

From Variability Tolerance to Approximate Computing in Parallel Integrated Architectures and Accelerators

Abbas Rahimi · Luca Benini
Rajesh K. Gupta

From Variability Tolerance to Approximate Computing in Parallel Integrated Architectures and Accelerators

Abbas Rahimi
Department of Electrical Engineering
and Computer Sciences
University of California Berkeley
Berkeley, CA
USA

Rajesh K. Gupta
Department of Computer Science
and Engineering
University of California, San Diego
La Jolla, CA
USA

Luca Benini
Integrated Systems Laboratory
ETH Zurich
Zürich
Switzerland

ISBN 978-3-319-53767-2 ISBN 978-3-319-53768-9 (eBook)
DOI 10.1007/978-3-319-53768-9

Library of Congress Control Number: 2017932004

© Springer International Publishing AG 2017

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, express or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Printed on acid-free paper

This Springer imprint is published by Springer Nature
The registered company is Springer International Publishing AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

*To my wife with everlasting love and
gratitude*

–Abbas Rahimi

Foreword

There is no question that computing has dramatically changed society, and has furthered humanity in ways that were hard to foresee at its onset. Many factors have contributed to its unfettered success including the adoption of Boolean logic and algorithmic thinking, the invention of the instruction set machine, and the advent of the semiconductor technology. The latter offered us a semi-perfect switch device and effective ways of storing data. All these factors have led to an amazing run of almost 7 decades. Over time, the quest for ever higher performance in the presence of power and energy limitations have forced us to make major changes to how the processors were architected and operated—such as the introduction of concurrency, the adoption of co-processors and accelerators, or the adoption of ever more complex memory hierarchies. However, in essence the fundamentals remained unchanged

For a number of reasons, this model is at the verge of undergoing some major changes and challenges. On the one hand, the scaling model of semiconductors—commonly known as Moore’s law—is running out of steam, hence depriving us from a convenient means in improving computational performance, density and efficiency. On the other hand, the nature of computation itself is changing with data rather than algorithm taking primacy. Both these trends force us to reflect on some of the foundational concepts that have driven computation for such a long period. In an abundance of data, statistical distributions become more relevant than deterministic answers. Many perceptual tasks related to human-world interaction fall under the same class. Learning-based programming approaches are gaining rapid interest and influence. Simultaneously the lack of “the perfect switch”, as well as the high-variability of nanoscale devices operating under high energy-efficiency (that is, low-voltage) makes deterministic computing an extremely expensive if at all possible undertaking.

All of this has made researchers explore novel computational models that are “approximate” in nature. This means that errors and approximations are becoming acceptable as long as the outcomes have a well-defined statistical behavior. A number of approaches have been identified and are being actively pursued under

different headers such as approximate computing, statistical computing and stochastic processing. In this book, the authors investigate how errors caused by variation (especially those caused by timing) can be exposed to the software layers, and how they can be mitigated using a range of techniques and methods to reduce their impact. The document provides a clear insight of what is possible through pure software intervention.

This book is at the forefront of what is to come at the frontiers in the new age of computation. As such, I heartily recommend it as a great intellectual effort and a superb read.

Jan M. Rabaey

Preface

Variation in performance and power across manufactured parts and their operating conditions is an accepted reality in modern microelectronic manufacturing processes with geometries in nanometer scales. This book views such variations both as a challenge as well an opportunity to rearchitect the hardware/software interface that provides more resilient system architectures. We start with an examination of how variability manifests itself across various levels of microelectronic systems. We examine various mechanisms designers use, and can use, to combat negative effects of variability.

This book attempts a comprehensive look at the entire software/hardware stack and system architecture in order to devise effective strategies to address microelectronic variability. First, we review the key concepts on *timing errors* caused by various variability sources. We use a two-pronged strategy to mitigate such errors by jointly exposing hardware variations to the software and by exploiting flexibility made possible by parallel processing. We consider methods to predict and prevent, detect and correct, and finally conditions under which such errors can be accepted. For each of these methods, our work spans defining and measuring the notion of error tolerance at various levels, from instructions to procedures to parallel programs. These measures essentially capture the likelihood of errors and associated cost of error correction at different levels. The result is a design platform that enables us to combine these methods that enable detection and correction of erroneous results within a defined criterion for acceptable errors using a notion of memoization across the hardware/software interface. Pursuing this strategy, we develop a set of software techniques and microarchitecture optimizations for improving cost and scale of these methods in massively parallel computing units, such as general-purpose graphics processing units (GP-GPUs), clustered many-core architectures, and field-programmable gate array (FPGA) accelerators.

Our results show that parallel architectures and use of parallelism in general provides the best means to combat and exploit variability. Using such programmable parallel accelerator architectures, we show how system designers can

coordinate propagation of error information and its effects along with new techniques for memoization and memristive associative memory. This book naturally leads to use of these techniques into emerging area of approximate computing, and how these can be used in building resilient and efficient computing systems.

Berkeley, USA
Zürich, Switzerland
San Diego, USA
January 2017

Abbas Rahimi
Luca Benini
Rajesh K. Gupta

Contents

1	Introduction	1
1.1	Sources of Variability	1
1.2	Delay Variation	2
1.3	Book Organization	4
	References	6
 Part I Predicting and Preventing Errors		
2	Instruction-Level Tolerance	11
2.1	Introduction	11
2.2	Effect of Operating Conditions	12
2.3	Delay Variation Among Pipeline Stages	13
2.4	Instruction Characterization Methodology and Experimental Results	15
2.4.1	Gate-Level Simulation	15
2.4.2	Instruction-Level Delay Variability	16
2.4.3	Less Intrusive Variation-Tolerant Technique	17
2.4.4	Power Variability	18
2.5	Chapter Summary	19
	References	19
3	Sequence-Level Tolerance	21
3.1	Introduction	21
3.2	PVT Variations	22
3.2.1	Conventional Static Timing Analysis	24
3.2.2	Variation-Aware Statistical STA	26
3.3	Error-Tolerant Applications	27
3.3.1	Analysis of Adaptive Guardbanding for Probabilistic Applications	28

3.4	Error-Intolerant Applications.	30
3.4.1	Sequence-Level Vulnerability (SLV).	30
3.4.2	SLV Characterization	31
3.5	Adaptive Guardbanding	35
3.6	Experimental Results	37
3.6.1	Effectiveness of Adaptive Guardbanding	39
3.6.2	Overhead of Adaptive Guardbanding.	44
3.7	Chapter Summary.	44
	References.	45
4	Procedure-Level Tolerance.	47
4.1	Introduction	47
4.2	Variation-Tolerant Processor Clusters Architecture.	48
4.2.1	Variation-Aware VDD-Hopping	49
4.3	Procedure Hopping for Dynamic IR-Drop	51
4.3.1	Supporting Intra-cluster Procedure Hopping	51
4.4	Characterization of PLV to Dynamic Operating Conditions	54
4.5	Experimental Results	55
4.5.1	Cost of Procedure Hopping	57
4.6	Chapter Summary.	59
	References.	59
5	Kernel-Level Tolerance.	61
5.1	Introduction	61
5.2	Device-Level NBTI Model.	62
5.3	GP-GPU Architecture.	64
5.3.1	GP-GPU Workload Distribution	64
5.4	Aging-Aware Compilation	66
5.4.1	Observability: Aging Sensors	67
5.4.2	Prediction: Wearout Estimation Module	68
5.4.3	Controllability: Uniform Slot Assignment	68
5.5	Experimental Results	70
5.6	Chapter Summary.	73
	References.	73
6	Hierarchically Focused Guardbanding	75
6.1	Introduction	75
6.2	Timing Error Model for PVT.	76
6.2.1	Analysis Flow for Timing Error Extraction.	76
6.2.2	Parametric Model Fitting.	78
6.2.3	TER Classification.	80
6.2.4	Robustness of Classification	81
6.3	Runtime Hierarchically Focused Guardbanding	81
6.3.1	Observability	83
6.3.2	Controllability	84

6.4	A Case Study of HFG on GPUs.	85
6.5	Chapter Summary.	86
	References.	87

Part II Detecting and Correcting Errors

7	Work-Unit Tolerance	91
7.1	Introduction	91
7.2	Architectural Support for VOMP	94
7.3	Work-Unit Vulnerability and VOMP Work-Sharing	95
7.3.1	Intra- and Inter-corner WUV	98
7.3.2	Online WUV Characterization.	103
7.4	VOMP Schedulers	105
7.4.1	Variation-Aware Task Scheduling (VATS)	105
7.4.2	Variation-Aware Section Scheduling (VASS)	108
7.5	Experimental Results	109
7.5.1	Framework Setup	109
7.5.2	VOMP Results for Tasking	110
7.5.3	VOMP Results for Sections.	112
7.6	Chapter Summary.	113
	References.	114
8	Memristive-Based Associative Memory for Error Recovery	117
8.1	Introduction	117
8.2	Energy-Efficient GP-GPUs	119
8.2.1	Associative Memristive-Based Computing.	120
8.3	Collaborative Compilation	122
8.3.1	FPU Memristive-Based Computing.	124
8.4	Experimental Results	125
8.4.1	FPUs with AMM Modules	125
8.4.2	Energy Saving.	126
8.5	Chapter Summary.	129
	References.	129

Part III Accepting Errors

9	Accuracy-Configurable OpenMP	133
9.1	Introduction	133
9.2	Controlled Approximation	135
9.3	Accuracy-Configurable OpenMP Environment	136
9.3.1	Accuracy-Configurable FPUs.	136
9.3.2	OpenMP Compiler Extension for Approximation	137
9.3.3	Runtime Support	138
9.3.4	Application-Driven Hardware FPU Synthesis and Optimization.	139

9.4	Experimental Results	141
9.4.1	Error-Tolerant Applications	142
9.4.2	Error-Intolerant Applications	146
9.5	Chapter Summary	147
	References.	148
10	An Approximation Workflow for Exploiting Data-Level Parallelism in FPGA Acceleration	151
10.1	Introduction	151
10.2	OpenCL Execution Model	153
10.2.1	Mapping OpenCL Programs on FPGAs	153
10.3	GRATER: Approximation Design Workflow	154
10.3.1	Analysis and Pruning	155
10.3.2	Genetic-Based Approximation Algorithm	156
10.4	Experimental Results	159
10.4.1	Experimental Setup	159
10.4.2	Area Savings with Approximate Kernels.	160
10.4.3	Speedup.	160
10.5	Chapter Summary	163
	References.	163
11	Memristive-Based Associative Memory for Approximate Computational Reuse	165
11.1	Introduction	165
11.2	GPU Architecture Using A ² M ² Module.	167
11.2.1	Southern Islands Architecture	167
11.2.2	Approximate Associative Memristive Memory Module	168
11.3	Framework to Support A ² M ²	171
11.3.1	Execution Flow	171
11.3.2	Design Space for A ² M ²	173
11.4	Experimental Results	175
11.4.1	Experimental Setup	175
11.4.2	Energy Saving with Corresponding PSNR	177
11.5	Chapter Summary	178
	References.	179
12	Spatial and Temporal Memoization	181
12.1	Introduction	182
12.2	Spatial Memoization (Concurrent Instruction Reuse)	183
12.2.1	Single Strong Multiple Weak (SSMW) Architecture.	184
12.2.2	Experimental Results.	186
12.3	Temporal Memoization (Temporal Instruction Reuse)	188

- 12.3.1 Temporal Memoization for Error Recovery 188
 - 12.3.2 Experimental Results 189
 - 12.4 Chapter Summary 189
 - References 190
- 13 Outlook 191**
 - 13.1 Domain-Specific Resiliency 191
 - 13.1.1 Software 191
 - 13.1.2 Architecture 192
 - 13.1.3 Circuit 192
 - 13.2 Non-Von Neumann Massively Parallel Architectures 193
- Index 195**