [15] limits how far we can drive down the energy per operation (i.e., voltage scaling). This means that we cannot reduce the energy as we used to.

What if we reduce the excessive margin to enable better energy scaling? The direct manifestation of reducing margin is a timing error as shown in Fig. 1b. A timing error means capturing an invalid value to a storage element like a flip-flop or a memory cell, so the result of computation might become wrong. Instead of blindly dealing with variability and its resulting timing errors, we propose to expose them to

A. Rahimi (✉)
ETH Zurich, Zurich, Switzerland
e-mail: abbas@iis.ee.ethz.ch

R. K. Gupta
University of California San Diego, La Jolla, CA, USA
e-mail: gupta@cs.ucsd.edu

Fig. 1 Overdesigned (i.e., worst-case) hardware vs. underdesigned hardware and their interactions with software stack. **(a)** Worst-case design by considering excessive margin leads to loss of energy efficiency. **(b)** Reducing excessive margin results in timing errors

the higher levels in the stack where their side effects can be mitigated [27]. Essentially, we develop an opportunistic software layer that can operate with reduced margins and sense variability in *underdesigned* hardware—instead of overdesigned hardware with positive margins. The software layer accordingly performs adaptation by means of metadata mechanisms that reflect the state of hardware and variability, then the software can perform introspection and adaptation. The main continuations of this chapter lie on the application, software, and architectural layers as illustrated in Fig. 2.

Fig. 2 Main abstraction layers of embedded systems and this chapter’s major (green, solid) and minor (yellow, dashed) cross-layer contributions

1.1 Clockwise Y-Chart: From Positive Margin, to Zero Margin, to Negative Margin

In the following, we discuss about the possible approaches to reduce margin and handle timing errors. The tree possible approaches are conceptualized in a Y-chart shown in Fig. 3.

This first approach is to predict and prevent the errors by keeping a positive margin. Hence, we try to reduce the excessive margin but it is still positive to ensure the error-free execution of software. In this direction, our work spans defining and measuring the notion of error tolerance, from instruction set architecture (ISA) to procedures, and to parallel programs. These measures essentially capture the likelihood of errors and associated cost of error correction at different levels. We first characterize the manifestations of variability in ISA [28] that is the finest granularity to represent a processor functionality. Then, we characterize a sequence of instructions where the timing errors can be eliminated [31]. Going higher in the software stack, we schedule different procedure calls in multi-core architecture [29] and finally a large number of kernels on massively parallel cores [32] such that there are no timing errors. At the boundary of hardware/software, we focus on adaptive compilation methods to reduce the side effects of aging and increase lifetime for massively parallel integrated architectures like GPUs [30].

What is the next approach? The next approach is about detecting and correction errors by reducing the margin to zero (i.e., operating at the edge of errors). Basically, we reduce the margin to zero such that the errors can occur. Hence, we first need to detect the errors by means of circuit sensors [7, 11, 46] and then take actions to correct them. Our focus was instead on reducing the cost of error correction in software. In this direction, we focus on variability-aware runtime environment to cover various embedded parallel workloads in OpenMP including tasks [34, 38], parallel sections, and loops [37].

Finally, the third approach is about accepting errors—i.e., approximate computing—by pushing the margin to negative. This means that the errors and approximations are becoming acceptable as long as the outcomes have a well-

Fig. 3 Taxonomy of error tolerance in a clockwise Y-Chart: from positive margin, to zero margin, to finally negative margin

defined statistical behavior. In this approach, fatal errors can be avoided at the cost of benign approximation that can in fact allow for improving throughput and energy efficiency. Toward this goal, we enable approximate computing in instructions [33, 36, 39, 40], functional units [16–19], runtime execution environments [35], and ultimately hardware description languages [47], and high-level synthesis [23].

By looking at the Y-chart clockwise, we go from preventing errors, to correcting errors, and finally to accepting errors. This move also changes the margin from positive, to zero, and eventually to negative, leading to higher energy efficiency. In the rest of this chapter, we only focus on two methods describing how cooperative hardware/software techniques can improve energy efficiency and robustness in the presence of variability. These two methods, highlighted with bold in the Y-chart, cover examples from approaches with positive margin (i.e., predicting and preventing errors in Sect. 2) and negative margin (i.e., accepting errors in Sect. 3). Interested readers can refer to [42] for reading more about approaches with zero margin (i.e., detecting and correcting errors).

2 Positive Margin Example: Mitigating Aging in GPUs by Adaptive Compilation

In this section, we demonstrate a prime example of software that can respond to hardware variations due to aging. The goal is to reduce the excessive margin due to device aging in a setting where margin is still positive to guarantee correct execution. The idea is to combine hardware sensing circuits and adaptive software (an aging-aware compiler) to manage the workload stress. One major aging mechanism is negative bias temperature instability (NBTI) that adversely affects the reliability of a processing element by introducing new delay-induced faults. However, the effect of these delay variations is not uniformly spread across processing elements within a chip: some are affected more—hence less reliable—than others. We propose an NBTI-aware compiler-directed very long instruction word (VLIW) assignment that uniformly distributes the stress of instructions among available processing elements, with the aim of minimizing aging without any performance penalty [30]. The compiler matches the measured aging degradation with the distribution of instructions to equalize the expected lifetime of each processing element.

The rest of this chapter is organized as follows: Section 2.1 covers an overview of NBTI-induced performance degradation. Section 2.2 describes a GPU architecture and its workload distribution used in this study. Finally, our adaptive compiler is presented in Sect. 2.3.

2.1 NBTI Degradation

Among various aging mechanisms in hardware, the generation of interface traps under NBTI in PMOS transistors has become a critical issue in determining the lifetime of CMOS devices [10]. NBTI manifests itself as an increase in the PMOS transistor voltage threshold (V_{th}) that causes delay-induced failures (see Fig. 4 (left)). NBTI is best captured by the reaction–diffusion model [26]. This model describes NBTI in two stress and recovery phases. NBTI occurs due to the generation of the traps at the Si–SiO₂ interface when the PMOS transistor is negatively biased (i.e., during the stress phase). As a result, V_{th} of the transistor increases which in turn slows down the device. Removing stress from the PMOS transistor can eliminate some of the traps which partially recover the V_{th} shift. This is also known as the recovery phase. The work in [5] derived a long-term cycle-to-cycle model of NBTI. NBTI effects can be significant: its impact on circuit delay is about 15% on a 65 nm technology node and it gets worse in sub-65 nm nodes [4]. Further, NBTI-induced performance degradation is typically non-uniform which is a major concern for many-core GPUs, e.g., with up to 320 five-way VLIW processing elements [1].

Fig. 4 Adaptive compiler to mitigating aging in GPUs: (left) sensing NBTI degradation; (middle) naive kernel execution and its impact on degradation of processing elements; (right) adaptive compiler and healthy kernel execution

2.2 GPU Architecture and Workload

We focus on the evergreen family of AMD GPUs (a.k.a. Radeon HD 5000 series), designed to target not only graphics applications but also general-purpose data-intensive applications. The Radeon HD 5870 GPU compute device consists of 20 compute units (CUs), a global front-end ultra-thread dispatcher, and a crossbar to connect the global memory to the L1-caches. Every CU contains a set of 16 stream cores. Finally, each stream core contains five processing elements (PEs), labeled X, Y, Z, W, and T constituting a VLIW processor to execute machine instructions in a vector-like fashion. The five-way VLIW processor capable of issuing up to five floating point scalar operations from a single VLIW consists primarily of five slots ($slot_X$, $slot_Y$, $slot_Z$, $slot_W$, $slot_T$). Each slot is related to its corresponding PE. Four PEs (X, Y, Z, W) can perform up to four single-precision operations separately and perform two double-precision operations together, while the remaining one (T) has a special function unit for transcendental operations. In each clock cycle, VLIW slots supply a bundle of data-independent instructions to be assigned to the related PEs for simultaneous execution. In an n -way VLIW processor, up to n data-independent instructions, available on n slots, can be assigned to the corresponding PEs and be executed simultaneously. Typically, this is not done in practice because the compiler may fail to find sufficient instruction-level parallelism to generate complete VLIW instructions. On average, if m out of n slots are filled during an execution, we call the achieved packing ratio is m/n . The actual performance of a program running on a VLIW processor largely depends on the packing ratio.

2.2.1 GPU Workload Distribution

Here, we analyze the workload distribution on the Radeon HD GPUs at architecture level, where there are many PEs to carry out computations. As it is mentioned earlier, the NBTI-induced degradation strongly depends on the resource utilization, which depends on the execution characteristics of the workload. Thus, it is essential to analyze how often the PEs are exercised during the execution of the workload. To this end, we first monitor the utilization of various CUs (inter-CU) and then the utilization of PEs within a CU (intra-CU).

To examine the inter-CU workload variation, the total number of executed instructions by each CU is collected during a kernel execution. We observe that the CUs execute almost equal number of instructions, and there is a negligible workload variation among them. We have configured six compute devices with different number of CUs, $\{2, 4, \dots, 64\}$, to finely examine the effect of the workload variation on a variety of GPU architectures.¹ For instance, during DCT kernel execution, the workload variation between CUs ranges from 0% to 0.26% depending on the number of physical CUs on the compute device. Execution of a large number of different kernels confirms that the inter-CU workload variation is less than 3%, when running on the device with 20 CUs (i.e., HD 5870). This nearly uniform inter-CU workload distribution is accomplished by load balancing and uniform resource arbitration algorithms of dispatcher in the GPU architecture.

Next, we examine the workload distribution among the PEs. Figure 4 (middle) shows the percentage of the executed instructions by various PEs during execution of kernels. We only consider four PEs (PE_X , PE_Y , PE_Z , PE_W) which are identical in their functions [1]; they differ only in the vector elements to which they write their result at the end of the VLIW. As shown, the instructions are not uniformly distributed among PEs. For instance, the PE_X executes 40% of ALU instructions, while the PE_W executes only 19% of the instructions. This non-uniform workload variation causes non-uniform aging among PEs. In other words, some PEs are exhausted more than other and thus have shorter lifetime as shown in Fig. 4 (middle). Unfortunately, this non-uniformity happens within all CUs since their workload is highly correlated together. Therefore, no PE throughout the entire compute device is immune from this unbalanced utilization.

The root cause of non-uniform aging among PEs is the frequent and non-uniform execution of VLIW slots. In other words, higher utilization of PE_X implies that $slot_X$ of VLIW is occupied more frequently than the other slots. This substantiates that the compiler does not uniformly assign the independent instructions to various VLIW slots, mainly because the compiler only employs optimization for increasing the packing ratio through finding more parallelism to fully pack the VLIW slots. The VLIW processors are designed to give the compiler tight control over program

¹The latest Radeon HD 5000 series, HD 5970, has 40 CUs featuring 4.3 billion transistors in 40 nm technology.

execution; however, the flexibility afforded by such compilers, for instance, to tune the order of instructions packing, can be used towards reliability improvement.

2.3 Adaptive Aging-Aware Compiler

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 lifetime of all PEs (see Fig. 4 (right)). Here, we briefly describe how these statistics can be obtained from silicon, and how the compiler can predict and thus control the non-uniform aging. The adaptation flow includes four steps: (1) aging sensor readout; (2) kernel disassembly, static code analysis, and calibration of predictions; (3) uniform slot assignment; (4) healthy code generation. We explain them in the following.

The compiler first needs to access the current aging data (ΔV_{th}) of PEs to be able to adapt the code accordingly. The ΔV_{th} is caused by the temporal degradation due to NBTI and/or the intrinsic process variation, thus PEs even during the early life of a chip might have different aging. Employing the compact per-PE NBTI sensors [44, 45] which provide ΔV_{th} measurement with 3σ accuracy of 1.23 mV for a wide range of temperature enables large scale data-collection across all PEs. The performance degradation of every PE can be reliably reported by a per-PE NBTI sensor, thanks to the small overhead of these sensors. The sensors support digital frequency outputs that are accessed through memory-mapped I/O by the compiler in arbitrary epochs of the post-silicon measurement. After sensor readouts, the compiler estimates the degradation of PEs using the NBTI models. In addition to the current aging data, the compiler needs to have an estimate regarding the impact of future workload stress on the various PEs. Hence, a just-in-time disassembler disassembles a naive kernel binary to a device-dependent assembly code in which the assignment of instructions to the various slots (corresponding PEs) are explicitly defined and are thus observable by the compiler. Then, a static code analysis technique is applied that estimates the percentage of instructions that will be carried out on every PE in a static sense. It extracts the future stress profile, and thus the utilization of various PEs using the device-dependent assembly code. If the predicted stress of a PE is overestimated or underestimated, mainly due to the static analysis of the branch conditions of the kernel’s assembly code, a linear calibration module fits the predicted stress to the observed stress, in the next adaptation period.

Thus far, we have described how the compiler evaluates the current performance degradation (aging) of every PE and their future performance degradation due to

the naive kernel execution. Then, the compiler uses this information to perform code transformations with the goal of improving reliability, without any penalty in the throughput of code execution (maintaining the same parallelism). To minimize stress, the compiler sorts the predicted performance degradation of the slots increasingly and the aging of the slots decreasingly and then applies a permutation to assign fewer/more instructions to higher/lower stressed slots. This algorithm is applied for every adaptation period. As a result of the slot reallocation, the minimum/maximum number of instructions is assigned to the highest/lowest stressed slot for the future kernel execution. This reduces ΔV_{th} shifts by 34%, thus uniforming the lifetime of PEs and allowing for reducing the positive margin as shown in Fig. 4 (right).

Execution of all examined kernels shows that the average packing ratio is 0.3 which means there is a large fraction of empty slots in which PEs can be relaxed during kernels execution. This low packing ratio is mainly due to the limitation of instruction-level parallelism. The proposed adaptive compilation approach superposes on top of all optimization performed by a naive compiler and does not incur any performance penalty since it only reallocates the VLIW slots (slips the scheduled instructions from one slot to another) within the same scheduling and order determined by the naive compiler. In other words, our compiler guarantees the iso-throughput execution of the healthy kernel. It also runs fully in parallel with GPU on a host CPU, thus there will be no penalty for GPU kernel execution if dynamic compilation of one kernel can be overlapped with the execution of another kernel. You can refer to [49] for further details.

3 Negative Margin Example: Enabling Approximate Computing in FPGA Workflow

Modern applications including graphics, multimedia, web search, and data analytics exhibit significant degrees of tolerance to imprecise computation. This amenability to approximation provides an opportunity to reduce the excessive margin, namely to negative, by accepting errors that trade the quality of the results for higher efficiency. Approximate computing is a promising avenue that leverages such tolerance of applications to errors [12, 13, 20, 21, 24, 47]. However, there is a lack of techniques that exploits this opportunity in FPGAs.

In [23], we aim to bridge the gap between approximation and the FPGA acceleration through an automated design workflow. Exploiting this opportunity is particularly important for FPGA accelerators that are inherently subject to many resource constraints. To better utilize the FPGA resources, we devise an automated design workflow for FPGAs [23] that leverages imprecise computation to increase data-level parallelism and achieve higher computational throughput. The core of our workflow is a source-to-source compiler that takes in an input kernel and applies a novel optimization technique that selectively reduces the precision of the kernel data and operations. By selectively reducing the precision of the data and

operation (analogous to setting margin to negative), the required area to synthesize the kernel on the FPGA decreases allowing to integrate a larger number of operations and *parallel* kernels in the fixed area of the FPGA (i.e., improving energy efficiency per unit of area). The larger number of integrated kernels provides more hardware context to better exploit data-level parallelism in the target applications. To effectively explore the possible design space of approximate kernels, we exploit a genetic algorithm to find a subset of safe-to-approximate operations and data elements and then tune their precision levels until the desired output quality is achieved. Our method exploits a fully software technique and does not require any changes to the underlying FPGA hardware. We evaluate it on a diverse set of data-intensive OpenCL benchmarks from the AMD accelerated parallel processing (APP) SDK v2.9 [3]. We later describe OpenCL execution model and its mapping on FPGA in Section 3.1. The synthesis result on a Stratix V Altera FPGA shows that our approximation workflow yields $1.4\times\text{--}3.0\times$ higher throughput with less than 1% quality loss (see Sect. 3.2).

3.1 OpenCL Execution Model and Mapping on FPGAs

Altera and Xilinx recently offer high-level acceleration frameworks for OpenCL [2, 43], hence we target acceleration of data-intensive computational OpenCL applications. The challenge is however devising a workflow that can be plugged into the existing toolsets and can automatically identify the opportunities for approximation while keeping the quality loss reasonably low. OpenCL is a platform-independent framework for writing programs that execute across a heterogeneous system consisting of multiple compute devices including CPUs or accelerators such as GPUs, DSPs, and FPGAs. OpenCL uses a subset of ISO C99 with added extensions for supporting data and task-based parallel programming models. The programming model in OpenCL comprises of one or more device kernel codes in tandem with the host code. The host code typically runs on a CPU and launches kernels on other compute devices like the GPUs, DSPs, and/or FPGAs through API calls. The instance of an OpenCL kernel is called a work-item. These kernels execute on compute devices that are a set of compute units (CUs), each comprising of multiple PEs having ALUs. The work-items execute on a single PE and exercise the ALU.

The Altera OpenCL SDK [2] allows programmers to use high-level OpenCL kernels, written for GPUs, to generate an FPGA design with higher performance per Watt [9]. In this work, an OpenCL kernel is first compiled and then synthesized as a special dedicated hardware for mapping on an FPGA. FPGAs can further improve the performance benefits by creating multiple copies of the kernel pipelines (synthesized version of an OpenCL kernel). For instance, this replication process can make n copies of the kernel pipeline. As the kernel pipelines can be executed independently from one another, the performance would scale linearly with the number of copies created owing to the data-level parallelism model supported by OpenCL.

In the following, we describe how our method can reduce the amount of resources for a kernel pipeline to save area and exploit remaining area resources to boost performance by replication. Our method systematically reduces the precision of data and operations in OpenCL kernels to shrink the resources used per kernel pipeline by transforming complex kernels to simple kernels that produce *approximate* results.

3.2 Source-to-Source Compiler

We provide a source-to-source compiler to generate approximate kernels from OpenCL kernels with exact specification as shown in Fig. 5. This transformation automatically detects and simplifies parts of the kernel code that can be executed with reduced precision while preserving the desired quality-of-result. To achieve this goal, our compiler takes in as inputs, an *exact* OpenCL kernel, a set of input test cases, and a metric for measuring the quality-of-result target. The compiler investigates the exact kernel code and detects data elements, i.e., OpenCL kernel variables, that provide possible opportunities for increased performance in exchange for accuracy. It then automatically generates a population of approximate kernels by

Fig. 5 FPGA approximation design workflow

Fig. 6 Example of mapping exact and approximate kernels of Sobel filter on FPGA

means of a genetic algorithm. It can select the approximate kernels that produce acceptable results with the help of GPU profiling. These approximate kernels provide improved performance benefits by reducing the area when implemented on the FPGAs. The compiler finally outputs an optimized approximate kernel with the least area whose output quality satisfies the quality-of-result target.

The compiler uses the precision of the operations and data to tune performance as a trade-off against precision. The transformation investigates a set of kernels where in each version, some of these potential variables are replaced with a less accurate variable. To avoid a huge design space exploration, we devise an algorithm that first detects those variables that are amenable to approximation and then applies a genetic algorithm to approximate the kernel. We discuss the details of our algorithm in [23].

Figure 6 shows an example of Sobel filter kernel that is optimized by our compiler. Naive mapping of exact Sobel kernel allows us to map five instances of the kernel on the FPGA. However, by using the approximate version of kernel, we can map 13 instances of kernel on the same FPGA roughly improving throughput by 2 $\times$, thanks to the data-level parallel execution of the kernel, while meeting the quality constraint. We set the quality loss target to a maximum of 0.7% for image processing applications (which is equivalent to PSNR of a minimum 30 dB) and 1% for other applications which is conservatively aligned with other work on quality trade-offs [20, 21, 24, 48]. Benchmarking five kernels from OpenCL AMD APP SDK v2.9 shows that our compiler integrates a larger number of parallel kernels on the same FPGA fabric that leads to 1.4 $\times$ –3.0 $\times$ higher throughput on

a modern Altera FPGA with less than 1% loss of quality. This is a prime example of accepting *disciplined* errors, in the context of approximate computing, for improved throughput. The approach can be generalized to any controllable error caused by various sources. Further details are provided in [23].

4 Conclusion

Microelectronic variability is a phenomenon at the intersection of microelectronic scaling, semiconductor manufacturing, and how electronic systems are designed and deployed. To address this variability, designers resort to margins. We show how such excessive margins can be reduced, and their effects can be mitigated, by a synergy between hardware and software leading to efficient and robust microelectronic circuit and systems.

We first explore approaches to reduce the margin and enable better than worst-case design while avoiding the errors. We demonstrate its effectiveness on GPUs where the effect of variations is not uniformly spread across over thousands processing elements. Hence, we devise an adaptive compiler that equalizes the expected lifetime of each processing element by regenerating an aging-aware healthy kernel. Such new kernel guides its workload distribution that does matter for the aging, hence effectively responding to the specific health state of GPUs.

Next, we focus on approaches that significantly reduce the margins by accepting errors and exploiting approximation opportunities in computation. We explore purely software transformation methods to unleash untapped capabilities of the contemporary fabrics for exploiting approximate computing. Exploiting this opportunity is particularly important for FPGA accelerators that are inherently subject to many resource constraints. To better utilize the FPGA resources, we develop an automated design workflow for FPGA accelerators that leverages approximate computation to increase data-level parallelism and achieve higher computational throughput.

References

1. Advanced Micro Devices, Inc: AMD Evergreen Family Instruction Set Architecture
2. Altera sdk for openc1: <http://www.altera.com/products/software/openc1/openc1-index.html>
3. Amd app sdk v2.9: <http://developer.amd.com/tools-and-sdks/openc1-zone/amd-accelerated-parallel-processing-app-sdk/>
4. Bernstein, K., Frank, D., Gattiker, A., Haensch, W., Ji, B., Nassif, S., Nowak, E., Pearson, D., Rohrer, N.: High-performance cmos variability in the 65-nm regime and beyond. *IBM J. Res. Dev.* **50**(4.5), 433–449 (2006). <https://doi.org/10.1147/rd.504.0433>
5. Bhardwaj, S., Wang, W., Vattikonda, R., Cao, Y., Vrudhula, S.: Predictive modeling of the nbt1 effect for reliable design. In: Custom Integrated Circuits Conference, 2006, CICC '06, pp. 189–192. IEEE (2006). <https://doi.org/10.1109/CICC.2006.320885>

6. Bowman, K., Duvall, S., Meindl, J.: Impact of die-to-die and within-die parameter fluctuations on the maximum clock frequency distribution. In: Solid-State Circuits Conference, 2001. Digest of Technical Papers, ISSCC, 2001, pp. 278–279. IEEE International (2001). <https://doi.org/10.1109/ISSCC.2001.912637>
7. Bowman, K., Tschanz, J., Kim, N.S., Lee, J., Wilkerson, C., Lu, S., Karnik, T., De, V.: Energy-efficient and metastability-immune resilient circuits for dynamic variation tolerance. *IEEE J. Solid State Circuits* **44**(1), 49–63 (2009). <https://doi.org/10.1109/JSSC.2008.2007148>
8. Bowman, K., Tokunaga, C., Tschanz, J., Raychowdhury, A., Khellah, M., Geuskens, B., Lu, S.L., Aseron, P., Karnik, T., De, V.: Dynamic variation monitor for measuring the impact of voltage droops on microprocessor clock frequency. In: Custom Integrated Circuits Conference (CICC), 2010, pp. 1–4. IEEE (2010). <https://doi.org/10.1109/CICC.2010.5617415>
9. Chen, D., Singh, D.: Invited paper: Using opencl to evaluate the efficiency of cpus, gpus and fpgas for information filtering. In: 22nd International Conference on Field Programmable Logic and Applications (FPL), 2012, pp. 5–12. <https://doi.org/10.1109/FPL.2012.6339171>
10. Chen, G., Li, M.F., Ang, C., Zheng, J., Kwong, D.L.: Dynamic nbti of p-mos transistors and its impact on mosfet scaling. *IEEE Electron Device Lett.* **23**(12), 734–736 (2002). <https://doi.org/10.1109/LED.2002.805750>
11. Drake, A., Senger, R., Deogun, H., Carpenter, G., Ghiasi, S., Nguyen, T., James, N., Floyd, M., Pokala, V.: A distributed critical-path timing monitor for a 65nm high-performance microprocessor. In: Solid-State Circuits Conference, 2007, ISSCC 2007. Digest of Technical Papers, pp. 398–399. IEEE International (2007). <https://doi.org/10.1109/ISSCC.2007.373462>
12. Esmailzadeh, H., Sampson, A., Ceze, L., Burger, D.: Architecture support for disciplined approximate programming. In: Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS XVII, pp. 301–312. ACM, New York, NY, USA (2012). <https://doi.org/10.1145/2150976.2151008>. <http://doi.acm.org/10.1145/2150976.2151008>
13. Esmailzadeh, H., Sampson, A., Ceze, L., Burger, D.: Neural acceleration for general-purpose approximate programs. In: Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO-45, pp. 449–460. IEEE Computer Society, Washington, DC, USA (2012). <https://doi.org/10.1109/MICRO.2012.48>
14. Ghosh, S., Roy, K.: Parameter variation tolerance and error resiliency: New design paradigm for the nanoscale era. *Proc. IEEE* **98**(10), 1718–1751 (2010). <https://doi.org/10.1109/JPROC.2010.2057230>
15. Gupta, P., Agarwal, Y., Dolecek, L., Dutt, N., Gupta, R.K., Kumar, R., Mitra, S., Nicolau, A., Rosing, T.S., Srivastava, M.B., Swanson, S., Sylvester, D.: Underdesigned and opportunistic computing in presence of hardware variability. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **32**(1), 8–23 (2013). <https://doi.org/10.1109/TCAD.2012.2223467>
16. Jiao, X., Rahimi, A., Narayanaswamy, B., Fatemi, H., de Gyvez, J.P., Gupta, R.K.: Supervised learning based model for predicting variability-induced timing errors. In: 2015 IEEE 13th International New Circuits and Systems Conference (NEWCAS), pp. 1–4 (2015). <https://doi.org/10.1109/NEWCAS.2015.7182029>
17. Jiao, X., Jiang, Y., Rahimi, A., Gupta, R.K.: Wild: A workload-based learning model to predict dynamic delay of functional units. In: 2016 IEEE 34th International Conference on Computer Design (ICCD), pp. 185–192 (2016). <https://doi.org/10.1109/ICCD.2016.7753279>
18. Jiao, X., Jiang, Y., Rahimi, A., Gupta, R.K.: Slot: A supervised learning model to predict dynamic timing errors of functional units. In: Design, Automation Test in Europe Conference Exhibition (DATE), 2012 (2017)
19. Jiao, X., Rahimi, A., Jiang, Y., Wang, J., Fatemi, H., de Gyvez, J.P., Gupta, R.K.: Clim: A cross-level workload-aware timing error prediction model for functional units. *IEEE Trans. Comput.* **67**(6), 771–783 (2018). <https://doi.org/10.1109/TC.2017.2783333>
20. Kahng, A., Kang, S.: Accuracy-configurable adder for approximate arithmetic designs. In: Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE, pp. 820–825 (2012)

21. Kulkarni, P., Gupta, P., Ercegovic, M.: Trading accuracy for power with an underdesigned multiplier architecture. In: 2011 24th International Conference on VLSI Design (VLSI Design), pp. 346–351 (2011). <https://doi.org/10.1109/VLSID.2011.51>
22. Li, X., Qin, J., Bernstein, J.: Compact modeling of mosfet wearout mechanisms for circuit-reliability simulation. *IEEE Trans. Device Mater. Reliab.* **8**(1), 98–121 (2008). <https://doi.org/10.1109/TDMR.2008.915629>
23. Lotfi, A., Rahimi, A., Yazdanbakhsh, A., Esmailzadeh, H., Gupta, R.K.: Grater: An approximation workflow for exploiting data-level parallelism in fpga acceleration. In: 2016 Design, Automation Test in Europe Conference Exhibition (DATE), pp. 1279–1284 (2016)
24. Moreau, T., Wyse, M., Nelson, J., Sampson, A., Esmailzadeh, H., Ceze, L., Oskin, M.: Snnap: Approximate computing on programmable socs via neural acceleration. In: 2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA), pp. 603–614 (2015). <https://doi.org/10.1109/HPCA.2015.7056066>
25. Murali, S., Mutapic, A., Atienza, D., Gupta, R., Boyd, S., Benini, L., De Micheli, G.: Temperature control of high-performance multi-core platforms using convex optimization. In: Proceedings of the Conference on Design, Automation and Test in Europe, DATE '08, pp. 110–115. ACM, New York, NY, USA (2008). <https://doi.org/10.1145/1403375.1403405>. <http://doi.acm.org/10.1145/1403375.1403405>
26. Ogawa, S., Shiono, N.: Generalized diffusion-reaction model for the low-field charge-buildup instability at the si-sio2 interface. *Phys. Rev.* **51**(7), 4218–4230 (1995)
27. Rahimi, A.: From variability-tolerance to approximate computing in parallel computing architectures. Ph.D. thesis, University of California San Diego, <https://escholarship.org/uc/item/1c68g008> (2015)
28. Rahimi, A., Benini, L., Gupta, R.K.: Analysis of instruction-level vulnerability to dynamic voltage and temperature variations. In: Design, Automation Test in Europe Conference Exhibition (DATE), 2012, pp. 1102–1105 (2012). <https://doi.org/10.1109/DATE.2012.6176659>
29. Rahimi, A., Benini, L., Gupta, R.K.: Procedure hopping: A low overhead solution to mitigate variability in shared-l1 processor clusters. In: Proceedings of the 2012 ACM/IEEE International Symposium on Low Power Electronics and Design, ISLPED '12, pp. 415–420. ACM, New York, NY, USA (2012). <https://doi.org/10.1145/2333660.2333754>. <http://doi.acm.org/10.1145/2333660.2333754>
30. Rahimi, A., Benini, L., Gupta, R.K.: Aging-aware compiler-directed vliw assignment for gpgpu architectures. In: Proceedings of the 50th Annual Design Automation Conference, DAC '13, pp. 16:1–16:6. ACM, New York, NY, USA (2013). <https://doi.org/10.1145/2463209.2488754>. <http://doi.acm.org/10.1145/2463209.2488754>
31. Rahimi, A., Benini, L., Gupta, R.K.: Application-adaptive guardbanding to mitigate static and dynamic variability. *IEEE Trans. Comput.* (2013). <https://doi.org/10.1109/TC.2013.72>
32. Rahimi, A., Benini, L., Gupta, R.K.: Hierarchically focused guardbanding: An adaptive approach to mitigate pvt variations and aging. In: Design, Automation Test in Europe Conference Exhibition (DATE), 2013, pp. 1695–1700 (2013). <https://doi.org/10.7873/DATE.2013.342>
33. Rahimi, A., Benini, L., Gupta, R.K.: Spatial memoization: Concurrent instruction reuse to correct timing errors in simd architectures. *IEEE Trans. Circuits Syst. II Express Briefs* **60**(12), 847–851 (2013). <https://doi.org/10.1109/TCSII.2013.2281934>
34. Rahimi, A., Marongiu, A., Burzio, P., Gupta, R.K., Benini, L.: Variation-tolerant openmp tasking on tightly-coupled processor clusters. In: Design, Automation Test in Europe Conference Exhibition (DATE), 2013, pp. 541–546 (2013). <https://doi.org/10.7873/DATE.2013.121>
35. Rahimi, A., Marongiu, A., Gupta, R.K., Benini, L.: A variability-aware openmp environment for efficient execution of accuracy-configurable computation on shared-fpu processor clusters. In: 2013 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), pp. 1–10 (2013). <https://doi.org/10.1109/CODES-ISSS.2013.6659022>

36. Rahimi, A., Benini, L., Gupta, R.K.: Temporal memoization for energy-efficient timing error recovery in gpgpus. In: Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014, pp. 1–6 (2014). <https://doi.org/10.7873/DATE2014.113>
37. Rahimi, A., Cesarini, D., Marongiu, A., Gupta, R.K., Benini, L.: Improving resilience to timing errors by exposing variability effects to software in tightly-coupled processor clusters. *IEEE J. Emerging Sel. Top. Circuits Syst.* **4**(2), 216–229 (2014). <https://doi.org/10.1109/JETCAS.2014.2315883>
38. Rahimi, A., Cesarini, D., Marongiu, A., Gupta, R.K., Benini, L.: Task scheduling strategies to mitigate hardware variability in embedded shared memory clusters. In: Proceedings of the 52Nd Annual Design Automation Conference, DAC '15, pp. 152:1–152:6. ACM, New York, NY, USA (2015). <https://doi.org/10.1145/2744769.2744915>. <http://doi.acm.org/10.1145/2744769.2744915>
39. Rahimi, A., Ghofrani, A., Cheng, K.T., Benini, L., Gupta, R.K.: Approximate associative memristive memory for energy-efficient gpus. In: Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition, DATE '15, pp. 1497–1502 (2015). <http://dl.acm.org/citation.cfm?id=2757012.2757158>
40. Rahimi, A., Benini, L., Gupta, R.K.: Circa-gpus: Increasing instruction reuse through inexact computing in gp-gpus. *IEEE Des. Test* **33**(6), 85–92 (2016). <https://doi.org/10.1109/MDAT.2015.2497334>
41. Rahimi, A., Benini, L., Gupta, R.K.: Variability mitigation in nanometer cmos integrated systems: A survey of techniques from circuits to software. *Proc. IEEE* **104**(7), 1410–1448 (2016). <https://doi.org/10.1109/JPROC.2016.2518864>
42. Rahimi, A., Benini, L., Gupta, R.K.: From Variability Tolerance to Approximate Computing in Parallel Integrated Architectures and Accelerators. Springer International Publishing (2017)
43. SDAccel: <http://www.xilinx.com/products/design-tools/sdx/sdaccel.html> (2015)
44. Singh, P., Karl, E., Sylvester, D., Blaauw, D.: Dynamic nbtI management using a 45 nm multi-degradation sensor. *IEEE Trans. Circuits Syst. I Regular Papers* **58**(9), 2026–2037 (2011). <https://doi.org/10.1109/TCSI.2011.2163894>
45. Singh, P., Karl, E., Blaauw, D., Sylvester, D.: Compact degradation sensors for monitoring nbtI and oxide degradation. *IEEE Trans. Very Large Scale Integr. VLSI Syst.* **20**(9), 1645–1655 (2012). <https://doi.org/10.1109/TVLSI.2011.2161784>
46. Tschanz, J., Bowman, K., Walstra, S., Agostinelli, M., Karnik, T., De, V.: Tunable replica circuits and adaptive voltage-frequency techniques for dynamic voltage, temperature, and aging variation tolerance. In: 2009 Symposium on VLSI Circuits, pp. 112–113 (2009)
47. Yazdanbakhsh, A., Mahajan, D., Thwaites, B., Park, J., Nagendrakumar, A., Sethuraman, S., Ramkrishnan, K., Ravindran, N., Jariwala, R., Rahimi, A., Esmaeilzadeh, H., Bazargan, K.: Axilog: Language support for approximate hardware design. In: 2015 Design, Automation Test in Europe Conference Exhibition (DATE), pp. 812–817 (2015). <https://doi.org/10.7873/DATE.2015.0513>
48. Yazdanbakhsh, A., Park, J., Sharma, H., Lotfi-Kamran, P., Esmaeilzadeh, H.: Neural acceleration for gpu throughput processors. In: Proceedings of the 48th International Symposium on Microarchitecture, MICRO-48, pp. 482–493. ACM, New York, NY, USA (2015). <https://doi.org/10.1145/2830772.2830810>. <http://doi.acm.org/10.1145/2830772.2830810>
49. Yuan, F., Xu, Q.: Intimefix: A low-cost and scalable technique for in-situ timing error masking in logic circuits. In: Design Automation Conference (DAC), 2013 50th ACM / EDAC / IEEE, pp. 1–6 (2013)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.