KU LEUVEN

ARENBERG DOCTORAL SCHOOL
Faculty of Engineering Science

Workload Dependent Mitigation Approaches for Performance Variability:

Ensuring Timing Guarantees of Integrated Circuits

Ji-Yung Lin

Supervisors:
Prof. Francky Catthoor
Dr. Pieter Weckx
(imec)

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Engineering Science (PhD): Electrical Engineering

March 2024

Workload Dependent Mitigation Approaches for Performance Variability:

Ensuring Timing Guarantees of Integrated Circuits

Ji-Yung LIN

Examination committee:
Prof. Adhemar Bultheel, chair
Prof. Francky Catthoor, supervisor
Dr. Pieter Weckx, supervisor
(imec)
Prof. Guido Groeseneken
Prof. Georges Gielen
Prof. Said Hamdioui
(Delft University of Technology)
Prof. Joni Dambre
(Ghent University)

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Engineering Science (PhD): Electrical Engineering

March 2024

© 2024 KU Leuven – Faculty of Engineering Science
Uitgegeven in eigen beheer, Ji-Yung Lin, Celestijnenlaan 200A box 2402, B-3001 Leuven (Belgium)

Alle rechten voorbehouden. Niets uit deze uitgave mag worden vermenigvuldigd en/of openbaar gemaakt worden door middel van druk, fotokopie, microfilm, elektronisch of op welke andere wijze ook zonder voorafgaande schriftelijke toestemming van de uitgever.

All rights reserved. No part of the publication may be reproduced in any form by print, photoprint, microfilm, electronic or any other means without written permission from the publisher.

Acknowledgements

The accomplishment of this thesis is a joint effort of many people. Without their contribution, I could not have completed this work. Therefore, I would like to express my sincere gratitude to all the people who have been assisted in this work along the way of my research.

First of all, I would like to give my special thanks to my promoter, Prof. Francky Catthoor. You provided me the opportunity to pursue this exceptional academic goal and guided me along the way with encouragement and patience. Your enthusiasm and devotion to research and collaboration have always been an inspiration to me. Many thanks to you for all the guidance you have given to me.

I would also like to express my thanks to my daily supervisor, Prof. Pieter Weckx. You have been not only a mentor but also a friend to talk with. You have helped me a lot not only in research but also in my Belgian life. I especially cherish all the inspirational brainstorming with you about research questions. Many thanks to you for all the guidance both inside and outside my research work.

I would like to thank all the members of the supervision committee: I am grateful to Prof. Guido Groeseneken and Prof. Georges Gielen for your examinations on my work and suggestions on the research direction. I am also grateful to Prof. Joni Dambre and Prof. Said Hamdioui for your time in examining the thesis and giving constructive comments, especially on scientific rigorousness and research prospective. Your participation makes this thesis significantly better in quality.

The research on performance variability evaluation of CNT technology is conducted in collaboration with Stanford University. Therefore, I specifically thank the strong support of Carlo Gilardi and his colleagues at Stanford University on the variability distribution generation, compact model creation and library characterization of the CNFET technology, as well as the insightful

introduction and discussion on CNT technology.

Next, I would like to thank my imec colleagues. Thanks to Subrat Mishra and Alessio Spessot for your precious advice and assistance in this research. Thanks to Odysseas Zografos and Dwaipayan Biswas for all the suggestions during the writing. In addition, I would like to thank Hyungrock Oh, Halil Kukner, Mohit Gupta, Zhicheng Wu, and all other team members who have directly or indirectly helped during my PhD process. Your generosity in offering help has always warmed my heart.

I would also like to thank my Taiwanese fellows: Wei-Ming Wu, Kenny Wu, Thomas Tsau, Yu Fang, Tsang-Hsuan Wang, Wen-Chieh Chen, Medea Huang, Chao-Yuan Huang, Yi-Hsuan Li, and other Taiwanese students for your company. You have always been someone I can rely on to turn around a bad day. Together we had many good times traveling and exploring the world. Thank you all for the good fellowship.

I am very grateful to my parents and my sister. You have always been supportive of my decision to study abroad, even though it means I could not accompany you all the time. Because of your understanding, I can pursue my achievements without worries. In addition, you have been caring for my life, health and well-being but tolerant of my willfulness. Thank you for your support and consideration.

Finally, my deepest thanks to my wife, Zoey. It was you who encouraged me to step out of my comfort zone and go for this PhD in this land that was then uncharted to both of us. You have always loved me and given me all you have. You have been fully supportive of my life. You have always been understanding and consoled me when I was down. I feel extremely fortunate to be with you for the journey we have been through. You are my love and the reason for all my achievements.

Abstract

Performance variability, the phenomenon that the execution time of the same application varies every time it is executed, is ubiquitous in modern computers. Performance variability is especially detrimental to time-sensitive and time-critical applications, which are the applications required to be completed before a certain deadline. For time-sensitive applications like multimedia, violating deadlines may cause quality loss. For time-critical applications like avionics, violating deadlines will cause even more severe consequences, such as safety issues. Performance variability can originate from the device and circuit level, generated by process variations, voltage variations, temperature fluctuation, and aging. The variability can also be generated from the architecture and software levels, due to dynamic execution features of architectures and program flows of software. With continuous development toward smaller devices and more dynamic architectures and software, performance variability increases and threatens the reliability of both time-sensitive and time-critical applications.

To reduce performance variability, mitigation approaches are used in modern processors. Mitigation approaches use on-chip monitors to detect the amount of variability at run time. Based on the amount of variability and the prediction of future workload, a run-time controller changes the processor speed by altering system knobs, such as dynamic voltage and frequency scaling (DVFS). Among all kinds of mitigation approaches, workload-dependent approaches operate on the principle of using the design-time profiling information to reduce the uncertainties in future execution time calculation at run time. Therefore, workload-dependent approaches are the best option for minimizing system costs while reducing deadline misses.

This thesis focuses on the realization of a workload-dependent mitigation approach tackling various performance variability. We have improved the ways of tackling performance variability from the architecture and software levels based on two features. First, we introduce heterogeneous datapaths (HDP), a knob of faster switching time than DVFS, to improve the time granularity of

mitigation from the millisecond level to the microsecond level. By combining both the DVFS and HDP, this approach can tackle performance variability of multiple timescales down to the sub-millisecond level. Second, the run-time controller adopts the concept of dynamic scenarios in its workload-dependent scheduling algorithm, so a tight upper bound of future execution time can be calculated at run-time, which enables providing true timing guarantees. The cycle-accurate simulation results show that the proposed approach can ensure zero deadline misses with a smaller time slack than the conventional approach. This advantage can translate into an energy reduction of 32% on average.

Furthermore, we have developed a cross-layer reliability and variability mitigation framework to simultaneously ensure functional correctness and timing guarantees. It is composed of three layers. First, circuit-level mitigation like voltage droop prevention mitigates device-level variability in process, voltage, and aging, reducing register-level errors. Second, register-level mitigation like checkpointing and rollback-recovery corrects errors at the cost of execution time overhead. Last, at the processor-level, performance variability mitigation compensates for the execution time overhead at the cost of energy overhead. Cross-layer analysis of this framework results in the concept of the "error rate wall" (ERW), the system-specific error probability bound within which reliability can be ensured. ERW can be improved by optimizing system parameters, such as the number of checkpoints as shown by experiments.

We further extend the use of the mitigation framework to evaluate and mitigate the variability impacts of different technologies at the processor level. To instantiate this, the performance variability and mitigation costs of a processor of carbon nanotube field-effect transistors (CNFET) are evaluated. Results show that the performance of a processor consisting of the 6σ slow-corner CNFETs is 22.3% slower than the one of nominal CNFETs. And though it requires 80% extra energy consumption to boost the slow-corner processor to be at the same speed as the nominal processor, this energy consumption is still small than the nominal silicon CMOS processor by a ratio of 2.1.

Beknopte samenvatting

Prestatievariabiliteit, het fenomeen dat de uitvoeringstijd van dezelfde toepassing varieert elke keer dat deze wordt uitgevoerd, is alomtegenwoordig in moderne computers. Prestatievariabiliteit is vooral nadelig voor tijdgevoelige en tijdkritische toepassingen. Dit zijn toepassingen die voor een bepaalde deadline voltooid moeten zijn. Voor tijdgevoelige toepassingen zoals multimedia kan het schenden van deadlines namelijk kwaliteitsverlies veroorzaken. Voor tijdkritische toepassingen, zoals luchtvaartelektronica, heeft het overschrijden van deadlines zelfs nog ernstigere gevolgen, zoals veiligheidsproblemen. Prestatievariabiliteit kan ontstaan op apparaat- en circuitniveau, gegenereerd door procesvariaties, spanningsvariaties, temperatuurschommelingen en veroudering. De variabiliteit kan ook worden gegenereerd op architectuur- en softwareniveau, door dynamische uitvoeringskenmerken van de data- en controlostromen in de applicatiesoftware. Met de voortdurende ontwikkeling naar kleinere apparaten en dynamischere architecturen en software neemt de prestatievariabiliteit toe en dit vormt een bedreiging voor de betrouwbaarheid van zowel tijdgevoelige als tijdkritische toepassingen.

Om de prestatievariabiliteit te verminderen, worden in moderne processoren mitigatiebenaderingen gebruikt. Mitigatiebenaderingen gebruiken monitoren op de chip om de hoeveelheid variabiliteit tijdens het uitvoeren te detecteren. Op basis van de hoeveelheid variabiliteit en de voorspelling van de toekomstige werklast, verandert een controle-eenheid tijdens de uitvoering de processorsnelheid door de systeemknoppen aan te passen, zoals Dynamic Voltage and Frequency Scaling (DVFS). Van alle soorten risicobeperkende benaderingen werken werklastafhankelijke benaderingen volgens het principe van het gebruik van profileringsinformatie uit de ontwerptijd om de onzekerheden in de berekening van de toekomstige uitvoeringstijd tijdens de uitvoering te verminderen. Daarom zijn werklastafhankelijke benaderingen de beste optie voor het minimaliseren van systeemkosten terwijl ze het niet halen van de kritische tijdslimieten verminderen.

Deze dissertatie richt zich op de realisatie van een werklastafhankelijke mitigatiebenadering die verschillende prestatievariabelen aanpakt. We hebben de manieren om prestatievariabiliteit op architectuur- en softwareniveau aan te pakken verbeterd op basis van twee kenmerken. Ten eerste introduceren we heterogene datapaden (HDP), een knop met een snellere schakeltijd dan DVFS, om de tijdgranulariteit van mitigatie te verbeteren van milliseconden- naar microsecondenniveau. Door zowel DVFS als HDP te combineren, kan deze aanpak prestatievariabiliteit van meerdere tijdschalen aanpakken tot op submilliseconde-niveau. Ten tweede past de controle-eenheid tijdens de uitvoering het concept van dynamische scenario's toe in zijn werklastafhankelijke planningsalgoritme, zodat tijdens de uitvoering een strakke bovengrens van de toekomstige uitvoeringstijd kan worden berekend, waardoor echte tijdsgaranties opnieuw haalbaar zijn. De cyclusnauwkeurige simulatieresultaten laten zien dat de voorgestelde aanpak geen enkele kritische tijdslimiet mist en dit kan gegarandeerd worden met met een kleinere tijdspeling dan de conventionele aanpak. Dit voordeel kan zich vertalen in een energiebesparing van gemiddeld 32%.

Verder hebben we een raamwerk voor betrouwbaarheid en variabiliteitsmitigatie ontwikkeld dat over meerdere ontwerpniveaus actief is, en dat tegelijkertijd functionele correctheid en timinggaranties te garanderen. Het bestaat uit drie niveaus. Ten eerste, circuitniveau beperkingen zoals de preventie van spanningsreducties, verminderen de apparaatniveau variabiliteit in proces, spanning en veroudering, waardoor fouten op registerniveau worden vermeden. Ten tweede worden fouten gecorrigeerd op registerniveau, met technieken gekend onder de termen "checkpointing" en "rollback", ten koste van de uitvoeringstijd. Ten slotte compenseert op processorniveau de beperking van de prestatievariabiliteit de meerkost in uitvoeringstijd ten koste van de meerkost in energie. Niveauoverschrijdende analyse van dit raamwerk resulteert in het concept van de "error rate wall" (ERW), de systeemspecifieke foutkansgrens waarbinnen betrouwbaarheid kan worden gegarandeerd. ERW kan worden verbeterd door systeemparameters te optimaliseren, zoals het aantal controlepunten, zoals blijkt uit experimenten.

In een laatste bijdrage breiden we het gebruik van het voorgestelde mitigatie-raamwerk verder voor beperking verder uit om de variabiliteitsimpact van verschillende technologieën op processorniveau te evalueren en te beperken. Om dit te concretiseren worden de prestatievariabiliteit en mitigatiekosten van een processor met koolstofnanobuis-veldeffecttransistoren (CNFET) geëvalueerd. De resultaten laten zien dat de prestaties van een processor bestaande uit de langzame hoek CNFETs 22,3% langzamer is dan die van nominale CNFETs. En hoewel er 80% extra energieverbruik nodig is om de langzame-hoek CNFET processor op te voeren tot dezelfde snelheid als de nominale processor, is dit

energieverbruik nog steeds lager dan de nominale silicium CMOS-processor met een verhouding van 2,1.

List of Abbreviations

ADPCM	Adaptive Differential Pulse-Code Modulation
AI	Artificial Intelligence
ALU	Arithmetic Logic Unit
BE	Best-Effort
BEOL	Back-End-of-Line
BPP	Bit Per Pixel
BTI	Bias Temperature Instability
CMOS	Complementary Metal-Oxide-Semiconductor
CNFET	Carbon Nanotube Field-Effect Transistor
CNT	Carbon Nanotube
CRR	Checkpointing and Rollback-Recovery
CSR	Control and Status Register
DC	Direct Current
DEMUX	Demultiplexer
DMA	Domain-Specific Accelerator
DMR	Dual Modular Redundancy
DS	Dynamic Scenario
DVFS	Dynamic Voltage and Frequency Scaling
ECC	Error Correction Circuitry
EDP	Energy-Delay Product
EM	Electromigration
ERW	Error Rate Wall
FEOL	Front-End-of-Line
GPU	Graphic Processing Unit
HCI	Hot Carrier Injection
HDP	Heterogeneous Datapath
HW	Hardware
ICG	Integrated Clock Gating
IPC	Instruction Per Cycle
ISA	Instruction Set Architecture

JPEG	Joint Photographic Experts Group
LER	Line Edge Roughness
m-CNT	Metallic Carbon Nanotube
MCU	Minimum Coded Unit
MEOL	Middle-End-of-Line
MGG	Metal Grain Granularity
MOSFET	Metal-Oxide-Semiconductor Field-Effect Transistor
MP2	MPEG-1 Audio Layer II
MPEG	Moving Picture Experts Group
MUX	Multiplexer
MVSF	Multiple-Voltage/Single-Frequency
NBTI	Negative Bias Temperature Instability
OS	Operating System
P&R	Place and Route
PBTI	Positive Bias Temperature Instability
PC	Personal Computer
PG	Power Gating
PLL	Phase-Locked Loop
QCIF	Quarter Common Intermediate Format
RDF	Random Dopant Fluctuation
RFID	Radio-Frequency Identification
RGG	Ratio of the Average Grain Size to the Gate Area
RTL	Register-Transfer Level
RTN	Random Telegraph Noise
RTS	Run-Time Situation
s-CNT	Semiconducting Carbon Nanotube
SoC	System-on-Chip
SRAM	Static Random-Access Memory
SVMF	Single-Voltage/Multiple-Frequency
SW	Software
TDDB	Time-Dependent Dielectric Breakdown
TDP	Thermal Design Budget
TMR	Triple Modular Redundancy
TN	Thread Node
VLSI	Very Large Scale Integration
WCET	Worst-Case Execution Time
WD	Workload-Dependent
WFV	Work Function Variation

Contents

Acknowledgements i

Abstract iii

Beknopte samenvatting v

List of Abbreviations ix

Contents xi

List of Figures xv

List of Tables xxi

1 Introduction 1
- 1.1 Timing Guarantees of Applications . . . 1
- 1.2 Performance Variability in Integrated Systems . . . 4
- 1.3 Main Gaps in Related Research . . . 6
- 1.4 Problem Statement . . . 7
- 1.5 Contributions of this Thesis . . . 8
- 1.6 Structure of this Thesis . . . 9

2 Variability in Integrated Circuits 11
- 2.1 Device- and Circuit- Level Variability . . . 11
  - 2.1.1 Process Variation . . . 12
  - 2.1.2 Voltage Variation . . . 15
  - 2.1.3 Temperature Variation . . . 17
  - 2.1.4 Aging Effect . . . 17
- 2.2 Architecture-Level Performance Variability . . . 21
- 2.3 Software-Level Performance Variability . . . 23
- 2.4 Approaches to Tackling Variability . . . 25

2.4.1 Guardbanding . . . 25
2.4.2 Run-Time Mitigation . . . 26
2.5 Summary . . . 28

3 Mitigation Approaches for Performance Variability: State-of-the-Art 29
3.1 Structure of Mitigation Approaches for Performance Variability 29
3.2 System Knobs . . . 30
3.2.1 Dynamic Voltage and Frequency Scaling (DVFS) . . . 31
3.2.2 Task Mapping and task migration . . . 33
3.2.3 Gas Pedals . . . 34
3.3 Adaptive Control Algorithms . . . 35
3.3.1 Algorithms without Timing Guarantees . . . 36
3.3.2 Algorithms with Timing Guarantees . . . 37
3.4 Summary . . . 41

4 Workload-Dependent Multi-Timescale Mitigation Approach for Performance Variability: Theory 43
4.1 Motivation and Principles . . . 44
4.2 Heterogeneous Datapaths (HDP) . . . 47
4.3 Collaboration of Multiple Knobs . . . 52
4.4 Dynamic-Scenario-Based Scheduling Algorithm . . . 54
4.4.1 Principles of Dynamic Scenarios . . . 55
4.4.2 Design-Time Scenario Clustering . . . 60
4.4.3 Priniciples of the run-time scheduling algorithm . . . 63
4.4.4 Run-Time Scheduling for a Single Knob . . . 65
4.4.5 Run-Time Scheduling for Multiple Knobs . . . 68
4.4.6 Overheads of the Algorithm . . . 73
4.5 Proof of Timing Guarantees . . . 74
4.5.1 Summary of the Mathematical Model . . . 74
4.5.2 Instantiation of the Mathematical model . . . 76
4.5.3 Proof of Hard Constraints . . . 81
4.6 Applicability to other architectures and technologies . . . 86
4.7 Summary . . . 87

5 Workload-Dependent Multi-Timescale Mitigation Approach for Performance Variability: Experiments 89
5.1 Experimental Setup . . . 90
5.1.1 Processor Simulation Flow . . . 90
5.1.2 Overall Simulation Flow . . . 92
5.2 Case Study I: Lower-Band Quantization of ADPCM Encoding Application . . . 94
5.2.1 Design-Time Scenario Clustering . . . 96
5.2.2 Run-Time Mitigation Results . . . 99

5.3 Case Study II: JPEG Decoding Application . . . 103
5.3.1 Design-Time Scenario Clustering . . . 104
5.3.2 Run-Time Mitigation Results . . . 107
5.4 Discussion of the Case Studies . . . 110
5.5 Summary . . . 111

6 Cross-Layer Reliability and Variability Mitigation Framework: Theory 113
6.1 Motivation and Principles . . . 114
6.2 Mitigation of Device-Level Variability . . . 116
6.2.1 State-of-the-Art Mitigation Approaches . . . 116
6.2.2 Register-Level Errors . . . 118
6.3 Register-Layer Error Correction . . . 120
6.3.1 Error Correction Techniques . . . 120
6.3.2 HW/SW Reliability Mitigation with OCEAN . . . 122
6.3.3 Theoretical Analysis of Time Overhead . . . 123
6.3.4 Optimal Checkpointing . . . 126
6.4 Processor-Layer Mitigation of Performance Variability . . . 129
6.5 System Reliability Evaluation . . . 130
6.6 Discussion . . . 133
6.6.1 Comparison with Other Cross-Layer Mitigation Approaches 133
6.6.2 Applicability to other architectures and technologies . . . 134
6.7 Summary . . . 135

7 Cross-Layer Reliability and Variability Mitigation Framework: Experiments 137
7.1 Cross-layer Simulation Framework . . . 138
7.2 Simulation of Register-Level Errors . . . 140
7.3 Cross-layer Interdependency Analysis . . . 142
7.3.1 Simulation Flow and Experimental Setup . . . 144
7.3.2 Results and Discussion . . . 147
7.4 Analysis with Optimal Checkpointing . . . 149
7.4.1 Experimental Setup . . . 149
7.4.2 Results and Discussion . . . 150
7.5 Summary . . . 153

8 Performance Variability Evaluation of the Carbon Nanotube Technology 155
8.1 Introduction to Carbon Nanotube Field-Effect Transistors (CNFET) . . . 156
8.2 Variability of CNFET Devices . . . 157
8.3 Processor-level Evaluation of Performance Variability . . . 161
8.3.1 Technology Set-ups and Variability Models . . . 161
8.3.2 Experimental Setup . . . 164

8.3.3 Results and Discussion 166
8.4 Mitigation Cost Estimation 168
8.5 Discussion 170
8.6 Summary 171

9 Conclusion and Future Work **173**
9.1 Main Messages 173
9.2 Contributions 174
9.3 Perspective 176

Bibliography **179**

List of Publications **201**

List of Figures

1.1 Intel CPU transistor count trend. Replotted from [1] 2
1.2 General requirements of chip reliability 3
1.3 Dynamism in execution time of a MP2 decoding application for decoding of Vivaldi's "Four Seasons" Concerti [2]. Each data point represents the execution time of decoding a bitstream frame of the audio stream. 5
1.4 Main sources of performance variability in every abstraction layer 5
1.5 Structure of the thesis and overview of the chapters 10

2.1 Taxonomy of the sources of device- and circuit-level variability. Only the main sources are enumerated and the miscellaneous sources are left out for clarity. 12
2.2 The trend of the average number of dopant atoms reduced with technology nodes. Replotted from [3] 13
2.3 LER measurements for advanced lithography processes reported by various labs. The dashed line indicates the required levels in Semiconductor Industry Association (SIA) roadmap. The inset shows LER observed in sub-100nm e-beam generated lines. Replotted from [4]. 14
2.4 Relations of WFV and RGG for various gate materials (TiN, TaN, WN, MoN, and RuMo [5]). Replotted from [6]. 15
2.5 Illustration of a voltage droop occurrence, including all its frequency components. Replotted from [7] 16
2.6 The mechanisms of RTN and BTI. Replotted from [8]. (a) The system is in dynamic equilibrium, manifested by Random Telegraph Noise (RTN) in small devices. (b) With a gate voltage stress, the system is approaching the dynamic equilibrium with the gradual discharging of excess charged oxide defects, known as BTI. 18

2.7 The downscaling of devices reduces the number of defects to only a handful. As a result, the stochastic properties become prominent, resulting in large variations in device performance and lifetime. Replotted from [9]. 19
2.8 Block diagrams of ARM Zen 2 "Matisse" SoC architecture. Replotted from [10]. (a) the architecture of a "core complex" (CCX). (b) the architecture of the whole SoC, composed of four CCXs. Infinity Fabric is a proprietary interconnection architecture which facilitates data transfers among linked components. 22
2.9 Illustrative example of performance variability generated by branch prediction (a) control flow of a task containing one branch (b) time diagram of the processor in cases of either branch prediction hit or miss. 23
2.10 Performance variability of a JPEG decoder application [11] (a) block diagram of the JPEG decoder. Replotted from [12] (b) execution cycles of the JPEC decoder, benchmarked with different input images of the same size on a RISC-V ISA simulator, assuming $IPC = 1$ [13] 24
2.11 Impact of each non-ideality (quantified in V_{dd} terms) on SRAM supply voltage guardband under different CMOS technologies. The total guardband would reach the same level as the minimum V_{dd} with the technology downscaling, limiting the voltage scaling. Replotted from [14] 26
2.12 Block diagram of a run-time mitigation system. Replotted from [15] 27

3.1 Block diagram of the mitigation approach for performance variability 30
3.2 Variability in execution time (seconds) and energy consumption (J) when running the Blackscholes application (from PARSEC benchmark [16]) using different core combinations on ARM's big.LITTLE architecture which contains four big (B) and four LITTLE (L) cores. Replotted from [17] 33
3.3 Comparison of reliability (deadline hit rate) and cost (energy, etc.) among different types of adaptive control algorithms: best-effort (BE), worst-case execution time based (WCET), and workload-dependent (WD). 35
3.4 Scheduling example of algorithms with timing guarantees . . . 38

4.1 Example of how a deadline miss could occur if switching time and job execution time are comparable. The unexpected long execution time of Job $(N-1)$ plus the long switching time leave almost no time budget for Job N, resulting in the deadline miss. 45

4.2 Block diagram of the workload-dependent multi-timescale mitigation approach 46
4.3 Block diagram of the Ariane core. Replotted from [18] 48
4.4 Schematic of the HDP design in the ALUs of Ariane core . . . 48
4.5 Layout of the fast ALU of the HDP 50
4.6 Energy consumption v.s. clock frequency of physical designs of ALU. 50
4.7 Variability sources and mitigation knobs arranged by the timescales, which tells whether the knobs can react faster than the variability for successfully mitigation. 53
4.8 Speed and energy trade-off of the ALUs running in each of the possible combinations of the DVFS mode (0.5V–0.9V) and the HDP mode (fast, mid, and slow) 55
4.9 Depictions of the basic concepts of dynamic scenarios. (a) At design time, different parts of the workload are classified into run-time situations (RTS). (b) RTSs are clustering into scenarios based on their timing and energy costs. (c) the monitoring parameters and the system costs of scenarios and the monitoring parameters are recorded in a table. (d) at run time, system costs are predicted based on the scenarios 57
4.10 Basic flow of realizing a solution to dynamism-related problems based on dynamic scenario principles 58
4.11 Process of generating a scenario set in the design-time process of the dynamic scenario based approach 61
4.12 The generic flow of DS-instantiated scheduling algorithms . . . 63
4.13 Refined upper bound process to calculate the minimum required speed for the next TN for ensuring timing guarantee 67
4.14 Flow diagram of the run-time process of the multi-timescale DS based approach, in which the modes of knobs are decided sequentially from the slow-switching ones to the fast-switching ones 69
4.15 Enhanced refined upper bound process to calculate the minimum required speed for the next TN for ensuring timing guarantee in the multiple-knob scheduling algorithm 72

5.1 The processor simulation flow of benchmarking the execution time and HDP energy consumption for running a certain workload 91
5.2 The cycle-accurate simulation flow to acquire the execution time and energy consumption when running a workload under mitigation 93
5.3 Block diagram of the encoding process of ADPCM G.722 standard [19] 95
5.4 Illustration of the samples of an acoustic wave of 22050 Hz, used as the input data of the workload of the lower sub-band quantization block of ADPCM-encoding 96

5.5 Distribution of the instruction count of the TNs of the ADPCM quantization workload, in which the great difference between WCET and realistic cases are shown 97
5.6 The relation between the volume and the number of instructions for each TN. TNs are clustered into scenarios by volume. . . . 98
5.7 Selected modes over the entire run of the ADPCM quantization workload with the 1.0-ms frame period. (a) DVFS mode (b) HDP mode 100
5.8 Trace of slack over the entire run of the ADPCM quantization workload with the 1.0-ms frame period 101
5.9 ADPCM quantization workload energy consumption of different methods under different time-criticality conditions 102
5.10 Program flow of the JPEG decoding application. In the diagram, each step (grey square block) of the program flow is defined as a TN. 104
5.11 Relation between the number of instructions and the BPP for the TNs of decoding an MCU row. It shows the high influence of BPP and MCU row location on the number of instructions. . 105
5.12 Relation between the execution time under the 0.7V/fast HDP mode and the BPP for the TNs of the initialization step. It shows the execution time is almost constant regardless of the input image content. 106
5.13 Selected modes over the entire run of the JPEG decoding workload with a 8.0-ms frame period. (a) DVFS mode (b) HDP mode 108
5.14 Trace of slack over the entire run of the JPEG decoding workload with a 8.0-ms frame period 109
5.15 JPEG decoding workload energy consumption of different methods under different time-criticality conditions 109

6.1 Overall structure of the cross-layer mitigation framework 114
6.2 Critical path monitor distribution in each core of IBM POWER7™ processor [20]. The AND buffers are used to aggregate the monitor signals and the final signals could trigger knobs such as throttling or the digital phase-locked loop (DPLL). 117
6.3 Measurement of a droop event with and without the voltage droop prevention approach [20] 117
6.4 Relation between the error rate and the margin 118
6.5 Schematic diagram of OCEAN, the optimized HW/SW reliability mitigation approach [21] 121

6.6 A CRR example of dividing a task T1 to five phases (P1-P5), showing the impact on error mitigation. The faulty task/phase is shaded with light grey, the checkpoints are shaded with dark grey, and the data chunks are shaded in stripe-patterned blue [21] 123
6.7 Probability distribution of the number of rollbacks of a segment, assuming the error probability of the segment is 0.2 124
6.8 Expected number of rollbacks for a segment of 10^5 cycles, under different error probability of cycles 125
6.9 The expected value of the total execution cycles normalized to the original execution cycles for the segment (10^5 cycles), under different error probability of each cycle. 125
6.10 Example of how the number of checkpoints affects the time overhead under different error probability conditions 126
6.11 Relations between time overhead and the number of checkpoints for a segment of 10^5 cycles, under different error probability of cycles (p) 127
6.12 Optimal number of checkpoints under different error probabilities of each cycle 128
6.13 Expected value of total execution cycles with the optimal number of checkpoints compared to the scheme of one checkpoint per segment, under different error probabilities of each cycle 129
6.14 Illustration of the limit of the performance variability mitigation in mitigating the execution cycle overhead of a workload 131
6.15 Error rate walls under 0.5 utilization of the original workload in the CRR schemes of Figure 6.13 133

7.1 Simulation framework of the cross-layer reliability and variability mitigation 138
7.2 Workload-dependent error simulation flow 141
7.3 Error rate per cycle under different clock frequencies 141
7.4 Illustration of an example of how CRR and performance variability mitigation can collaborate to meet the dual reliability requirements 143
7.5 Interdependency of CRR and performance variability mitigation 143
7.6 Cross-layer simulation flow incorporating the register-layer CRR and the processor-layer performance variability mitigation . . . 144
7.7 Average rollbacks of every computation phase under different error probability conditions 146
7.8 Total executed cycles normalized to the cycles of the original workload, under different error probability conditions 146
7.9 Deadline hit rate under different error probability conditions . 147
7.10 Total energy consumption normalized to the energy consumption of the original workload, under different error probability conditions 148

7.11 Optimal number of checkpoints per TN under different error probability conditions . . . 150
7.12 Total executed cycles with different numbers of checkpoints per TN (n_{cp}), under different error probability conditions. The numbers of executed cycles are normalized to the cycles of the original workload. The red dash line indicates the error rate wall under the optimal n_{cp} condition. . . . 151
7.13 Deadline hit rate with different numbers of checkpoints per TN (n_{cp}), under different error probability conditions. The curves of $n_{cp} = 1000$ and optimal n_{cp} overlaps with each other, yielding similar deadline hit rates. . . . 152
7.14 Total energy consumption with different numbers of checkpoints per TN (n_{cp}), under different error probability conditions. The energy consumptions are normalized to the one of the original workload . . . 153

8.1 3D view of the planer CNFET structure. Replotted from [22] . 156
8.2 Energy vs. clock frequency results across different CNFET and silicon technology options of 5 nm node on 32-bit commercial processor core. Replotted from [23] . . . 157
8.3 Taxonomy of variability in CNFETs. Only the main sources are enumerated and the miscellaneous sources are left out for clarity. 158
8.4 Simulated I_{on} variability of minimum-width CNFET at 32nm technology node, contributed by various variability sources. Replotted from [24] . . . 160
8.5 Generation process of the distribution of CNT count per device 163
8.6 Simulation flow of processor-level variability evaluation of CNFET and silicon CMOS technology . . . 165
8.7 Energy v.s. achieved frequency of the Pareto-optimal physical designs of each corner, normalized to the same arbitrary units. (a) All curves (b) CNFET technology (c) 2nm silicon CMOS technology . . . 167
8.8 Summary of speed and performance variability of CNFET and silicon CMOS (a) maximum achieved frequency (b) maximum achieved frequency normalized to respective nominal corners . . 168
8.9 Reposition of DVFS operating points from the SVMF scheme to the MVSF scheme. The mitigation costs are indicated by the voltage, frequency, and energy differences between the two sets of DVFS points. . . . 169
8.10 The DVFS relations of each corner of CNFET designs, including the operating points of both SVMF and MVSF schemes (a) supply voltage v.s. clock frequency (b) energy per cycle v.s. clock frequency . . . 170

List of Tables

3.1 DVFS modes of the Intel Core2 Duo E6850 processor [25] 31

4.1 Summary of the ALU Designs in the HDP 51
4.2 Denotations of the mathematical model of scheduling with hard timing constraints 75
4.3 Relations between the variables in the math model and in our proposed algorithm 77

5.1 Summary of DVFS model 92
5.2 Scenario Look-up Table for the ADPCM Quantization Application 98
5.3 Scenario Look-up Table for the JPEG-Decoding Application . 106
5.4 Mitigation result summary with different methodologies and benchmarks 110

8.1 BEOL assumptions in physical designs 162
8.2 Device parameters of each design corner of CNFET and silicon CMOS technologies 164

Chapter 1

Introduction

This chapter introduces the research topic of this thesis, including the background, motivation, scope, and significance of the work. We will start with an introduction of the goal of this work, the timing guarantee requirements of applications in Section 1.1. Then, Section 1.2 follows up with performance variability in integrated systems, which is the main threat to timing guarantees. Section 1.3 discusses the main gaps in the literature in the field of timing guarantees. Section 1.4 describes the problem statement this thesis aims to solve, and Section 1.5 highlights the main contributions of our work in filling the gaps of the state-of-the-art. Finally, Section 1.6 describes the structure of the thesis and the content of each chapter.

1.1 Timing Guarantees of Applications

With the rapid development of the semiconductor transistor technology, electronic devices made of Very Large Scale Integration (VLSI) chips become necessities in our daily lives, with countless functions such as computation, communication, entertainment, etc. Ever since the early age of VLSI chips in the 1970s, the number of transistors per area approximately doubled every year, following Moore's Law [26] as depicted in Figure 1.1. In the modern era, a typical chip can contain billions of transistors. The continuous transistor downscaling has fueled great improvement in the chip performance and cost. Over the past 40 years, the performance has grown by 50,000× [27] and the price per transistor has been cut by 1,000,000× [28], enabling wide ranges of applications from the tiny RFID tags for object identification to the gargantuan

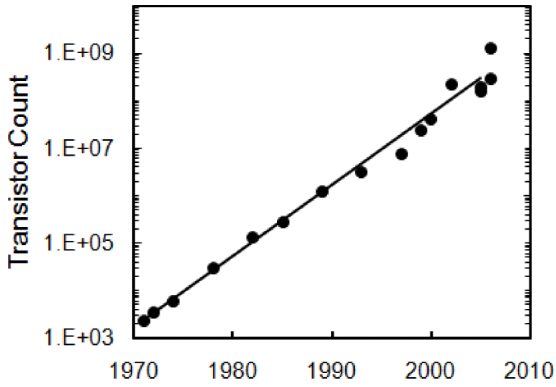


Figure 1.1: Intel CPU transistor count trend. Replotted from [1]

supercomputers driving scientific computations such as weather forecasts. The huge number of transistors also enables complicated system designs. A highly integrated system-on-chip (SoC) design can contain hundreds of processor cores, multiple levels of memory, and sophisticated interconnection networks all inside a single chip.

Even with the high transistor counts and system complexity, VLSI chips are required to ensure reliability throughout the whole lifespan. The VLSI chips are required to operate without failures regardless of the changeable internal and external conditions. In general, the following are the reliability requirements for a typical chip:

- **Functional correctness**: the chip needs to operate without failing and produce correct computation results under all circumstances.
- **Timing guarantees**: the chip needs to ensure the timeliness of delivering the computation results. The results could be obsolete if not produced in time, and thus fail to serve the purpose they are designed for.
- **Thermal constraints**: the chip temperature needs to be limited under a certain level even though the devices and the interconnects dissipate heat. Otherwise, thermal runaway could happen and the devices and interconnects could break down due to the high temperature.
- **Parametric correctness**: beyond the aforementioned hard reliability requirements, there are other parametric requirements for different metrics specific to each field. For example, energy consumption is limited for battery-powered and energy-harvesting platforms.



Figure 1.2: General requirements of chip reliability

- **Lifetime**: the aforementioned requirements need to be ensured for the whole lifetime specification even though the chip could wear out. Normally consumer products need to be reliable for 5 years, while safety-critical products need to work for at least 10 years.

Timing guarantee is the main focus of this thesis among all reliability requirements. Among the diverse applications of VLSI chips, most applications care about the timeliness of delivering the calculation results. In practice, results are required to be calculated before certain deadlines. Failing to meet the deadline constraints will result in numerous consequences from quality loss to safety concerns. Based on the criticality of ensuring timing guarantees, applications can be further classified into three groups:

- **Time-critical**: timeliness is vital for the function of these applications, where they must always meet their timing specifications without any deadline violations. For these applications, missing deadlines might cause catastrophic outcomes such as safety problems. Vehicles, avionics and biomedical systems are examples of time-critical applications. Since these systems closely interact with the real-world environment and physical bodies, delayed computation results mean that the results are based on outdated information about the real world, causing dysfunction which could cost human lives.

- **Time-sensitive**: deadline violations cause quality loss for these applications. Occasional deadline misses are tolerable but must be guaranteed to be below a specified threshold. Hence, the variability and reliability management system should be deployed to monitor whether this timing specification is fulfilled, and proactively prevent potential violation of the specification. Multimedia such as real-time video is one example of this type. The timing of a video is specified by its frame rate, which defines the frequency at which consecutive images (i.e. frames) are displayed on the screen. In real-time video processing, failing to present a frame in time occasionally is acceptable, since it may not be noticeable by human eyes. However, losing several frames in a sequence can cause visible video lagging, significantly degrading user experiences, which should be prevented.
- **General**: even though without obvious timing specifications, nearly all the remaining applications have implicit maximum latency constraints, in consideration of user experiences. For instance, users of consumer products (e.g. mobile phones or PCs) are not willing to wait longer than a certain time for a response from their application running on their processing platform. Thus, the latency violation rate should stay below a small threshold, which has to be guaranteed for products to be economically viable in the market.

In conclusion, timing requirements are prevalent in every application domain. Therefore, ensuring timing guarantees is a crucial property for VLSI chips to achieve high quality and reliability of their functions.

1.2 Performance Variability in Integrated Systems

Despite the importance of timing guarantees, the execution time of applications in modern VLSI chips tends to be dynamic and unpredictable. The execution time of an application can be different every time it is executed, due to numerous variability sources. The term "performance variability" is used to describe the variations in execution time, which poses serious threats to timing guarantees.

As an illustrative example of performance variability, Figure 1.3 shows the dynamism in the execution time of a MPEG-1 Audio Layer II (MP2) decoding application for decoding bitstream frames of Vivaldi's "Four Seasons" Concerti, in which each data point represents the execution time of decoding a bitstream frame of the audio stream. The measured execution time for processing each frame is highly diverse, ranging from below one second to above two seconds.

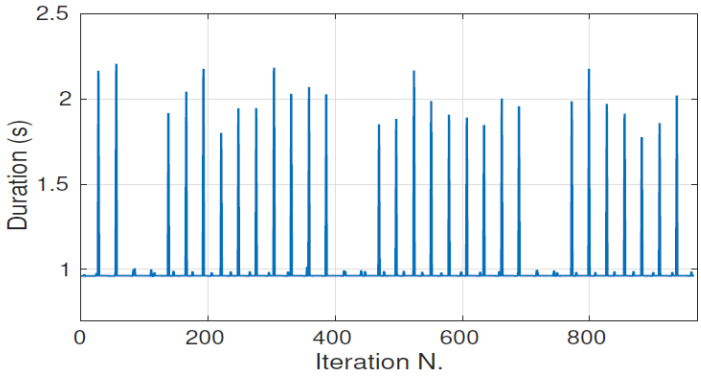


Figure 1.3: Dynamism in execution time of a MP2 decoding application for decoding of Vivaldi's "Four Seasons" Concerti [2]. Each data point represents the execution time of decoding a bitstream frame of the audio stream.

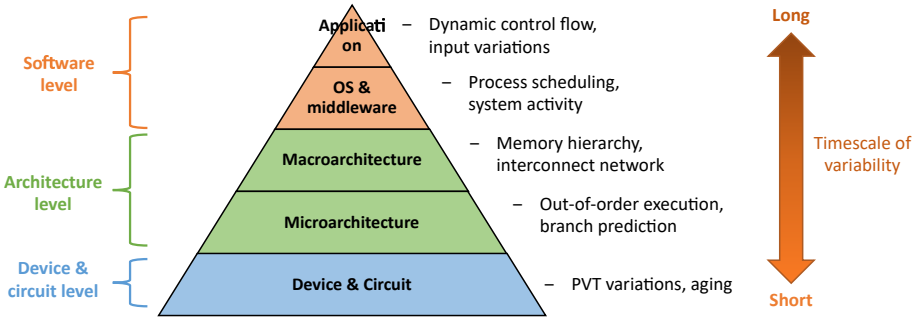


Figure 1.4: Main sources of performance variability in every abstraction layer

Performance variability can be originated from all abstraction layers from the device level to the application level. Figure 1.4 instantiates typical performance variability sources in each level. The timescale of each variability source is widely diverse. In general, the higher the level, the larger the timescale of a single variability event. For example, at the lowest device level, the delay jitter of a logic gate caused by bias temperature instability (BTI) can be even smaller than one picosecond [29]. At the circuit level, the glitches in the voltage supply can oscillate with a period smaller than one nanosecond [7]. At the microarchitecture level, whether a branch prediction hits or misses results in tens of cycles difference in executed cycles [30], equivalently tens of nanoseconds on a 1-GHz processor core. At the highest application level, different input data

can result in seconds of difference in execution time, as the example of Figure 1.3.

With the endless pursuit of technology scaling and system integration, performance variability is increasing at all levels. At the device level, the downscaling of device dimensions makes the fabrication process more and more difficult to control, which increases variability among devices. At the architecture level, the increase in transistor count enables more and more dynamic, complicated, and heterogeneous architectures, which increases variability in execution time. At the software level, the trends toward dynamic execution flow and distributed computation also introduce more variables in execution time, which exacerbates performance variability.

Therefore, mitigation approaches for performance variability are indispensable to ensure timing guarantees. However, with various variability sources from all abstraction layers, a holistic solution to performance variability is no easy task. Moreover, interdependencies can be present across multiple layers, further complexing the problem. Hence, the main topic of this thesis is to overcome these challenges to develop solutions to performance variability.

1.3 Main Gaps in Related Research

Though approaches for ensuring timing guarantees have been developed and used over the past years, advances in technology and computer architecture have brought new challenges to timing guarantees. Therefore, we suppose there is still plenty of room for improvement in the state-of-the-art approaches. In the following, we briefly introduce the important gaps in the related field, while the details will be discussed in later chapters.

- Calculation of the upper bound of execution time is the key to ensuring timing guarantees for any approach. In the past, Worst-Case Execution Time (WCET) [31, 32] analysis was developed to compute the upper bound of the execution time of an application under all possible variability conditions. While WCET is the foundation of the conventional approaches of timing guarantees, the complexity of modern hardware and computer architectures greatly increases the execution time overhead induced by WCET to an unacceptable amount, making WCET-based approaches undesirable. Since then, the specific problem formulation of mitigating performance variability with full timing guarantees has not been achieved up to now in state-of-the-art. Many researchers have claimed that only static and highly pessimistic upper-bound techniques can ever accomplish

this. In this context, though many mitigation approaches have been proposed, their guarantees of timing assurance or deadline miss rate are questionable [31]. Nevertheless, we suppose that the conjecture of no possible timing guaranteed approach is too pessimistic, and that timing guarantees can be ensured by run-time workload-dependent approaches, so a paradigm shift toward full timing guarantee can be created. More details are discussed in Chapter 3.

- To ensure timing guarantees, real-time scheduling is the most prominent approach [33]. However, the research on real-time scheduling is normally based on simplified mathematical processor models, which cannot fully reflect the performance variability below the software level. Therefore, the state-of-the-art approaches are insufficient in dealing with lower-level variability. Since performance variability comes from each abstraction layer in different timescales, it is necessary to establish a cross-layer framework which combines all techniques involved in the performance variability of each layer. In this way, timing guarantees can be holistically ensured and mitigation costs can be accurately evaluated. More details are discussed in Chapter 6.
- As far as other reliability requirements (e.g. functional correctness) are concerned, there is a trend of integrating multiple techniques to deal with multiple reliability objectives in the recent development of mitigation approaches. For example, real-time scheduling can be combined with checkpointing and rollback-recovery as well as thermal management to tackle three reliability objectives simultaneously [34]. While these techniques extend the scope of mitigation approaches for reliability, a holistic framework is required to solve the interdependencies of all these objectives across all abstraction layers. This kind of framework could pave the way for a unified methodology for all reliability objectives can be created. More details are discussed in Chapter 6.

1.4 Problem Statement

This research aims to develop a methodology for ensuring timing guarantees in modern processors and applications. The conventional WCET-based approaches for ensuring timing guarantees are based on imposing a total execution time margin of the worst-case conditions of all performance variability sources. However, with the growing complexity of modern computer architecture and applications, as well as the increasing variability in advanced technology nodes such as voltage variations, the execution time margin considered in WCET becomes unacceptably high, inducing too much overhead in performance and

energy. Hence, new methodologies are required to ensure timing guarantees for state-of-the-art processors and applications. We suppose that run-time workload-dependent mitigation approaches, which can adapt to every run-time situation and only add the required overhead to meet the timing requirements, can be the new methodology that ensures timing guarantees with low overhead. Therefore, the goal of this research is to realize such a methodology.

1.5 Contributions of this Thesis

We summarize the contributions of this thesis to the state-of-the-art as follows:

- We have developed a novel workload-dependent multi-timescale mitigation approach, which can ensure full timing guarantees in the presence of architecture-level and software-level performance variability of different timescales down to the sub-millisecond level, while the energy consumption is minimized at the same time. This approach aims to mitigate performance variability for timing guarantees of all application types (time-critical, time-sensitive, and general). This approach has two main features which improve the state-of-the-art approaches. First, the heterogeneous datapath (HDP) knob enables fast switching between different processor speeds, so that the performance variability can be tackled in sub-millisecond granularity. Second, on the principle of dynamic scenarios, the proposed approach can derive the upper bound of execution time at run time to prevent deadline misses.
- To evaluate the mitigation approach, we have developed a cycle-accurate simulation flow, which can simulate the performance variability in cycle-level granularity. The flow enables the benchmark of important processor metrics (execution time, energy consumption, etc.) to compare the reliability and costs achieved by different performance variability mitigation approaches.
- We develop a cross-layer reliability and variability mitigation framework, which can tackle the device-level and circuit-level performance variability to ensure both functional correctness and timing guarantees. The framework combines mitigation approaches of three different layers, including mitigation approaches of device-level variability (e.g. voltage variations) at the circuit level, checkpointing and rollback-recovery at the register level, and performance variability mitigation at the processor level.

- With the aforementioned framework, analysis of cross-layer dependencies and the holistic system impact of variability is performed. Co-simulation across layers shows the cost and the system reliability limit in ensuring the dual reliability requirements. The simulation results reveal the phenomenon of the "error rate wall" (ERW), a system-specific error rate limit beyond which reliability cannot be ensured. In addition, we show that the reliability of the overall system can be improved by changing system parameters, which indicates the potential of optimizing system parameters to enhance reliability.
- The use of the mitigation framework is further extended to evaluate and mitigate the variability impacts of new emerging technologies, such as the carbon nanotube field-effect transistors (CNFET) in our case study. In this study, the performance variability and mitigation costs between CNFET and silicon CMOS technologies are compared. Results show that before mitigation, the performance of a processor consisting of the 6σ slow-corner CNFETs is 22.3% slower than the one of nominal CNFETs. The performance variability of CNFETs is smaller than the silicon CMOS. After mitigation, the slow-corner CNFET processor can be at the same speed as the nominal-corner CNFET processor with 80% energy cost. Even with the energy cost, the energy consumption is still 2.1× smaller than the processor of silicon CMOS.

1.6 Structure of this Thesis

The structure of the thesis is shown in Figure 1.5: In Chapter 2, we give an overview of variability in integrated circuits, in which every source of performance variability in each layer is addressed. Moreover, we discuss the general principles of the approaches to tackling variability, including guardbanding and run-time mitigation. In Chapter 3, we elaborate on the state-of-the-art mitigation approaches for performance variability, in which each component (monitors, system knobs, adaptive control algorithms) is discussed in detail. In Chapter 4 & 5, we present the workload-dependent multi-timescale approach we have developed for tackling performance variability. Chapter 4 introduces the theoretical basis and the design of this approach, while Chapter 5 shows the simulation flow and the results. Thereafter, we propose the cross-layer reliability and variability mitigation framework in Chapter 6 & 7. The principles and theoretical basis are described in Chapter 6, and the simulation results are discussed in Chapter 7. In Chapter 8, the framework is extended to evaluate the variability impacts and mitigation costs of CNFETs and silicon CMOS at

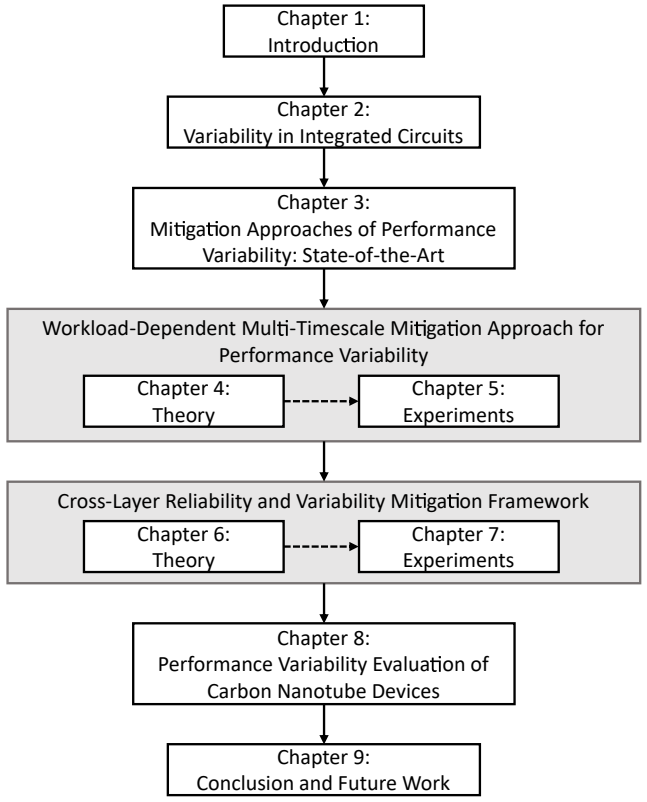


Figure 1.5: Structure of the thesis and overview of the chapters

the processor level. Finally, Chapter 9 summarizes the main messages and contributions of this thesis, and discusses the future perspective of this work.

Chapter 2

Variability in Integrated Circuits

In this chapter, we elaborate on a wide variety of general variability sources in all abstraction layers. Each variability source can affect many quality metrics, such as speed, power, reliability, etc. Nevertheless, we focus on how these variability sources would cause performance variability in integrated systems. Furthermore, we discuss the general strategies used in tackling variability.

This chapter is structured as follows: in the first three sections, we elaborate on different variability sources from lower levels to higher levels. First, device- and circuit-level variability sources are discussed in Section 2.1. Then, architecture-level variability sources are discussed in Section 2.2. Software-level variability sources are discussed in Section 2.3. Finally, we elaborate on the main approaches in tackling variability in Section 2.4.

2.1 Device- and Circuit- Level Variability

Performances of devices and circuits of the same design are intended to be identical from one to another, and constant over the whole lifetime. However, due to numerous non-ideal effects, their performances are actually nonuniform and change over time. Due to process variation [35], differences already exist among devices and circuits when just fabricated, known as *time-zero variability*. Then, when in operation, their performance will change over time, which is

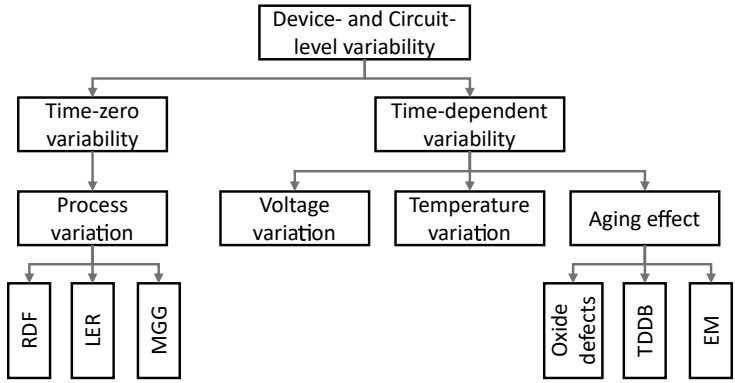


Figure 2.1: Taxonomy of the sources of device- and circuit-level variability. Only the main sources are enumerated and the miscellaneous sources are left out for clarity.

called *time-dependent variability*. The variability can be influenced by the workload, environmental conditions, or the stochastic nature of devices.

Figure 2.1 shows the taxonomy of the main sources of variability. Time-zero variability is attributed to process variation. And for time-dependent variability, the main sources are voltage variation, temperature variation, and aging effect. In the next few subsections, we elaborate on the main sources of device- and circuit-level variability.

2.1.1 Process Variation

A typical VLSI chip contains billions of transistors which all have to operate correctly. Among all these transistors, no two devices are identical, even at the time they are just manufactured. The differences mainly come from process variation, which is the variations originated from nonidealities in process control or the stochastic nature of matter. Composed of these variable devices, the final products are consequently nonuniform in speed, power, and any other quality metrics, which directly affects the yield. Therefore, process variation has become one of the main design challenges in the semiconductor industry, since it is directly linked to the design and manufacturing, and determines to some extent the final cost of the products.

In the following, we discuss some of the most prominent sources of process variations:

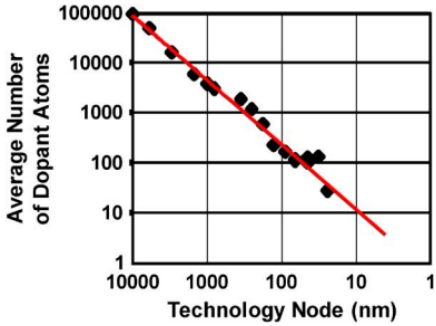


Figure 2.2: The trend of the average number of dopant atoms reduced with technology nodes. Replotted from [3]

- **Random Dopant Fluctuation**: Random Dopant Fluctuation (RDF) is the difference in the number and the distribution of dopants in the channel of devices, due to the stochastic process of implant ionization [36]. It arises as a prominent variation source as a consequence of the downscaling of transistor dimensions, in that the average number of dopant atoms decreases as a function of technology, as shown in Figure 2.2 [3]. In the advanced technology nodes, the average dopants per transistor can be fewer than 100, resulting in high randomness in the positions and number of dopants. Consequently, there will be significant variability in many important device parameters, such as threshold voltage (V_{th}), subthreshold slope (SS), cut-off frequency (f_t), and gate capacitance (C_g). In V_{th} variability, RDF is one of the major contributors [37].

  By atomistic simulations, an empirical relationship for planner devices with channel length below 100nm and dopant concentration above $10^{18} cm^{-3}$ is as below [4]:

$$\sigma V_{th} = 3.18 \times 10^{-8} \frac{T_{OX} N_{0.4}^{tot}}{\sqrt{L_{eff} W_{eff}}} \tag{2.1}$$

  in which $T_{OX}, N_{tot}, L_{eff}, W_{eff}$ represent the oxide thickness, total doping concentration, effective length, and effective width respectively. The relationship shows a polynomial dependence of V_{th} variability on the total doping concentration.

- **Line Edge Roughness**: Line Edge Roughness (LER) is the roughness and thickness variations of patterned structures, which subsequently result

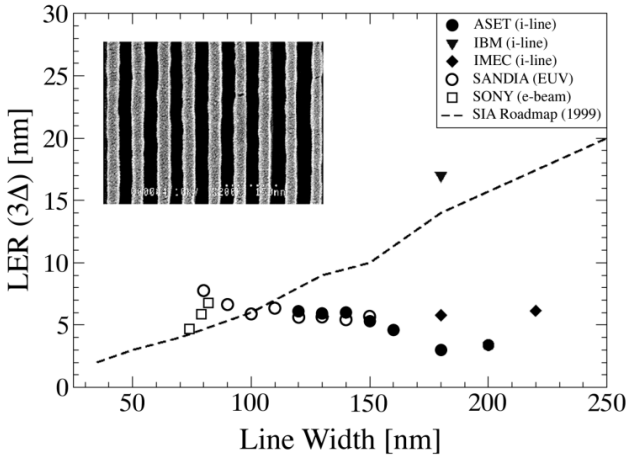


Figure 2.3: LER measurements for advanced lithography processes reported by various labs. The dashed line indicates the required levels in Semiconductor Industry Association (SIA) roadmap. The inset shows LER observed in sub-100nm e-beam generated lines. Replotted from [4].

in randomly varied device dimensions such as gate length and width. It is caused by the limitations and nonidealities of both the sub-wavelength lithography and etching process [36]. LER consequently induces both the increase of the leakage current and the variation of V_{th}.

The amount of LER is typically expressed by its 3σ value [4], where σ represents the standard deviation of the distances from the average edge to the actual edge at different local positions. Measurements of 3σ values from various labs are shown in Figure 2.3 [4], which indicates that LER is not scaled proportionally with the downscaling of line widths by the advance of lithography processes. This results in an increase of V_{th} variability by LER in advanced technology nodes.

- **Metal Grain Granularity**: Metal Grain Granularity (MGG) is the variability in grain boundaries of transistor gates, which are made of metal or polysilicon [36, 38]. Depending on the process conditions and the randomness in the process of crystalization, the grains can have different sizes and crystal directions, resulting in work function variation (WFV) [39] in the gate area. Consequently, local V_{th} conditions are created and the aggregated V_{th} varies from device to device [40].

  MGG-induced V_{th} variability scales according to the *ratio of the average grain size to the gate area* (RGG) [6]. Figure 2.4 shows the relations of

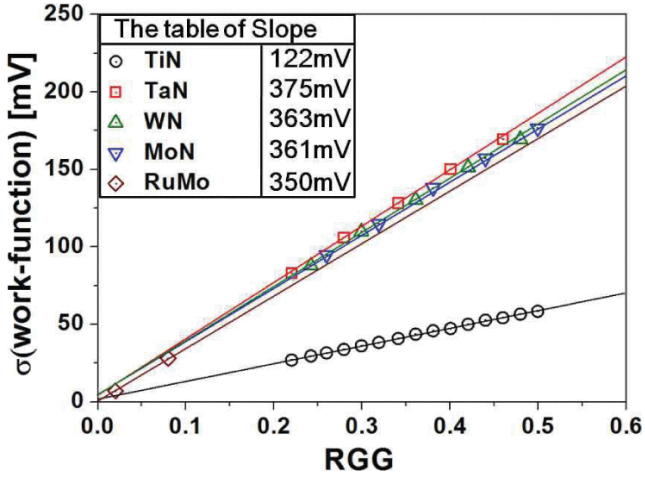


Figure 2.4: Relations of WFV and RGG for various gate materials (TiN, TaN, WN, MoN, and RuMo [5]). Replotted from [6].

WFV and RGG for various gate materials [6] (TiN, TaN, WN, MoN, and RuMo [5]). With the same gate material and grain size, the WFV is increased with the shrinking of the gate area. This trend indicates that the V_{th} variability by MGG is increased with the gate area reduction by technology scaling.

2.1.2 Voltage Variation

While devices and circuits are intended to function at a stable supply voltage level, they actually suffer from voltage variations occasionally during operation. The main sources of voltage variations include the di/dt noise and the IR drop [7], which originate from parasitic inductances and resistances in the power distribution network respectively.

The variation in supply voltage changes the switching current of devices, consequently impacting the propagation delay of logic gates and the speed of the entire circuit. Typically, the relation of the propagation delay (t_{pd}), supply voltage (V_{dd}), and threshold voltage (V_{th}) follows the following equation, where α is the velocity saturation index typically ranging from 1 to 2 [41, 42]:

$$t_{pd} \propto \frac{V_{dd}}{(V_{dd} - V_{th})^{\alpha}} \tag{2.2}$$

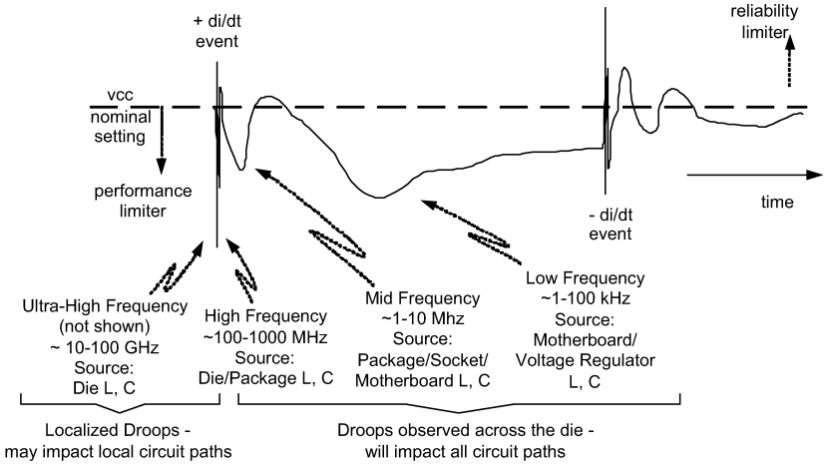


Figure 2.5: Illustration of a voltage droop occurrence, including all its frequency components. Replotted from [7]

In prior work, 10% change in V_{dd} is estimated to cause 30% change in t_{pd} in the 130 nm technology, while 1% change in V_{dd} causing 4% t_{pd} in 90 nm technology [43,44]. Thus, a voltage decrease can significantly enlarge propagation delays, causing logic errors. If frequency scaling is applied, the clock frequency can be changed at run time to avoid errors. Generally, the maximum achievable clock frequency (f_{max}) of a digital circuit is inversely proportional to the delay of the critical path. Therefore, it follows the equation below [45]:

$$f_{max} \propto \frac{(V_{dd} - V_{th})^{\alpha}}{V_{dd}} \tag{2.3}$$

Thus, when the voltage is decreased, the clock frequency needs to be reduced, which results in performance variability.

A particularly challenging type of voltage variation is the *voltage droops* [46]. Generated by a sudden increase in the switching activity of the circuit, an abrupt high current is induced to the power delivery network, causing a large IR drop. As shown in Figure 2.5, a voltage droop contains a wide range of frequency components, in which the high-frequency component occurs in only a few nanoseconds. It is reported that the effect of voltage droops can reach more than 10% voltage reduction and 15% delay increment [46], resulting in logic errors. Due to its nanosecond-level timescale, voltage droops are required to be handled within only a few clock cycles, posing a severe challenge in reliability.

2.1.3 Temperature Variation

As the number of transistors increases, so does the power density, resulting in a rise in chip temperature. If not managed properly, the aggregated power consumption can exceed the thermal design budget (TDP), leading to an overly high temperature which causes chip breakdown. Hence, thermal management and heat dissipation techniques come to the rescue, ensuring the chip's temperature under the thermal constraint.

Besides the average temperature, chips also suffer from uneven distribution of temperatures, which depends on the environment, workload, and heat transportation. Unbalanced power consumption can create local *hotspots*, resulting in performance variation across the chip, such as nonuniform speed among the cores in a multi-core processor.

Temperature fluctuation also influences circuit performance. On one hand, device performance has a dependency on temperature. On the other hand, increased temperature leads to increased metal resistance, lengthening the propagation delay of interconnects. Typically, the temperature fluctuates at a slower timescale of the hundreds of microsecond level [47], due to the relatively large time constant of heat transportation. Therefore, it can be mitigated by a wide variety of techniques, such as DVFS [48], task mapping [49], and fan speed control [50].

2.1.4 Aging Effect

Apart from the variation of environmental conditions (e.g. voltage, temperature), chips also suffer from temporary or permanent changes in the structures of devices and interconnects over time, known as aging effects. These effects cause both short-term and long-term changes in multiple parameters of devices and circuits, leading to performance degradation and shortening the chip's lifetime. Moreover, high voltage and high temperature accelerate many aging effects, further exacerbating the problems. In the following, we elaborate on some of the main aging mechanisms.

- **Oxide defects**: The imperfection in the gate oxide lattice and the oxide-channel interface can contain defects which can capture and emit individual charges. Each defect can be characterized by its position, charge capture time (τ_c), and charge emission time (τ_e) [8, 29]. Depending on these properties, the charging and discharging of individual gate oxide defects cause a handful of effects (RTN, BTI, HCI), leading to the change of multiple device parameters over time.

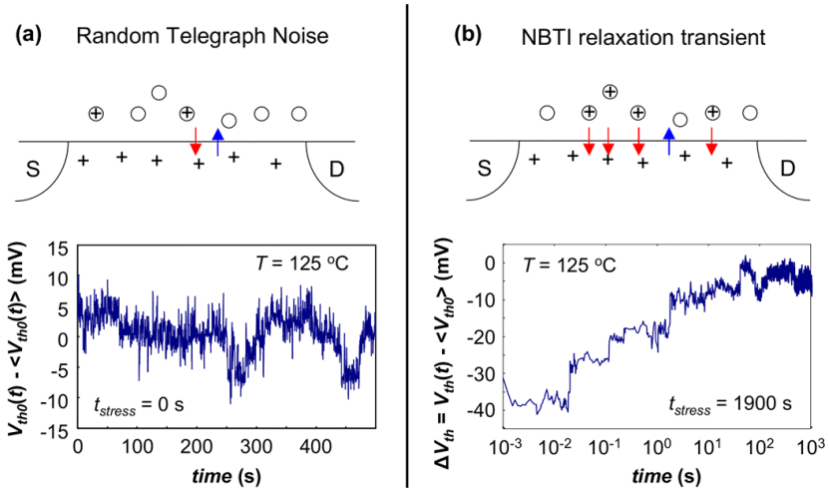


Figure 2.6: The mechanisms of RTN and BTI. Replotted from [8]. (a) The system is in dynamic equilibrium, manifested by Random Telegraph Noise (RTN) in small devices. (b) With a gate voltage stress, the system is approaching the dynamic equilibrium with the gradual discharging of excess charged oxide defects, known as BTI.

Stochastic charging and discharging of oxide defects of short time constants (τ_c, τ_e) are manifested by *Random Telegraph Noise* (RTN), which is the phenomenon of high-frequency random V_{th} fluctuation [51]. RTN occurrs when the system is in dynamic equilibrium, as shown in Figure 2.6 (a) [8]. It induces delay fluctuation of logic gates, which can reach 10% in a measurement of ring oscillator [52].

In contrast, *Bias Temperature Instability* (BTI) is the directional V_{th} degradation and recovery over time, attributing to the charging and discharging of oxide defects of both short and long time constants in the non-equilibrium state, as shown in Figure 2.6 (b). It can be further classified into Negative Bias Temperature Instability (NBTI) and Positive Bias Temperature Instability (PBTI) when occurring on PMOS and NMOS respectively [53]. Typically, The long-term trend of BTI tends to gradually shift toward increased V_{th} and lowered carrier mobility. As a result, the chip's lifetime can be shortened due to BTI effects.

Hot Carrier Injection (HCI) is also caused by oxide charge trapping, but the mechanism is widely different. HCI originates from the high lateral electric field in the channel, normally due to high drain-source bias (V_{DS}). Carriers accelerate in the field and obtain high kinetic energy.

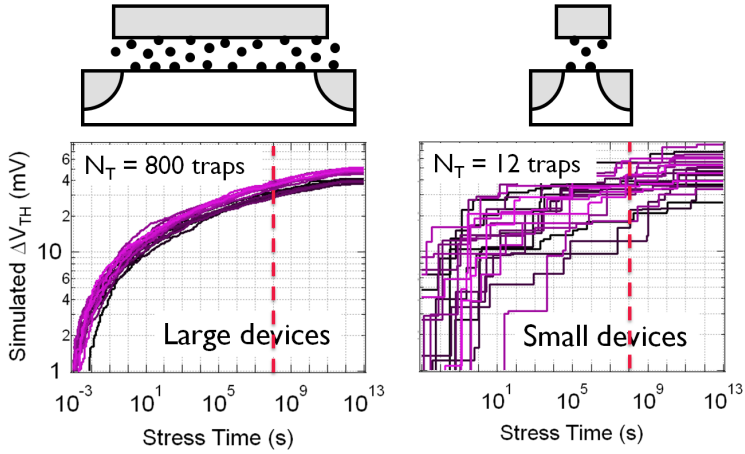


Figure 2.7: The downscaling of devices reduces the number of defects to only a handful. As a result, the stochastic properties become prominent, resulting in large variations in device performance and lifetime. Replotted from [9].

These *hot carriers* can inject directly into the gate oxide, or create high energy electrons and holes due to scattering, which in turn can also be injected into the oxide [54–56]. In consequence, changes accumulate in the gate oxide near the drain region, while the untrapped carriers contribute to gate leakage current and substrate leakage current. Similar to BTI, trapped changes caused by HCI result in increased V_{th} and reduced carrier mobility [57].

With the downscaling of the devices, the number of defects is reduced to only a handful. As a result, the stochastic properties become prominent, resulting in large variations in device performance and lifetime [9]. Figure 2.7 shows an example of the scaling effects of BTI. The V_{th} variability transforms from gradual changes as a combined effect of hundreds of effects, to a combination of individual discrete steps, each corresponding to a single charging/discharging event. The V_{th} variability also becomes dispersed among devices. Moreover, the effects of RTN, BTI, and HCI are highly dependent on voltage, temperature, and workload, which further complicates the estimation of the chip's performance degradation and lifetime.

- **Time-Dependent Dielectric Breakdown**: Time-Dependent Dielectric Breakdown (TDDB) is the phenomenon that damages accumulated in

insulators induce leakage current and eventually cause the loss of insulating property of the material [58]. It occurs in both the gate oxide of the devices and the low-k dielectrics of the interconnects [59].

In the gate oxide, The injected charges can lead to trap-assisted leaking paths, increasing the gate leakage current steadily. This phenomenon is called Stress Induced Leakage Current (SILC) [60]. Afterwards, as trapped minority carriers are gradually built up in the damages of the gate oxide, conduction paths can be formed between the gate and the substrate when the vertical placement of the oxide changes are aligned. At this moment, the channel current is suddenly cut down, ending up with intermittent or permanent faults. This effect is called a hard breakdown, which poses a threat to the chip's lifetime. The gate oxide breakdown is influenced by supply voltage, device temperature, oxide thickness, and oxide materials [61,62]. With the thinning down of gate oxide in advanced technologies, this effect is also exacerbated and presents as a serious reliability problem.

Similar mechanisms also occur in the dielectric of the interconnects. Atoms of the conductors (e.g. copper) can infiltrate into the dielectric and create conductive paths between wires, leading to leakage current and thus increasing wire delay. Eventually, the hard breakdown occurs and the insulating property is lost. This process can be accelerated by high temperature and high electric field. Some features in advanced technologies further aggravate the effect. First, the reduction of interconnect width and thickness increases the current density, while the smaller pitch increases the electric field. Second, the weaker breakdown strength of the porous, low-k dielectrics also enhances the effect [59].

- **Electromigration**: Electromigration (EM) is the phenomenon of undesired displacement of the conductor's atoms in the interconnects caused by electron flow [63–65]. As the electrons move through the interconnects, they can collide with the atoms of the conductor. As a result, the atoms are displaced, leaving voids in their original positions and creating extrusions in the regions where these atoms drift to. Both the voids and the extrusions induce variation in resistance, which affects the chip's performance. If the dispaced atoms further accumulate, the voids can cause open circuits, and the extrusions can cause short circuits between neighbouring interconnects. In consequence, the chip could break down. With the shrinking of interconnects enabled by advanced lithography, the current density is increased, which enhances the EM effect. Therefore, EM becomes a main problem of the reliability of interconnects.

2.2 Architecture-Level Performance Variability

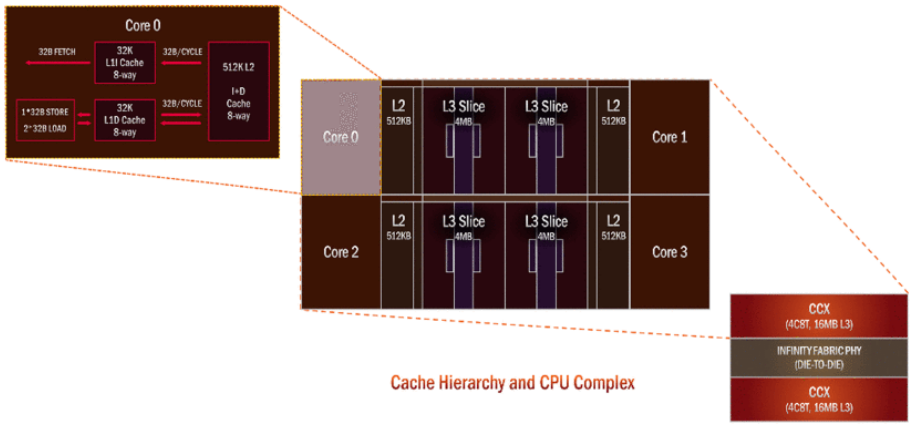
The ever-growing number of transistors has fueled the innovation and sophistication of the processor architecture, improving the throughput and versatility of processors. In a single processor core, the single-thread performance is enhanced by various microarchitectural features such as pipelining, out-of-order execution, speculative execution, and instruction-level parallelism. Multi-core processors enable simultaneous executions of multiple threads in different cores, further boosting the throughput. Furthermore, system-on-chip (SoC) architecture integrates various functional units (e.g. GPU) in one chip, making processors more versatile [66]. Recently, the development of heterogeneous multi-core architectures and domain-specific accelerators (DMA) further improves the throughput and energy efficiency in specialized domains, such as artificial intelligence (AI) [67, 68]. Apart from the computation cores, memory hierarchy minimizes the average latency of data access, and interconnection networks facilitate data transfers among all units.

To illustrate the complexity of modern processor chips, Figure 2.8 shows the block diagrams of ARM Zen 2 "Matisse" SoC architecture [10], a general-purpose processor designed for PC. It contains 16 processor cores, three levels of caches (either private or shared), interconnection networks, and several miscellaneous functional units (e.g. memory controller, IO controller).

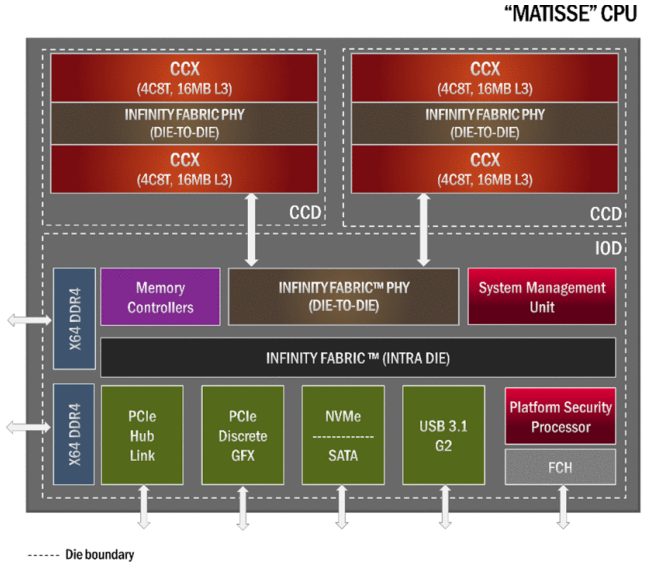
With so many units and convoluted interactions, it is formidable to analyze the exact execution time of a workload. At the microarchitecture level, each unit has its own features of maximizing average throughput by exploiting dynamism, leading to variable latency [32]. At the macroarchitecture level, multiple units can contend for the same shared resources (e.g. caches, communication networks), so that one unit could delay the execution of another one [32, 69]. All of these architectural features contribute to performance variability, which is problematic for ensuring timing guarantees.

Branch prediction is a typical example of an architectural feature which causes performance variability. When a branch instruction is encountered in a processor core, the next instructions to be executed depend on the branch outcome, which would only be available when the branch instruction is resolved. Waiting for the branch outcome would cause unfilled pipelines, resulting in several cycles of latency. Since branches are very common in programs, these latencies could accumulate to a significant throughput loss. Hence, predicting the branch outcome and speculatively executing the instructions of the predicted path can reduce these latencies. If the branch predictor is accurate, the latencies can be mostly removed and the throughput can be substantially boosted.

To illustrate the performance variability generated by branch prediction, Figure



(a)



(b)

Figure 2.8: Block diagrams of ARM Zen 2 "Matisse" SoC architecture. Replotted from [10]. (a) the architecture of a "core complex" (CCX). (b) the architecture of the whole SoC, composed of four CCXs. Infinity Fabric is a proprietary interconnection architecture which facilitates data transfers among linked components.

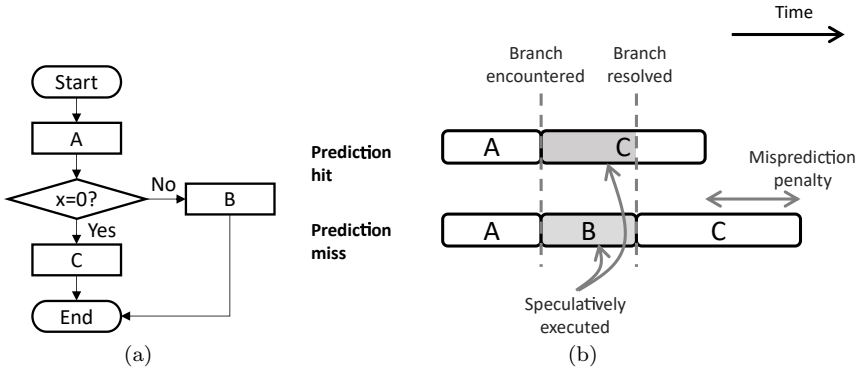
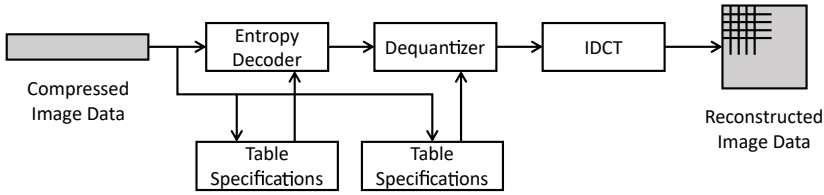


Figure 2.9: Illustrative example of performance variability generated by branch prediction (a) control flow of a task containing one branch (b) time diagram of the processor in cases of either branch prediction hit or miss.

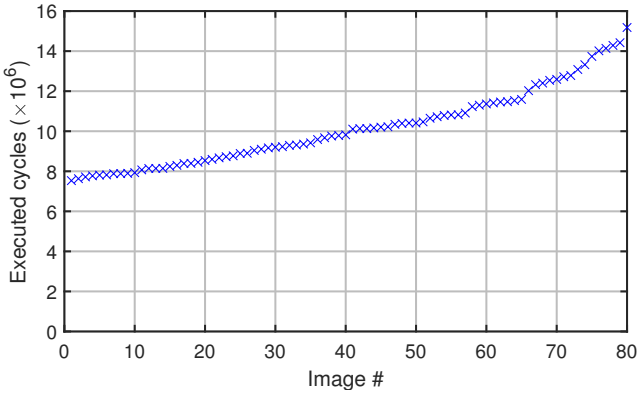
2.9 shows a simple example of a program containing one branch. If the branch prediction hits, the speculative execution outcomes become valid and no time is wasted. On the contrary, if the branch prediction misses, the speculative execution outcomes have to be discarded. Thus, the time spent in the wrong branch directions is wasted, which is called the branch misprediction penalty. Evidently, whether branch prediction hits or misses will affect the execution time of the program. However, in order to improve prediction accuracy, the policy of branch predictors is complicated, often involving multiple strategies which can be dynamically switched depending on the application context and the processor states [70]. Therefore, it is difficult to completely characterize the behaviours of the branch predictors in advance, resulting in uncertainty in execution time.

2.3 Software-Level Performance Variability

Performance variability is manifested in the OS, middleware, and applications in many manners. As the OS schedule time slots and allocate hardware resources among multiple processes, the execution time of an application will be highly dependent on other co-existing processes and the arbitration policy of the OS. In addition, the OS can interrupt the application with system administration tasks (e.g. system health monitoring), resulting in the increase and uncertainty in the execution time of the application [69].



(a)



(b)

Figure 2.10: Performance variability of a JPEG decoder application [11] (a) block diagram of the JPEG decoder. Replotted from [12] (b) execution cycles of the JPEC decoder, benchmarked with different input images of the same size on a RISC-V ISA simulator, assuming $IPC = 1$ [13]

During the execution of the application, different execution paths of the control flow can be taken according to the input data, leading to widely dispersed execution times. For example, the loop bound of a loop can be dependent on a certain input variable. Similar to the dynamism in the architecture, dynamic control flows are extensively exploited to improve performance and realize various functions, making the execution time difficult to analyze and predict.

As a representative example, Figure 2.10b shows the execution cycles of a JPEG decoder application [11], benchmarked with different input images of the same size on a RISC-V ISA simulator [13], assuming instruction per cycle (IPC) equals one. In this simulation, the highest execution time is approximately 2× of the lowest one, exhibiting substantial performance variability. Within the program flow of JPEG decoding (shown in Figure 2.10a [12]), the main source of performance variability comes from the entropy decoder block, in which

Huffman decoding is performed based on the Huffman tables and the encoded bit string in the input data. Hence, the computation required for decoding is highly dependent on the input data.

2.4 Approaches to Tackling Variability

Many approaches have been developed to alleviate the undesired effects of variability. These approaches are instantiated on similar principles and tailored for different kinds of variability. In the following, we introduce some of the most important types of approaches to tackling variability.

2.4.1 Guardbanding

Guardbanding is the most straightforward method to deal with variability. In guardbanding, an upper bound of the variability among all possible conditions is calculated, which is called the *guardband* or *margin* [71]. In design time, the chips are designed to function correctly within the specifications (i.e. design corners), which are defined as the nominal condition plus the guardband, so that the chip will not fail even in the circumstance of maximum variability. The guardband can be defined in different ways. Analytically, the guardband can be calculated from the worst-case scenario analysis, or defined as certain quantiles (normally 3σ or 6σ) of the presumed statistical distribution of variability. Practically, it can also be determined according to the worst value from empirical observations, or simply defined as an arbitrary value from industrial conventions, typically 10%.

The use of guardbanding is prevalent in every variability field. For example, the corner-based design paradigm requires the chips to work under the circumstance that all device parameters are deviated from their nominal values by 3σ in process variation [72]. To account for voltage variation, chips are designed to work even when the supply voltage is 10% deviated from its nominal level. For the BTI effect, a guardband of V_{th} is obtained assuming devices are under the DC stress at 125°C over 3 years [73]. In tackling performance variability, WCET analysis is also a manifestation of guardbanding [31,32].

Guardbanding is conventionally the main approach, if not the sole approach, to dealing with variability. However, it is no longer sufficient for the great magnitude and complexity of variability in modern chips. With the increasing amount of every variability source, the guardband can be greatly enlarged to $2\times$ or more, significantly degrading the chip's quality or even making the chip's

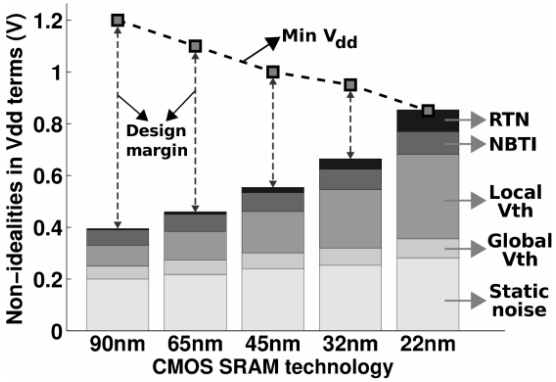


Figure 2.11: Impact of each non-ideality (quantified in V_{dd} terms) on SRAM supply voltage guardband under different CMOS technologies. The total guardband would reach the same level as the minimum V_{dd} with the technology downscaling, limiting the voltage scaling. Replotted from [14]

design infeasible. Figure 2.11 instantiates the failing of guardbanding in SRAM design, in which the guardbands for the worst case of each variability source sum up to an enormous voltage margin, which invalidates the scaling of V_{dd} in advanced technology [14].

The necessity for large guardbands is also questionable. In fact, guardbanding can lead to *over-margining*, which means the assumed worst-case scenario is too conservative that it would never happen in the real world. Therefore, people have been working on more sophisticated and accurate variability models, striving to eliminate the pessimism in the guardbands as much as possible. The results are deemed to be limited since many variability mechanisms are dependent on the run-time environment and workload, which is agnostic at design time.

2.4.2 Run-Time Mitigation

Since we can only know the actual amount of variability at run time, it is sensible to deal with variability at run time as well. Therefore, run-time mitigation arises from the idea to make the chips observe and mitigate the variability continuously [15], operating on similar principles of adaptive control systems.

Figure 2.12 shows the block diagram of a run-time mitigation system, in which monitors are distributed across widespread positions in the processor to monitor the amount of local variability in real time. The types of monitors are diverse,

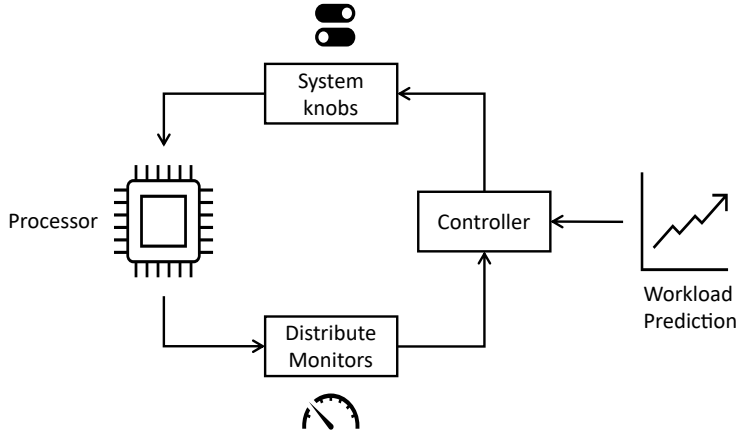


Figure 2.12: Block diagram of a run-time mitigation system. Replotted from [15]

depending on the type of variability to observe. For example, voltage sensors and temperature sensors are deployed to directly detect on-chip voltage and temperature variations. In contrast, in-situ critical path monitors and canary circuits are used to detect performance fluctuation of the circuit, which implies the variations in voltage, temperature, and aging [47, 74–76].

The real-time variability information is read by the run-time controller, which runs an adaptive control algorithm to determine the system's reaction to the variability. The algorithm makes decisions based on multiple factors, including the variability and the workload of the past, present, and predicted future. The controller is usually a small routine process in the OS or the firmware, while many modern processors are also equipped with a dedicated microcontroller to run the controller.

Processors are equipped with system knobs, which can dynamically change the system parameters. For example, configurable clock generators and voltage converters allow the clock frequency and supply voltage to be altered at run time. Based on the decision of the controller, the system knobs can be switched, by which system parameters are dynamically changed to mitigate variability.

On the principle of run-time mitigation, many variability management approaches have been developed and widely used to effectively deal with different kinds of variability. To prevent voltage droop, many voltage droop mitigation approaches have been developed and applied in modern integrated systems [46, 74, 77–84]. As for temperature fluctuation, dynamic thermal management techniques are used to maintain chip temperature within operating

region [85–90]. In terms of lifetime, dynamic reliability management techniques are developed to mitigate long-term aging effects and prolong the lifetime of the chips [91–95]. However, since the effects of these approaches are normally validated empirically, they cannot provide hard guarantees of their effectiveness. To ensure timing guarantees, mitigation approaches for performance variability are also developed, which will be discussed in detail in the next chapter.

2.5 Summary

This chapter gives a comprehensive overview of the variability sources in each abstraction layer of integrated circuits. Firstly, the device- and circuit-level variability sources are enumerated, including process, voltage, and temperature variation, as well as several aging effects such as oxide defects, TDDB, and EM. These variations affect multiple device parameters (V_{th}, leakage current, etc.), circuit performance, and chip's lifetime. With the downscaling of the device dimensions, the magnitudes of these variations are mostly enlarged. Secondly, architecture-level performance variability originates from the architectural features of dynamic execution(e.g. branch prediction) and the contention of shared resources (e.g. caches). These features introduce uncertainty and unpredictability in execution time. With the increasing integration and complexity of processor architectures, the variability is exacerbated. Finally, software-level performance variability can be attributed to the resource allocation of the OS and the dynamic control flows of the applications, making the execution time difficult to analyze and predict.

To cope with variability, guardbanding is widely used to ensure the reliability of circuit design under the circumstance of the worst-case variability. However, large guardbands would degrade the quality of circuit design, and with the increasing variability, this problem is not acceptable anymore. Therefore, run-time mitigation solves the problem by allowing the chip to mitigate and adapt to the variability at run time. Based on this principle, many approaches have been developed to effectively deal with different types of variability such as voltage, temperature, aging, and performance variability.

Chapter 3

Mitigation Approaches for Performance Variability: State-of-the-Art

In this chapter, we elaborate on the state-of-the-art run-time mitigation approaches for tackling performance variability. In principle, these approaches aim to run workloads with just enough processor speed and hardware resources that the tasks could finish just in time. In this way, the minimum power and energy consumption can be reached to avoid thermal problems, reduce energy costs, and prolong the duration of battery-charged devices.

This chapter is structured as follows: first, we describe the general structure of mitigation approaches for performance variability in Section 3.1. Then, in Section 3.2 we discuss several typical system knobs used in performance variability mitigation. Finally, Section 3.3 discusses the heart of the mitigation approaches, the adaptive control algorithm for scheduling with and without timing guarantees.

3.1 Structure of Mitigation Approaches for Performance Variability

The mitigation approaches for performance variability follow the same principle of adaptive control systems, as discussed in Section 2.4.2, while the

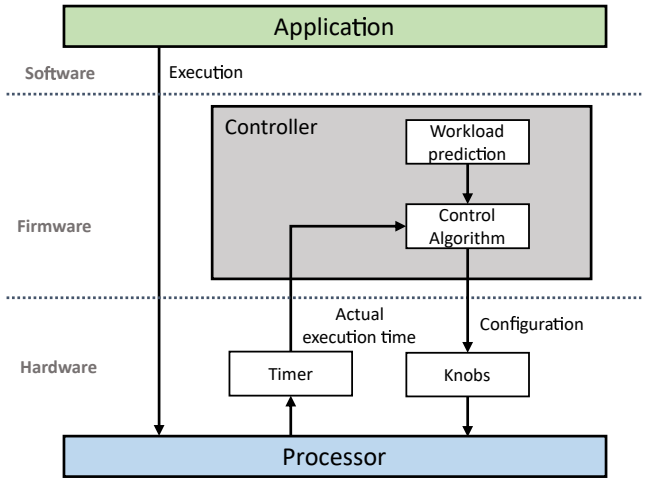


Figure 3.1: Block diagram of the mitigation approach for performance variability

implementation is tailored for tackling performance variability. Figure 3.1 shows the block diagram of the performance variability mitigation system. The timer acts as the monitor of the system, measuring the execution time of each part of the application. At the beginning of the application, the run-time controller formulates an optimal schedule of knob configurations over time, according to the timing guarantee requirements and the predictions of future workload. During the execution, the run-time controller reads the timing information in the timer to calculate the amount of performance variability, by comparing the actual execution time with the predicted execution time. Depending on the amount of performance variability at run time, the controller makes rolling decisions on adjusting the schedule and the knob configurations. By switching the knobs, the processor speed can be changed to offset the performance variability. In general, if the application progress lags behind the schedule, the controller can increase processor speed to catch up with the schedule. On the contrary, if the application progress is ahead of the schedule, the controller can slow down the processor to save energy.

3.2 System Knobs

The system knobs used in performance variability mitigation should be able to change the execution time of applications, either by adjusting the processor speed or by reallocating hardware resources to the applications. Depending

DVFS level	V_{dd} (V)	f (GHz)	DVFS level	V_{dd} (V)	f (GHz)
Level 1	1.30	3.074	Level 4	1.15	2.281
Level 2	1.25	2.852	Level 5	1.10	1.932
Level 3	1.20	2.588	Level 6	1.05	1.540

Table 3.1: DVFS modes of the Intel Core2 Duo E6850 processor [25]

on the application and the platform, the effect of system knobs can vary. For example, allocating different numbers of processor cores can substantially change the execution time of a parallelizable task, while a mostly sequential task is hardly affected.

These knobs are also different in switching time, which is the interval of changing the knob from one mode to another mode. Apparently, the switching time is essentially another source of performance variability which increases the execution time. When speeding up the processor, the new mode needs to take effect for a long enough time, so that the reduction of execution time can offset the original variability plus the switching time. Also, the switching time needs to be smaller than the timescale of the variability, in order not to significantly increase the total variability. Therefore, the timescale of the variability that a knob can handle is limited by its switching time.

The time granularity of the mitigation approaches is defined as the interval between two consecutive switchings of the knob. In general, the time granularity is much longer than the switching time, so the execution time overhead of knob switching is much smaller than the original workload. State-of-the-art approaches can operate down to the millisecond-level time granularity, limited by the switching time of dynamic voltage and frequency scaling (DVFS). To mitigate variability in more fine-grained time granularity, knobs with smaller switching time are required.

In the following, we discuss some of the most important knobs used in performance variability mitigation.

3.2.1 Dynamic Voltage and Frequency Scaling (DVFS)

Dynamic voltage and frequency scaling (DVFS) enables processor cores to dynamically change their supply voltage and clock frequency to vary their speed, power, and energy consumption [45]. The hardware setups of DVFS require a programmable clock generator, normally realized by a phase-locked loop (PLL), and a programmable voltage regulator, normally realized by a buck-type switching-mode DC-DC converter [25]. Since it is easy to use and provides a

wide range of modes with substantial speed-power trade-offs, DVFS draws high popularity and is implemented in most modern processors. Table 3.1 shows a typical DVFS example in the Intel Core2 Duo processor, which contains six different voltage and frequency modes [25].

The speed-power variation of DVFS comes from the relation between dynamic power (P_{dyn}), supply voltage (V_{dd}), and clock frequency (f) of the circuit as below:

$$P_{dyn} = aCV_{dd}^2 f \tag{3.1}$$

where a is a coefficient reflecting the switching activity, and C is the equivalent capacitance of the circuit. Assuming the number of execution cycles (N_c) for a workload is constant regardless of the clock frequency, then the execution time (T) can be calculated as N_c/f. In this case, the dynamic-power-induced energy consumption (E_{dyn}) will be equal to $P_{dyn} \times T$, resulting in the following relation:

$$E_{dyn} = N_c aCV_{dd}^2 \tag{3.2}$$

The relation between voltage and frequency of DVFS generally follows Equation 2.3. Based on this relation and Equation 3.1 & 3.2, we can derive that P_{dyn} is quadratic to cubic with respect to V_{dd}, and E_{dyn} is quadratic with respect to V_{dd}. Since dynamic power is the dominant component of total power consumption in modern processor cores, a small shift in V_{dd} will bring substantial changes in clock frequency, total power, and energy. This property makes DVFS well-suited for providing speed and power variations in processors.

Switching between DVFS modes requires time for the voltage and frequency to be stabilized at the new levels. When upscaling, the processor first configures the voltage regulator to increase the voltage to the new level, and then the PLL is configured to increase the frequency to the new level. When downscaling, the order is reversed in which the frequency is decreased first, and then the voltage is decreased. The switching time adds up to about 10–150 microseconds, depending on the DVFS modes before and after the transition [25]. While DVFS typically has the shortest switching time of all speed-energy-tradeoff knobs in the processors, it limits the time granularity of performance variability mitigation to the millisecond level, since the mitigation needs to operate in larger time granularity than the switching time.

Another drawback of DVFS resides in the incomplete freedom in selecting DVFS modes. In a multi-core processor, multiple cores can share the same voltage

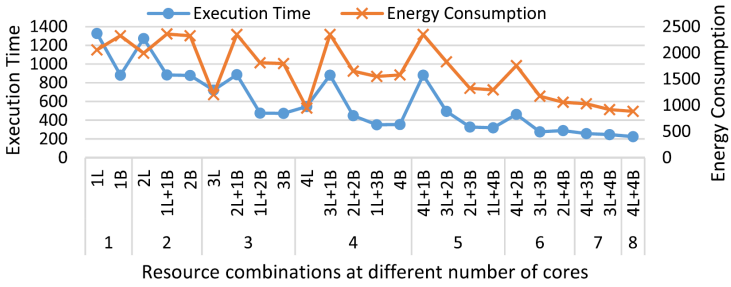


Figure 3.2: Variability in execution time (seconds) and energy consumption (J) when running the Blackscholes application (from PARSEC benchmark [16]) using different core combinations on ARM's big.LITTLE architecture which contains four big (B) and four LITTLE (L) cores. Replotted from [17]

domain, since the number of voltage domains in one chip is restricted by the number of package bumps. For correct operation, the shared voltage supply of the domain need to cater to the core with the highest voltage demand, even if the other cores request lower voltage levels for power/energy saving [96]. On the other hand, the availability of high-voltage modes is also limited by the thermal constraint. It is normally disallowed for most of the cores to use high-voltage modes simultaneously. Otherwise, the massive collective power density can break down the chips [67]. These limitations result in the reduction and uncertainty in the performance power, and energy benefits of DVFS.

3.2.2 Task Mapping and task migration

Heterogeneous multi-core architectures enable great flexibility in the type and number of processor cores used by an application. By distributing workloads on multiple cores, parallelizable applications can be much accelerated, governed by Amdahl's law [97]. In addition, heterogeneous processor cores provide even wider options of the speed, power, and energy levels [98,99]. Figure 3.2 shows an example of variability in execution time and energy consumption when running the same application with different core combinations [17]. With such a wide range of options to run every single task, the mapping of tasks to hardware resources has profound influences on the performance and energy efficiency of all tasks [100]. Furthermore, task mapping can be combined with DVFS to provide even more options to fine-tune the performance and energy efficiency of tasks.

Before the tasks start, the initial *task mapping* is solved to find the optimal task-to-core mapping in consideration of multiple objectives (performance, deadline constraints, energy, lifetime, etc.). During execution, *task migration* can be performed to reallocate the resources of the tasks, in order to react to run-time events, such as the entering of a new application [100]. To migrate a task, the controller first suspends the task and initiates a new process on the destination core. Then, the whole task context (codes, data, stack, register file contents, processor states, etc.) is transferred from the source core to the destination cores. Finally, the task is continued on the destination core, and the data on the source core are purged [101,102]. The switching time varies with the task context and the communication channels between the cores, typically up to tens of milliseconds [99].

The use of task mapping/migration also exhibits several important limitations. First, the millisecond-level switching time is much longer than DVFS, limiting the frequency of task migration. Also, considering the chip area costs, heterogeneous cores are normally only two or three types, which form very coarse-grained speed and power/energy levels. The acceleration enabled by multi-core also depends on the exploitable parallelism in the application. For a mostly sequential application, multi-core is ineffective.

Due to the above reasons, task mapping/migration is less flexible than DVFS, normally mitigating the performance variability of larger timescales. In general, task mapping/migration exploits the inherent performance and power/energy heterogeneity in macroarchitectures, so its coarse granularity is also attributed to the hefty sizes of macroarchitectures. To overcome these drawbacks, we suppose that it is possible to design fine-grained knobs by exploiting the heterogeneity in microarchitectures, such as heterogeneous datapaths (HDP) which we will show later in our work.

3.2.3 Gas Pedals

Gas pedals, also known as *turbo modes* or *computational sprinting*, are short-lived high-performance modes which are available only for a limited duration due to system restrictions [103]. A typical instantiation of this concept is the ultra-high voltage/frequency modes in DVFS, in which the high power dissipation could lead to thermal runaway if the modes run for an excessively long time. Gas pedals are implemented in some of the high-end commercial processors, such as Intel "Turbo Boost" technology [104,105] and AMD "Precision Boost Overdrive" technology.

When the processor switches to the gas pedal mode, the massive power dissipation gradually raises the chip temperature. Therefore, the controller

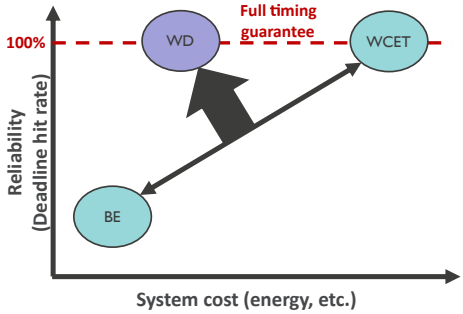


Figure 3.3: Comparison of reliability (deadline hit rate) and cost (energy, etc.) among different types of adaptive control algorithms: best-effort (BE), worst-case execution time based (WCET), and workload-dependent (WD).

needs to turn off the gas pedal in time, so the chip would not fail due to excessively high temperature. Due to the reliability concern, gas pedals must be used with caution. Normally, the controller only turns on the gas pedal reactively when highly intensive workloads enter, which require extremely fast execution to meet the timing requirements. However, the timing guarantees can still be lost if the gas pedal is used too late. To ensure timing guarantees, the proactive way is to use the gas pedal as a resource which can be dynamically scheduled when intensive workloads are predicted in the upcoming future.

3.3 Adaptive Control Algorithms

In the run-time controller, the adaptive control algorithm is intended for finding the optimal way to switch the knobs which can ensure timing guarantees while minimizing energy consumption. The algorithm normally runs in the firmware or the OS as a background process. It is required to be lightweight so that its execution time is short enough to quickly respond to performance variability. Therefore, it is not practical to exhaust all possible scenarios to derive the optimal solution. Some heuristics need to be applied.

Different adaptive control algorithms can be considered as the trade-off between reliability and system costs. In the context of performance variability mitigation, the concerned reliability and cost metrics are the deadline hit rate and the energy consumption respectively. The relation can be illustrated by the fact that in order to reach a high deadline hit rate, the algorithm needs to be more

conservative so that the high-speed modes are used more often, increasing energy consumption.

Figure 3.3 depicts the comparison of reliability and cost among different types of algorithms. Best-effort (BE) approaches are algorithms without timing guarantees, which have low energy overhead but inferior deadline hit rate. On the other hand, WCET-based approaches belong to algorithms with full timing guarantees, which means they can achieve 100% hit rate whenever it is possible. However, WCET-based approaches lead to high energy overhead. To reduce the system costs, workload-dependent (WD) approaches aim to improve the execution time prediction based on run-time scenarios of the workload. In this way, energy consumption can be improved while the full timing guarantee is preserved. In the following subsections, we further discuss the state-of-the-art for each type of algorithm in detail.

3.3.1 Algorithms without Timing Guarantees

Best-effort (BE) approaches are referred to as algorithms without timing guarantees, which aim to maintain stable performance of workloads with acceptable deadline miss rates while minimizing energy consumption. They cannot achieve the maximum deadline hit rate but can ensure guaranteed long-term throughput and near-optimal energy consumption. These approaches cannot reach the strict requirement of time-critical applications, so they are only feasible for general and time-sensitive applications.

The accuracy of workload prediction is crucial for the quality of BE approaches. If the actual execution time is not close to the prediction, the processor needs to dramatically speed up and down to compensate for the discrepancy in the workload progress generated by misprediction, resulting in extra energy consumption. Many heuristics are developed considering multiple factors including both design-time analysis and run-time observation in the past and present. In the following, we discuss several representative approaches:

- **PID-control**: The PID controller is widely used in various control system domains. On the same PID control principle, Rodopoulos et al. [106, 107] proposed a PID-controlled DVFS algorithm which regulates the performance of a workload in the presence of performance variability. It is easy to implement since it operates based on only one variable, the amount of performance variability at present. No prior knowledge of workload and no memory of past events are required. However, its effectiveness is also limited by its simple working principle, in that it can only reactively cope with performance variability after it occurs. Furthermore, its unawareness

of the content and the execution history of the workload also degrades the quality of its decisions. These drawbacks limit its performance in preventing deadline misses and saving energy.

- **History-based**: History-based approaches aims to predict the execution times of the upcoming jobs with the past jobs. For example, the predicted execution time of the next job can be the same as the last job [108], the weighted average of a couple of the most recent jobs [109], or the exponential moving average of the past jobs [110,111]. The controller can also keep a record of the execution time distribution of the past jobs, and then use the average [112] or a certain quantile [113] as the prediction. Apparently, the prediction accuracy of these heuristics depends on the similarity of the future workload with the past workload. An abrupt change in the behaviours of workload would not be expected by these heuristics. Hence, these heuristics are more suitable for periodic, static, or slow-changing workloads, rather than highly dynamic workloads.
- **Machine learning**: With the rapid development of machine learning in recent years, approaches based on machine learning are also created to improve workload prediction accuracy and scheduling quality [114,115]. The working principle of these approaches is to train a model which can translate the system states to the predicted processor performance, or the optimal knob setting for the next interval. The training can be performed at design time based on profiling data, or at run time based on online monitoring.

3.3.2 Algorithms with Timing Guarantees

To meet the ultimate timing requirements for time-critical applications, algorithms with full timing guarantees are developed to achieve 100% hit rate whenever it is possible, reducing energy consumption at the same time. Timing-sensitive and general applications are also benefited from these algorithms since they ensure a guaranteed quality for these applications.

Figure 3.4 depicts the working principle of this type of algorithm, supposing that the controller can look into a buffer with the information of the upcoming n jobs. For each job in the buffer, an upper bound of execution time can be calculated to represent the worst-case scenario of each job. At the start of the workload, an optimal schedule is formulated, which can ensure no deadline misses even under the worst-case scenario that all jobs end up with their upper-bound execution times. Then, Job 1 is executed under the modes assigned by the schedule. The actual execution time of Job 1 is usually smaller than the upper bound, so after Job 1 finishes, the actual progress of the workload is usually ahead of the

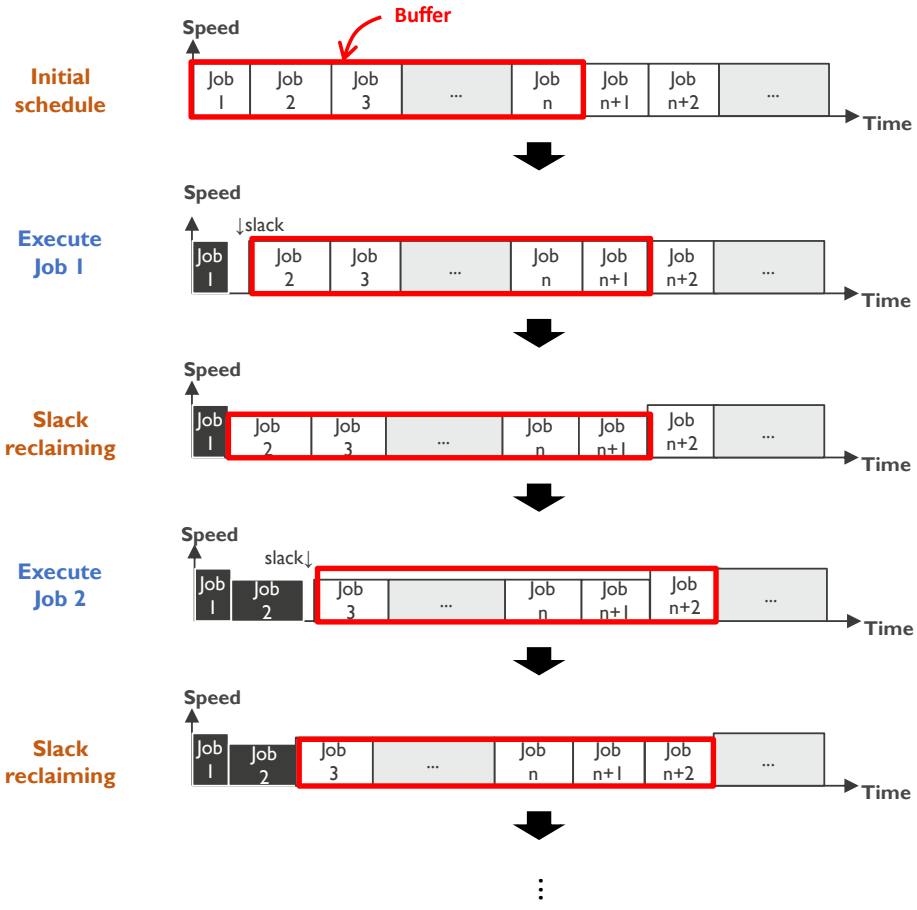


Figure 3.4: Scheduling example of algorithms with timing guarantees

schedule by an amount of time, which is called *slack*. The slack can be exploited by *slack reclaiming*, which means the schedule is refined by using the slack as an extra time resource to distribute it among the jobs in the buffer. Then, Job 2 is executed, which triggers another round of slack generation and reclaiming. With this repeated pattern, the controller keeps on rolling decisions until the workload ends.

To ensure timing guarantees, the key is to calculate the upper bound of execution time at run time. The upper bound is used to derive the criteria in which the deadlines will never be missed. The upper bounds should be as tight as possible so that the discrepancy between the upper bound and the actual execution time is minimal, preventing the algorithm from being over-conservative. In the calculation of upper bounds, WCET is used in most algorithms. However, WCET suffers from the problems of over-pessimism and over-simplification. Therefore, workload-dependent approaches have been proposed to improve the upper-bound calculation. In the following, we discuss the two approaches in detail.

Worst-case execution time (WCET) based approaches

Worst-case execution time (WCET) analysis has been developed to obtain the maximum execution time of an application in all circumstances. Most approaches with timing guarantees rely on WCET to provide an upper bound of execution time. A typical WCET-based approach is a combination of real-time scheduling as the controlling algorithm, DVFS as the system knob, and WCET as the execution time bounding method.

Many scheduling algorithms have been developed based on the WCET [33]. In the finest granularity, several works [116–119] have studied the optimal schedule of switching within a job which can minimize average energy consumption. However, these works usually overlook the time and energy overheads of switching, which makes these calculations impractical. For the scheduling of multiple tasks, the cycle-conserving algorithm of Pillai and Shin [120] marks out the basic principle of slack reclaiming in a fixed sequence of tasks. On this principle, real-time scheduling algorithms have been developed for both periodic [121–124] and aperiodic [125] tasks. Scordino and Lipari [126] further proposed GRUB-PA, a generalized algorithm for aperiodic, sporadic, and periodic tasks, in which the processor speed is dynamically set based on the total utilization of the tasks. This algorithm has been implemented in the Linux kernel [127]. Recently, machine learning (ML) is also utilized to train a model which can dynamically choose the best one among multiple scheduling algorithms based on the system states [128–131].

Despite the prevalence, WCET exhibits several important drawbacks. Firstly, WCET is normally much higher than the actual execution time. Measurements on real processors show that the observed maximum execution time of a common application can be 1.5–3 times the average execution time [69, 132]. On top of that, WCET estimates are reported to have overestimation by up to 50% compared to the actual maximum execution time [32]. Therefore, in the scheduling, assuming the WCET for all tasks leads to an exaggeration of the actual execution time, resulting in over-conservative decisions and large energy overheads.

Secondly, the complexity of modern architectures and software has been challenging for WCET analysis. There are several methods to perform WCET analysis. The static method analyzes the control flow of the application to find the longest execution path. Then, WCET is obtained by summing up the execution times of all steps along this path [31]. Though this method is the only way to derive a certified bound of execution time, the complication of modern architectures and applications makes the execution time formidable to analyze [32]. On the other hand, measurement-based and probabilistic methods obtain the WCET by estimating the maximum or the tailing part of the execution time distribution based on sampling data [31]. However, these methods cannot guarantee that the WCET is never underestimated [32], so they do not meet the utmost standard for time-critical applications.

Workload-dependent approaches

Workload-dependent approaches operate on the principle of using design-time information to improve the quality of run-time scheduling. Based on the reference of design-time information, the controller can dynamically eliminate unnecessary pessimism, reducing energy overheads while the timing guarantees are still ensured. Therefore, we suppose that workload-dependent approaches are key to realizing an algorithm with full timing guarantees and low system costs.

The amount and the types of information brought from design time to run time are crucial in the design of workload-dependent approaches. Bringing more information would improve the quality of scheduling, but it increases memory storage and complicates the calculation, inducing more execution time and energy overheads. Therefore, a workload-dependent approach should strike a balance between the scheduling quality and the run-time overheads.

Dynamic scenario (DS) methodology, a generic and systematic design-time/run-time methodology for handling the dynamic nature of modern systems, has been applied to solve this problem [2, 133]. In this methodology, similar system

behaviours are grouped into scenarios at design time. Then, at run time, by identifying the scenarios to which the current and upcoming run-time situations belong, the system behaviours and costs can be more accurately calculated and predicted [134,135]. Based on the principle of dynamic scenarios, cost-effective approaches can be developed for workload-dependent run-time prediction and calculation of execution time and energy consumption [107], which can dynamically trim the upper bound of execution time according to run-time scenarios.

The dynamic scenario methodology serves as a cost-effective low-error way to transfer system cost information from design time to run time. Therefore, in our work, these principles are instantiated to develop the workload-dependent mitigation approach for performance variability, which we will present in the next chapter.

3.4 Summary

In this chapter, mitigation approaches for performance variability in state-of-the-art are reviewed, including the overall structure and the components. The general structure of the approaches is introduced, which is an adaptive control system composed of a timer, a run-time controller, and speed-varying knobs.

The most popular knobs are DVFS and task mapping/migration. DVFS varies the supply voltage and clock frequency at run-time to realize the speed-energy trade-off. It is easy to use and offers wide speed and energy ranges. However, its drawbacks lie in significant switching times (10-150 μs) and incomplete freedom in selecting DVFS modes. Task mapping/migration can change the number and the types of cores allocated to a workload. However, it has long switching times (up to tens of milliseconds) and its effectiveness is highly dependent on the exploitable parallelism in the application.

The adaptive control algorithms in the run-time controller can be classified into two groups, depending on whether they can ensure timing guarantees. Best-effort approaches rely on heuristics to minimize energy consumption to regulate the long-term throughput, but they cannot ensure timing guarantees. On the other hand, WCET-based approaches ensure timing guarantees by using WCET as the upper bound of execution time in scheduling. However, the pessimism in WCET makes the approaches over-conservative and induces large energy overheads. Workload-dependent approaches are promising to solve this problem by bringing design-time information to help in run-time decisions. In this way, the upper bounds can be dynamically trimmed down to reduce pessimism and save energy.

Chapter 4

Workload-Dependent Multi-Timescale Mitigation Approach for Performance Variability: Theory

We have developed a novel workload-dependent multi-timescale mitigation approach to ensure timing guarantees for highly dynamic workloads, tackling the variability from the architecture and software layers in sub-millisecond time granularity. As motivated in Chapter 3, the specific problem formulation of mitigating performance variability with full timing guarantees has not been achieved up to now in state-of-the-art. Nevertheless, our approach will show that full timing guarantees can be achieved by workload-dependent approaches. In the following two chapters, we present the approach in full detail. In this chapter, the principles and design of the experimental approach are elaborated, while in Chapter 5, the simulation we performed in testing this approach are discussed.

This chapter is structured as follows: Section 4.1 describes the motivation and design principles of this approach. The rest of this chapter introduces the salient features of this approach. In Section 4.2, we present heterogeneous datapaths (HDP), a novel system knob. Section 4.3 shows how multiple knobs can collaborate to deal with the variability of different timescales. Thereafter, the complete scheduling algorithm based on dynamic scenarios is elaborated in Section 4.4. Then, we further provide the proof that the algorithm can ensure

timing guarantees in Section 4.5. Finally, the applicability of this approach to other architectures and technologies is discussed in Section 4.6 The work presented in this chapter has been published in the authored work of [136,137].

4.1 Motivation and Principles

As mentioned in Section 3.3.2, ordinary mitigation approaches for time-critical and time-sensitive applications are composed of a DVFS knob and an algorithm based on the WCET model. Although these approaches provide necessary timing guarantees in typical cases, they show strong drawbacks and limitations which undermine their practical application in providing full timing guarantees in highly dynamic workloads. Hence, the state-of-the-art has to fall back to some type of improved best-effort approach where the number of violations of the timing specification will occasionally exceed the imposed bound.

The first limitation of the mitigation approaches with timing guarantees lies in the WCET model. As discussed in Section 3.3.2, the WCET model is too simplistic to reflect the sophistication of modern hardware and computer architectures, resulting in over-margining and failed timing guarantees. Thus, workload-dependent approaches are the only viable option for potentially still ensuring timing guarantees. However, to the best of our knowledge, no known approaches have been published yet which achieve full timing guarantees.

Second, the switching time of DVFS limits the time granularity of mitigation to the millisecond level, as discussed in Section 3.2.1. In the scheduling of a sequence of jobs, the switching time of system knobs must be significantly shorter than the execution time of individual jobs. Otherwise, long switching time could make the deadlines difficult to meet. Figure 4.1 demonstrates an example of how a deadline miss could occur if switching time and job execution time are comparable. The upper and lower parts of this figure show the assumed schedule before and after the execution of Job $(N-1)$ respectively. Job $(N-1)$, a job close to the deadline, has an exceptionally long execution time, out of the expectation of the controller. And due to the long switching time, there is almost no time budget left to execute the Job N. Therefore, the deadline is missed even though the Job N is executed under the fastest mode. To avoid the deadline miss, the controller can increase the speed for the previous jobs so that Job $(N-1)$ finishes much earlier to leave enough timing budget for Job N plus the switching time. However, the energy consumption would be increased significantly.

Moreover, the switching time reduces the computation time spent on the original workload, consequently decreasing the maximum achievable throughput and

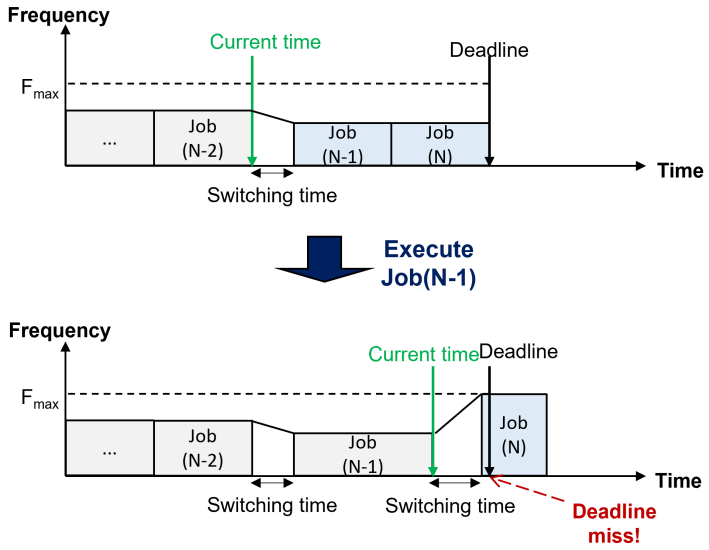


Figure 4.1: Example of how a deadline miss could occur if switching time and job execution time are comparable. The unexpected long execution time of Job $(N-1)$ plus the long switching time leave almost no time budget for Job N, resulting in the deadline miss.

increasing the speed and energy consumption required to execute the same workload. Therefore, the switching time should be limited to an amount that does not significantly influence the energy consumption and the throughput of the workload. If we assume the reasonable switching time overhead to be at most 1% of the original execution time, then in the case of DVFS, the minimum job execution time needs to be several milliseconds. The constraint in the job length prevents the mitigation from operating in finer granularity in time, degrading its capability to tackle sub-millisecond variability.

Our novel workload-dependent multi-timescale mitigation approach is developed to overcome the aforementioned drawbacks. This approach aims to provide guarantees in different conditions where applications are required to meet certain timing specifications. A typical specification is periodic deadlines in real-time applications, such as multimedia applications. Moreover, in general applications, processors are expected to process users' commands and respond to users within a maximal latency, so that the user experience is not degraded. These different requirements of timing guarantees can be handled in a unified way in our approach.

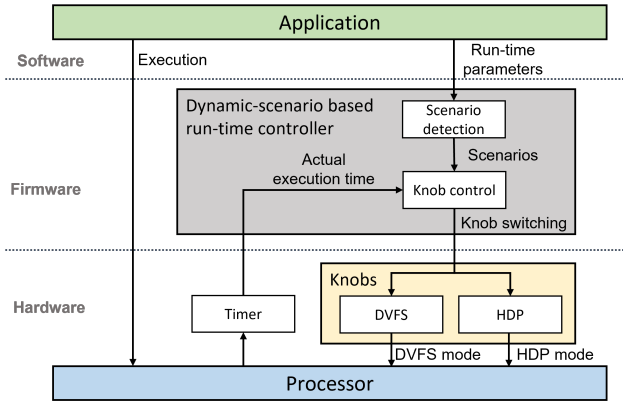


Figure 4.2: Block diagram of the workload-dependent multi-timescale mitigation approach

The proposed approach aims to deal with the performance variability from the architecture and software layers. As discussed in Section 2.2 & 2.3, different inputs of the software can lead to different execution flows in the architecture and the software, causing variability in execution time. In essence, the execution time of running certain software with given inputs on an architecture is deterministic. In other words, it is possible to calculate the exact execution time if we know every detail of the architecture, software and workload. However, this calculation is too complicated to be feasible at run time. Therefore, workload-dependent variability models of execution time need to be created for the use of run-time execution time prediction and bounding. We suppose the best form of the variability model is the dynamic-scenario-based statistic table generated according to design-time profiling analysis, which we will describe later in Section 4.4 and demonstrate by two example applications in Section 5.2.1 & 5.3.1.

Figure 4.2 shows the block diagram of the workload-dependent multi-timescale mitigation approach. In general, this approach is built on the principle of a performance variability mitigation system as described in Section 3.1, consisting of a run-time controller, system knobs, and timers as monitors of performance variability. The controller runs in the optimized firmware or a standalone power management unit, so that the time overhead is minimized. On top of that, this approach has two main features, which improve the two aforementioned drawbacks in state-of-the-art mitigation approaches.

First, the new HDP knob is introduced. With its short switching time of

tens of nanoseconds, HDP improves the time granularity of mitigation to the microsecond level. Furthermore, we develop a run-time controller of collaborating multiple knobs, so the merits of both HDP and DVFS can be exploited for mitigation. Second, a workload-dependent algorithm that properly instantiates the dynamic scenario concept is developed. It can ensure timing guarantees even when a highly dynamic workload is executed. Thanks to the two features, this approach can proactively respond to highly dynamic performance variability down to the microsecond level to ensure timing guarantees.

4.2 Heterogeneous Datapaths (HDP)

Heterogeneous datapaths (HDP) are multiple datapaths of the same functions in the same processor, each with different speed, power, and energy consumption. The heterogeneity of datapaths can originate from differences in circuit design, V_{th} of devices, body bias, etc. At run time, the processor can dynamically select which datapaths to activate so that its requirements of speeds and energy consumption are fulfilled. In past research, datapaths of multiple modes have been proposed, bearing similar concepts of HDP. Some examples include variable-latency functional units [138] and heterogeneous functional units in superscalar processors [139]. Nevertheless, the goals of these datapaths are to minimize average latency or maximize throughput. To our knowledge, HDP is the first to extend the use of multiple datapaths as a system knob enabling run-time changing of speed and energy levels.

To instantiate this idea, we implement HDP design in the RISC-V Ariane core [18], which is a modern representative 64-bit processor core. Ariane core is a building block of OpenPiton [140], a RISC-V multi-core processor platform that is available through open access. Figure 4.3 shows the microarchitecture of the Ariane core, which consists of six pipeline stages. It is an in-order (in-order issue, out-of-order execute, in-order commit), single-issue, 64-bit application-class processor. The dynamism in the microarchitecture includes out-of-order execution, branch prediction, and both instruction and data caches. With these features, it is a representative processor core which contains sources of architectural-layer performance variability.

We equip the Ariane core with HDP in multiple units of the execution stage, including the ALU, CSR buffer, multiplier, and branch unit (the collection of these units is referred to as ALU in the rest of this thesis for simplicity). Our HDP design is demonstrated in Figure 4.4, in which three ALUs (fast ALU, mid ALU, slow ALU) of the same function but different clock frequencies and energy levels are implemented in the pipeline stage of execution. At run time, only

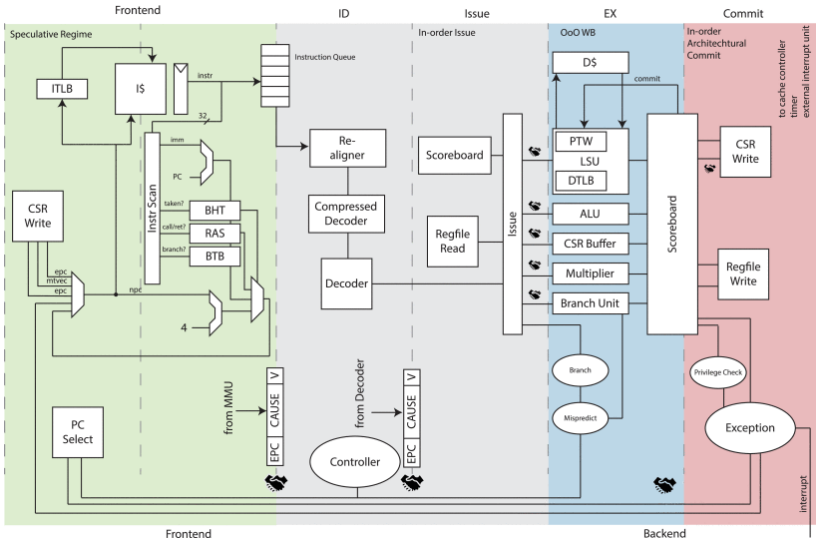


Figure 4.3: Block diagram of the Ariane core. Replotted from [18]

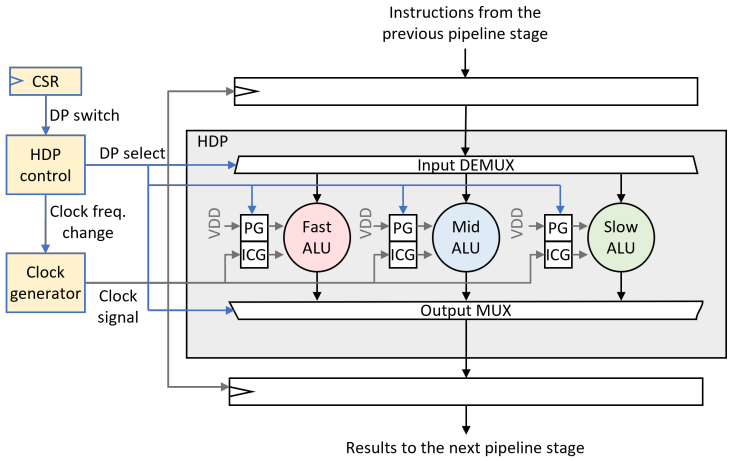


Figure 4.4: Schematic of the HDP design in the ALUs of Ariane core

one ALU is active at one time. The input demultiplexer (DEMUX) dispatches input signals to the active ALU, while the output multiplexer (MUX) collects output signals from the active ALU. When an ALU is not active, its energy consumption should be as small as possible. Therefore, each ALU is equipped with its own power gating (PG) and integrated clock gating (ICG) cells to shut down its local clock network and power supply during its idle time. To enable power gating in small power domains of an ALU size, the microarchitectural power gating technique [141] is applied.

A dedicated circuit is designed to control the HDP. To switch the mode of the HDP, the controller simply change the value of a control and status register (CSR) [142] specifically assigned for configuring the HDP state, by issuing a CSR-write instruction. The CSR value change triggers a sequence of actions of the HDP control unit. First, the control unit waits for the instructions currently executed in the active ALU to be finished. The waiting time is 66 cycles at most, which is the longest latency of all ALU functions. Meanwhile, the control unit wakes up the ALU the processor is going to switch to, and reduces the clock frequency if the successive ALU is slower than the active one. Once the active ALU is purged and the successive ALU is ready, the control unit switches the modes of the input DEMUX and the output MUX to change the direction of data signals. Finally, the control unit enables clock gating and power gating of the previously-active ALU and raises the clock frequency if the successive ALU is faster than the previously-active one.

The switching time of the HDP, including the waiting time, the changing time of the clock frequency, and the wake-up time of power gating, only adds up to tens of cycles in total, equivalently tens of nanoseconds in a normal processor of several GHz. The switching time overhead can be kept under 1% even if the HDP switches every tens of microseconds. Therefore, HDP can serve as a knob of fine-grained mitigation at the sub-millisecond level. For all practical applications with timing guarantees that we are aware of, this limit of μs granularity is sufficient for the imposed timing specifications.

The HDP is implemented based on the 3nm technology of IMEC [143], demonstrating the feasibility of the approach for highly scaled modern technologies where the variability ranges are the highest and providing the timing guarantees is hence the hardest. Each ALU is synthesized and physically designed with different clock frequency constraints. The relations of the clock frequency and energy of physical designs of ALUs are shown in Figure 4.6, simulated with a JPEG decoding workload. Different V_{th} options of devices contribute to the differences in the speed and energy among designs. In principle, ALUs of higher speed and lower energy are composed of devices of lower V_{th}. Designs with multiple device V_{th} types are generally better in the speed-energy trade-off, since they can be optimized by using lower-V_{th} devices in critical

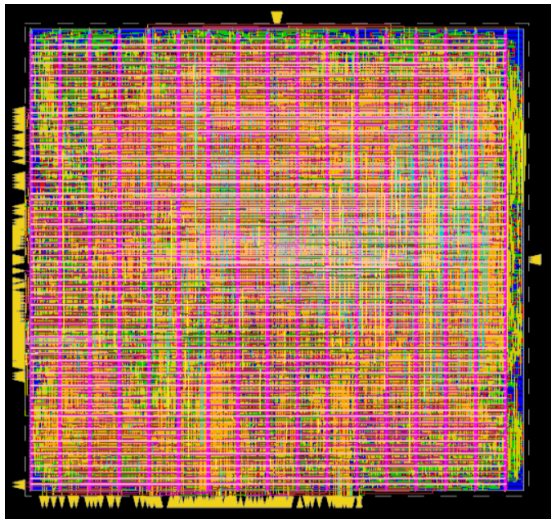


Figure 4.5: Layout of the fast ALU of the HDP

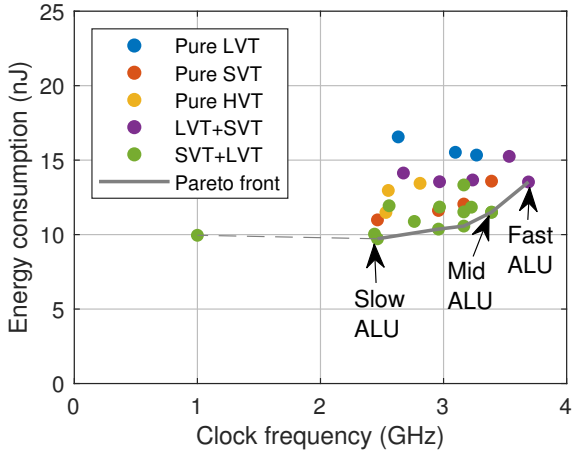


Figure 4.6: Energy consumption v.s. clock frequency of physical designs of ALU.

Design	Fast ALU	Mid ALU	Slow ALU
Clock frequency	3.69 GHz	3.39 GHz	2.46 GHz
Number of total gates	18099	18092	12827
Number of flip-flops	1188	1188	853
Number of other gates	16911	16904	11974
Multi-Vth libraries used	LVT, SVT	SVT, HVT	SVT, HVT
Technology	3nm [143]		
Operating condition	TT/0.7V/25C		

Table 4.1: Summary of the ALU Designs in the HDP

paths and higher-V_{th} devices in non-critical paths. The circuit topology also contributes to the differences in the speed and energy.

The candidates of ALU designs forming the HDP should be chosen from the ones on the Pareto front of energy-speed relations. To realize the full potential of the mitigation, the HDP design should make the most of the speed-energy curve. This means that the fastest HDP mode should be as fast as possible before the point at which speed improvement is so small that additional energy investment is not worth it any more. On the other hand, the slowest HDP mode should be as slow as possible until the point at which the energy is flattened and even increases due to larger leakage energy brought by longer execution time.

As a result, our HDP design has three modes, each of which uses a different ALU design on the speed-energy curve in Figure 4.6. Figure 4.5 shows the layout of the physical design of the fast ALU, and Table 4.1 summarizes the designs of ALUs in the HDP. Each ALU contains about 12,000–19,000 gates, within which about 800–1200 are flip-flops To diversify the speed and energy of ALUs, devices of different V_{th} are used. Compared to the fast ALU, the mid ALU saves 15% energy by 8% speed degradation, while the slow ALU saves 28% energy by 33% speed degradation. The design optimization of HDP by state-of-the-art EDA tools is limited due to the complexity of the timing constraints, so we expect the energy efficiency of HDP could be better in future with the improvement of EDA tools.

The introduction of HDP also induces overhead on the chip area. The amount of the area overhead depends on the number of modes of the HDP. Since the hardware of HDP is essentially multiple functional units of the same functions in different circuit topologies, the chip area of HDP is approximately multiple times the total area of the original functional units. However, in modern processors, the functional units only account for a small proportion of the whole chip area. Therefore, the area overhead is relatively small compared to the whole chip. In

our HDP design, the total three ALUs take up approximately 2.9x the area of the original fast ALU. However, in the original Ariane design, the ALU plus the multiplier only account for 13.2% area of the core, while the area of the processor core is estimated to be 10% of the whole area of the SoC [18]. Therefore, the area overhead of the HDP is estimated to be only 2.5% of the SoC.

While we demonstrate HDP in the datapath of Ariane core, the same design principles are viable for all kinds of datapaths, not only for microprocessor cores. It can also be easily extended to domain-specific architectures, such as GPUs or highly optimized parallel matrix-multiply-accumulate accelerators in the field of machine learning. Apart from datapaths, the concept of bringing heterogeneity into microarchitecture is also potentially applicable to other components in the entire macroarchitecture. For the memories, switching among different types of memory instances which exhibit different delay-energy trade-offs is quite easy to achieve. For example, one can imagine having part of a memory macro with smaller subbanks to store time-critical data. We look forward to the extensions of this concept to other components (interconnect networks, etc.) to be explored in the future, so that the energy efficiency and resiliency of the individual components and the entire macroarchitecture can be enhanced.

4.3 Collaboration of Multiple Knobs

Variability of different sources occurs in different timescales. For example, as mentioned in Section 2.1.2 and 2.1.3, abrupt voltage droop can happen in tens of nanoseconds, while temperature normally changes gradually over hundreds of microseconds. Depending on characteristics, architectural and software-level variability spans over a wide range of timescales from nanoseconds to seconds. To mitigate a kind of variability, the switching time of a knob must be shorter or comparable to the timescale of the variability. Otherwise, the knobs will not be quick enough to react to the variability.

A holistic view of the timescales of variability sources and mitigation knobs is illustrated in Figure 4.7. The timescales of the knobs make them suitable for tackling different type of variability sources. For example, the 10–150 μs switching time of DVFS makes it effective in tackling temperature fluctuation, but too slow to react to voltage droop. Therefore, a complete solution for all kinds of performance variability should integrate multiple knobs of different switching times, so each kind of variability is handled by the knobs of its respective timescales.

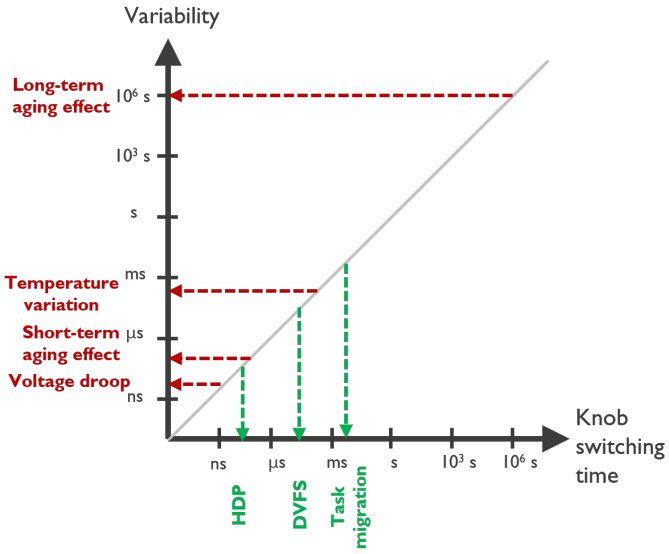


Figure 4.7: Variability sources and mitigation knobs arranged by the timescales, which tells whether the knobs can react faster than the variability for successfully mitigation.

In our approach, the multi-timescale concept is embodied in the use of both HDP and DVFS. These two knobs are complementary in switching time, speed range and energy range. DVFS enables wide speed and energy ranges, but its switching time is longer, so it is suitable to react to the large and slow variability at the millisecond level. On the other hand, HDP can switch in a shorter time and provide fine-grained speed and energy levels, but its speed and energy ranges are limited, so it is suitable to respond proactively to small and abrupt variability.

To illustrate this concept, Figure 4.8 shows all available speed and energy levels of executing a JPEG-encoding workload on the ALUs of the Ariane core equipped with both HDP and DVFS, benchmarked with the cycle-accurate simulation flow described later in Section 5.1.1. These levels are fine-grained and cover wide speed and energy ranges, in which the slowest mode has only 17% speed and 23% energy consumption compared to the fastest mode.

In general, the modes of better energy efficiency (low energy-to-speed ratio) are preferred. From Figure 4.8, one can obtain the Pareto-optimal modes in terms of energy efficiency, which are the modes close to the lower-right corner of

the figure. Although several modes are not Pareto-optimal, fast switching time could allow these modes to have more energy saving in mitigating fast variability. For example, switching from the 0.7/mid mode to the 0.6/fast mode requires switching time overhead of both DVFS and HDP, which reduces the time spent in the latter mode, lowering the energy efficiency. In this case, switching from the 0.7/mid mode to the 0.7/slow mode can yield better energy efficiency in the short term due to shorter switching time, even though the 0.7/slow mode is not Pareto-optimal.

The design of the knobs should consider the effectiveness and flexibility of the mitigation approaches. In principle, a good set of modes should have high energy efficiency, wide ranges of speed and energy, and fine-grained speed and energy levels. One can imagine a fully optimized set of modes to consist of Pareto-optimal modes in terms of energy efficiency, in which the neighbouring modes are separated by a minimal interval of speed and energy levels. However, the optimization of knobs is restricted by the circuit design and the underlying technology. For example, the design of HDP is limited by the design space of circuit topology and the device V_{th} flavours. Across this design space, the range of speed and energy that yields good energy efficiency is relatively small, as shown in Figure 4.6. In fact, most of the range of good energy efficiency is covered by the modes of our HDP design. Moreover, since the same HDP hardware is used regardless of the DVFS setting, optimizing a particular HDP curve in Figure 4.6 will also affect the other HDP curves, potentially making other curves sub-optimal. Therefore, it is not possible to optimize each mode individually at the same time. Due to the non-ideality of the knobs in switching time, speed/energy range, granularity, and energy efficiency, it is crucial for the scheduling algorithm to smartly schedule the mode setting to minimize these drawbacks for energy minimization and timing guarantees.

4.4 Dynamic-Scenario-Based Scheduling Algorithm

The adaptive control algorithm is based on the principles of dynamic scenarios (DS), aiming to proactively respond to performance variability to ensure timing guarantees. In principle, the algorithm takes advantage of the information of the dynamic scenarios, which are the groups of workload segments with similar execution time and energy costs, defined in design-time profiling. By identifying the scenario that the system encounters at run time, the algorithm can predict execution time with limited error, and find an upper bound of the execution time which is substantially less pessimistic than the WCET model.

In a nutshell, the core of dynamic scenarios is a set of generic principles in

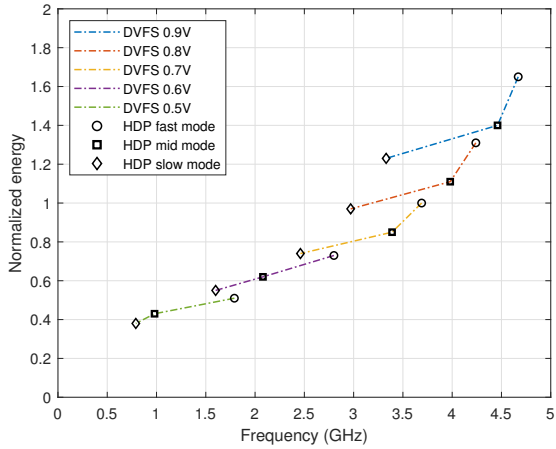


Figure 4.8: Speed and energy trade-off of the ALUs running in each of the possible combinations of the DVFS mode (0.5V–0.9V) and the HDP mode (fast, mid, and slow)

realizing design-time/run-time co-optimization methodologies that tackle any kind of dynamism in modern hardware/software systems with low design margins. The DS principles are developed in previous research and elaborated in [134,135]. These generic principles can be instantiated into practical approaches that tackle different variability problems in every new context.

Here, we instantiate the DS principles in the novel context of mitigating timing variations, which results in a new mitigation approach tuned to our specific objectives of ensuring timing guarantees for processors equipped with speed-varying knobs like HDP and DVFS. The DS-instantiated approach consists of a design-time scenario clustering method for creating dynamic scenarios, as well as a run-time scheduling algorithm for selecting the optimal mode of the knob while ensuring timing guarantees. The concepts and the instantiation of the dynamic scenario principles are explained in the following sections, while two implementations for different applications are shown in Section 5.2 & 5.3.

4.4.1 Principles of Dynamic Scenarios

In principle, dynamic scenarios are groups of similar system behaviours from the perspective of multi-dimensional system costs, such as execution time, energy consumption, and resource requirements. The scenarios provide a

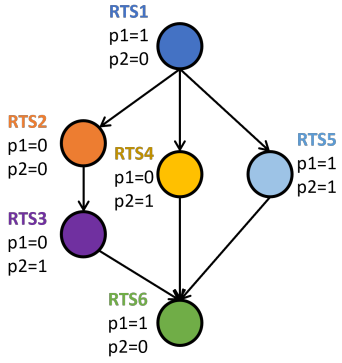
reference by which the run-time controller can estimate or calculate a bound of the system costs of the upcoming workloads. By changing the system behaviours to adapt to the upcoming scenario at run time, design systems can improve performance, reduce costs, and possibly recalibrate themselves during execution. The principles of dynamic scenarios are applicable in many contexts, instantiated to become solutions to many run-time system optimization problems, such as DVFS scheduling, energy-efficient exascale computing, and flexible manufacturing system design [135].

Figure 4.9 depicts the basic concepts of dynamic scenarios. Based on the design-time profiling and control flow analysis, the workload is segmented into atomic units called *run-time situations* (RTS), as depicted in Figure 4.9a. Each RTS is a situation of deterministic behaviours the system can encounter at run time, such as the body of a loop, a certain function the system needs to execute, or the read access to a certain chunk of data. In addition, each RTS is associated with its system costs, such as execution time and energy consumption. With different settings of knobs, the system costs of an RTS can vary and form a trade-off curve. As shown in Figure 4.9b, each curve in the time-energy plane represents the system costs of an RTS.

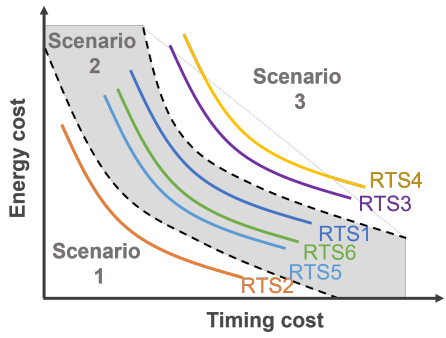
Since the number of RTSs is too many to be recorded for run-time usage at low costs, the RTSs are further clustered into scenarios. The RTSs in the same scenario share some similar properties which become the identity of the scenario. In general, the RTSs in the same scenario are required to be similar in the system cost trade-offs in executing them under different knob settings, so that the system cost of each RTS will be approximated by the aggregated values (average, maximum, minimum, etc.) of the system costs of all RTSs in the same scenario. Figure 4.9b visualizes this process, in which the clustering is equivalently the segmentation of the multidimensional cost space into regions. After clustering, each scenario contains multiple RTSs with similar costs. Then, *monitoring parameters* are discovered, which are the parameters used to distinguish scenarios at run time. The monitoring parameters and the system costs of each scenario are summarized and recorded in a table like Figure 4.9c.

At run time, based on the reference of the scenarios, the controller can identify the scenarios of the current and upcoming RTSs and then predict the future system costs with limited errors. Based on accurate system cost prediction, the controller can derive the optimal configuration of the system knobs for the upcoming workload. In addition, the costs can be updated and the scenarios can be re-clustered based on the monitoring at run time, which further enhances the adaptability of dynamic scenarios to variability [2].

The concept of dynamic scenarios leads to a generic flow consisting of several



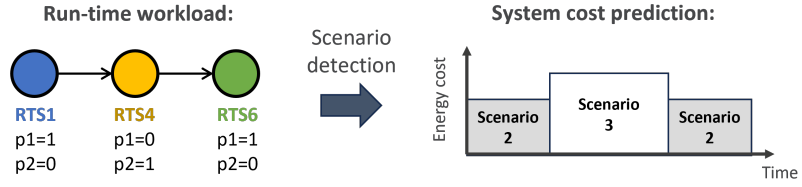
(a)



(b)

Monitoring parameters		Scenario	RTS	System costs	
p1	p2			Timing	Energy
0	0	**1**	2	low	low
1	0, 1	**2**	1, 5, 6	middle	middle
0	1	**3**	3, 4	high	high

(c)



(d)

Figure 4.9: Depictions of the basic concepts of dynamic scenarios. (a) At design time, different parts of the workload are classified into run-time situations (RTS). (b) RTSs are clustering into scenarios based on their timing and energy costs. (c) the monitoring parameters and the system costs of scenarios and the monitoring parameters are recorded in a table. (d) at run time, system costs are predicted based on the scenarios

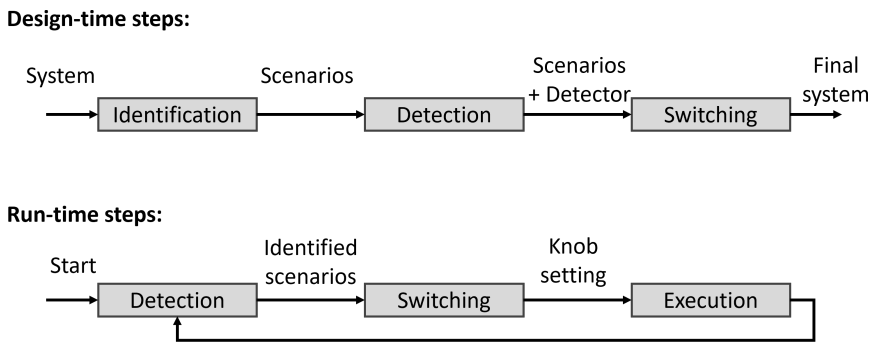


Figure 4.10: Basic flow of realizing a solution to dynamism-related problems based on dynamic scenario principles

design-time and run-time steps for realizing a solution to any dynamism-related problem in different contexts. The basic flow of these steps is shown in Figure 4.10. At design time, the required steps are described as follows:

1. **Identification of the scenario set:** In this step, the solution space of RTS-scenario mapping is explored, and an appropriate scenario set is determined. At the beginning of this step, the system needs to be profiled with high coverage to obtain the system costs of every possible situation of the system. Based on the profiling results, the RTSs are defined. The profiling approach and RTS definition are dependent on the system. Then, the profiled system costs of every RTS become the reference of the following scenario set creation.

2. **Detection of the scenario:** In this step, a function that maps an RTS to the scenario it belongs to is created, called the scenario detector. In practice, the system will execute this function at run time to identify the scenario of the encountered RTSs. Normally, the detector is realized in the form of a look-up table, in which the indices are the *monitoring parameters*. Normally, monitoring parameters can be certain system parameters and application variables which influence a lot on the system costs, such as the input size of the application. Since the run-time overhead of the detection depends on the scenario set, the scenario set may be refined in this step to reduce the detection overhead.

3. **Switching from one scenario to another:** Switching is the act of changing the system from one knob setting to another. Since the system cost of RTSs varies with the knob setting, different knob settings can be

preferred for the same RTS under different system conditions. This step aims to develop an algorithm which is used at run time to decide whether to switch the knob setting and how to switch the knob setting, based on the scenarios detected at run time. Since switching incurs run-time overhead, it is not always preferred to switch to the optimal knob setting for the scenario. Therefore, the algorithm should consider both the system costs of the scenario and the switching overhead at the same time in making switching decisions.

The final products of the design-time steps are the scenario set, the scenario detector, and the algorithm for the switching decisions. These products are stored and used at run time for predicting the dynamism and on-the-fly optimization of the system. A run-time controller is implemented which uses the scenario set to run the scenario detector and the algorithm to switch the knob setting of the system. In particular, the basic run-time steps performed in order by the controller are described as follows:

1. **Detection of the scenario:** In this step, the scenario detector is called to identify the scenarios of the RTSs the system encounters now and will encounter in the near future. The controller reads the monitoring parameters and uses the function defined in the scenario detector to acquire the scenarios of the encountered RTSs. With the scenario information, the bound of the system costs of the encountered RTSs can be known in advance by the controller.

2. **Switching from one scenario to another:** In this step, the algorithm is called to decide on whether to switch the knobs and how to switch the knobs, according to currently known scenarios of the future RTSs. Based on the result of the algorithm, the knob setting is switched or unswitched accordingly.

3. **Execution of the scenario:** In this step, the system executes the current RTS under the knob setting decided in the last step. Then the controller returns to the first step for the following RTSs.

These steps are repeated continuously so that the controller deals with the encountered RTSs in sequence and makes rolling decisions as the system proceeds.

In the context of performance variability mitigation, the above dynamic scenario principles are instantiated to develop a scheduling algorithm that controls multiple knobs to handle performance variability for timing guarantees. Each of the aforementioned steps is implemented in the instantiated methodology

but tailored for the specific purpose of performance variability mitigation. In the following sections, the details of the methodology are elaborated.

4.4.2 Design-Time Scenario Clustering

At design time, the scenario set, the scenario detector and the controlling algorithm are generated. The scenario set and the scenario detector should be tailor-made for the application context. Therefore, the detailed implementation of design-time steps can be different from one application to another. In this section, we only describe the general process of design-time steps without addressing any application-specific aspects, while Section 5.2 & 5.3 show two examples of design-time steps of creating the application-specific scenario set and the scenario detector.

Figure 4.11 shows the process of design-time steps for creating the application-specific scenario set. Since scenarios are grouped mainly based on system costs, it requires complete data on the system costs in executing each part of the application. Therefore, the application first needs to be thoroughly profiled in a finer timescale than the time granularity of the mitigation. In our approach, RTL-level simulation is performed to profile the application, which is accurate down to the cycle level, approximately the nanosecond level, much finer than the sub-millisecond time granularity of the mitigation. The application also needs to be profiled with different inputs and configurations to traverse as many different execution paths in the data flow as possible, so that the variability in the system costs of executing the application is thoroughly explored. The existing techniques for improving profiling coverage can be utilized, such as data flow analysis [135].

Then, the process of scenario set generation is performed, depicted in Fig. 4.11. First, the profiling traces of the workload are segmented into atomic units called *thread node* (TN). The TNs are further categorized into RTSs, in which TNs with identical roles in the data flow or identical behavioral characteristics are labeled as the same RTS. For example, TNs which perform different iterations of the same loop can have equivalent roles in the data flow and identical system costs, so they can be categorized into the same RTS. Essentially, a workload can be seen as a sequence of TNs, each of which instantiates the same or different RTSs.

There are no strict definitions of the size and the content of a TN or an RTS. However, since TNs and RTSs are the smallest executable units used in the scheduling of the control algorithm, their execution time should match the time granularity of the mitigation. In our approach, HDP enables mitigation down

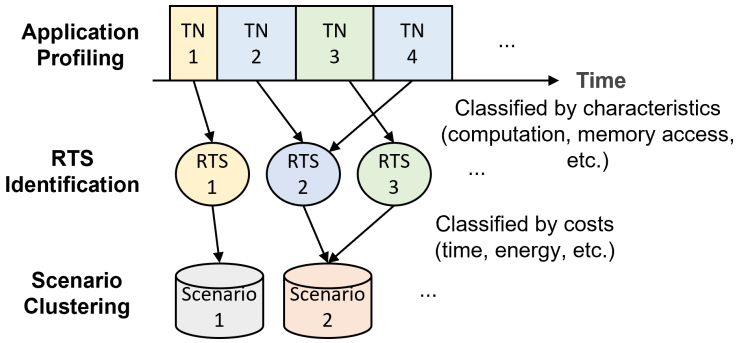


Figure 4.11: Process of generating a scenario set in the design-time process of the dynamic scenario based approach

to the job size of the order of tens of microseconds, so each TN and RTS has an execution time of approximately tens of microseconds.

TNs and RTSs can be defined either manually or automatically. The manual method relies on the knowledge of the software engineers on the data flow to group the segments of the profiling traces into RTSs. This method is applicable if the data flow is completely known by the engineers. However, for complicated applications, their data flows can be too difficult for the engineers to analyze. Therefore, automatic data flow analysis algorithms can be utilized to systematically analyze the data flow of the application and identify RTSs [135].

Each RTS has its system costs (execution time, energy consumption), which also depends on the knob setting that is used when executing the RTS. Therefore, the system costs of each RTS can be represented by a trade-off curve of multiple cost factors, in which each point on the curve represents the system cost of the RTS under a certain knob setting.

Thereafter, the RTSs are further clustered into dynamic scenarios. The primary consideration of the scenario clustering is the similarity in system costs. At run time, the system cost of each RTS will be approximated by the aggregated values (average, maximum, minimum, etc.) of the system costs of all RTSs in the same scenario. The error of this approximation will only be small if the differences in the system costs of RTSs in the same scenario are minimal.

During scenario clustering, the scenario detector is created at the same time, which can map an RTS to its belonging scenario based on the monitoring parameters. At run time, the controller should be able to run the scenario detector to identify the scenario of an RTS by the monitoring parameters, with low overhead of system costs. The choices of monitoring parameters are another

important consideration in scenario clustering. To reduce the run-time overhead, the monitoring parameters should be easily read at run time, such as application variables or metadata of the input data. The monitoring parameters should also have high impact on the system costs of the RTSs, so the scenarios, which are clustered by the monitoring parameters, can be distinct in system costs from one to another.

A common form of the scenario detector is a look-up table in which the indices are the monitoring parameters, and the look-up values are the belonging scenario of the RTS with the corresponding monitoring parameters. Hence, at run time, detecting the scenario of an RTS is done by reading the monitoring parameters of the RTS and using them as indices to look up the table. In this form of the scenario detector, each scenario can be seen as a region in the multidimensional space of the monitoring parameters. Therefore, the problem of grouping the RTSs into scenarios can be modelled as the segmentation of the parameter space into regions.

In practice, the discovery of monitoring parameters and the scenario clustering can be performed manually by analyzing the execution flows, if the complexity of the application is not too high. For applications of high complexity or unknown execution flows, automatic clustering algorithms can be developed on the same principles to perform these tasks systematically [144].

After clustering, statistics (worst RTS cost, average RTS cost, etc.) for each scenario are stored in a look-up table to be used at run time for the calculation of execution time. In this approach, we record the worst execution cycles and the average execution cycles for the RTSs of each scenario. At run time, the worst execution cycles are used as the upper bounds of execution cycles for ensuring timing guarantees. On the other hand, the average execution cycles are used as the references for the prediction of execution cycles for normal cases. The monitoring parameters also are recorded as the indices of the look-up table. At run time, detecting a scenario is merely querying the look-up table by the monitoring parameters.

In practice, the scenario table can be stored in a specific region of the program codes of the application, which can be read by the controller at run time. Alternatively, the table can be stored in a specific region in the memory of the system, if the system is domain-specific that only runs a limited number of applications.

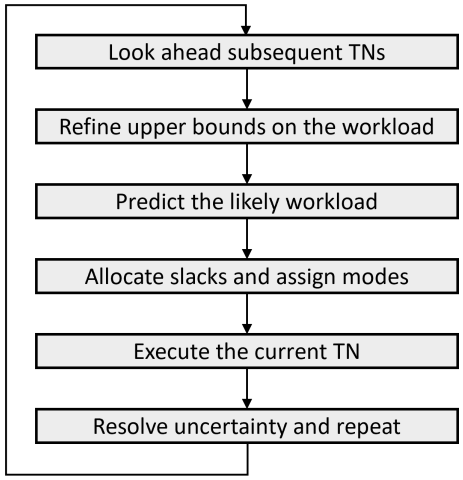


Figure 4.12: The generic flow of DS-instantiated scheduling algorithms

4.4.3 Priniciples of the run-time scheduling algorithm

At run time, the exact execution time of the workload depends on the actual inputs, such as the compressed image to be decoded by the image decoder, or the sound waves to be encoded by an audio encoder. By applying different inputs, the variability in execution time is injected into the system, which could potentially cause deadline misses. Nevertheless, the algorithm has access to the dynamic scenario table, which contains parameterized bound and expected execution time based on the correlation-based analysis performed at design time, elaborated in the last section. With these references, the run-time controller can obtain the upper bound of the execution time with full guarantee, as well as predict the execution time with limited discrepancy. Therefore, with proper scheduling of the modes of the knobs, variability in execution time can be offset before the deadlines, so full timing guarantees can be ensured.

The principles of generic DS-instantiated scheduling algorithms have been developed and elaborated in [133], in which the essential steps of the algorithm are defined and shown in Figure 4.12. Here, we summarize the principles from step to step as follows:

- **Look ahead subsequent TNs:** The controller gathers the updated information of the subsequent TNs, so that the current TN and subsequent TNs are considered together in the following decision process. The current TN and the subsequent TNs that the controller can look ahead form the

optimization window of the mode scheduling of the knobs. In this way, the controller can avoid selecting knob settings that are only optimal for the current TN but sub-optimal in the long run. With dynamic scenarios, the action taken in this step is actually to identify the scenarios of these TNs and look up the scenario-specific information brought from the design time.

- **Refine upper bounds on the workload:** Given the information obtained in the last step, the controller calculates the refined upper bounds of the uncertainty in the execution time of the TNs in the optimization window. The bounds are used to derive the maximum slacks that can be exploited while ensuring timing guarantees. The tighter the bounds the larger the slacks. Thus, the bounds should be as tight as possible to maximize the slacks.

- **Predict the likely workload:** The controller predicts the most likely execution time of the TNs in the optimization window. The predictions are used in the following steps to figure out the knob settings that are most likely to be optimal over the optimization window, but not for the extreme cases that rarely happen.

- **Allocate slacks and assign modes:** The controller solves the optimal slack allocation and mode assignment over all TNs in the optimization window. The amounts of slacks need to be calculated by the worst case of ensuring timing guarantees, in which the hardest deadline constraints and the upper-bound execution time of each TN are assumed. Only when the slack is available beyond what is required to meet the hardest deadline, then that slack can be used on the TNs. Once the amount of slacks is obtained, the optimal slack allocation and mode assignment should be solved with the predicted most likely execution time of the TNs. The switching overheads also need to be considered in this optimization problem, so the slack should be particularly allocated for the switching time and the switching energy should add to the total energy cost, which makes frequency switching undesirable. Due to the switching overheads and the discrete modes of knobs, the derivation of slack allocations and knob settings becomes a nonlinear optimization problem, which could not be solved with small computation effort. Hence, heuristics need to be applied to find near-optimal solutions in practice.

- **Execute the current TN:** The controller switches the knob to the assigned mode for the current TN. Then, the TN is executed.

- **Resolve uncertainty and repeat:** Once the current TN completes its execution, its uncertainty is resolved and its exact execution time is known.

Also, more information about the subsequent TNs can become available during the execution time. Thus, the controller needs to revise its optimal schedule with the new and updated information by repeating the previous optimization process. In this way, the controller keeps incrementally optimizing the schedule as the uncertainties gradually resolve over time, so near-optimal decisions are reached at any time during the workload.

4.4.4 Run-Time Scheduling for a Single Knob

Based on the principles described in the last section, we have implemented the DS-instantiated scheduling algorithm which supports both single-knob and multiple-knob settings. In the following, we describe the single-knob algorithm, showing the essential procedure of the algorithm. The algorithm assumes a knob with negligible switching overheads, such as HDP. Then, in the next section, we further describe the full multiple-knob algorithm, showing how the algorithm is extended to handle the switching overheads and assign modes of multiple knobs.

The overall procedure of the run-time single-knob scheduling is shown in Algorithm 1. At the start of the workload, an *optimal schedule* is generated by scheduling with the average number of cycles of each TN and the hard deadlines of the workload. This schedule is the presumed ideal schedule which can ensure timing guarantees and minimize energy consumption if there is no variability. The goal of the algorithm is to keep the progress ahead of the optimal schedule, but only by a necessary amount of slack required to ensure timing guarantees. Then, the controller defines a deadline for each TN, which is equal to the TN finish time in the optimal schedule. These deadlines serve as checkpoints for the progress of the workload. In addition, the controller creates a buffer to keep track of the scenarios of the TNs in the optimization window, so the controller can look into the buffer to evaluate the future workload.

At the start of the execution of each TN, the controller calls the scenario detector to identify the scenario of the last TN in the buffer [145]. Then, two processes are performed to decide the mode of the knob to run the next TN. First, the controller calculates the timing guarantee criteria by a *refined upper bound* procedure, shown in Algorithm 2. The timing guarantee criteria are expressed by the minimum required speed for the next TN, which is the slowest speed to run the next TN while still ensuring timing guarantees. Fig. 4.13 gives a visual illustration of the principle of this procedure. To calculate the minimum required speed, the controller generates a schedule which could create the greatest possible slack. This means that the controller assumes the other scenarios in the buffer to be their worst execution time and run at the fastest

Algorithm 1 Run-time scheduling

Input: thread nodes $TN_{1...N}$, buffer size B
Output: modes which each TN uses $m_{1...N}$
$CB_{1...N} \leftarrow$ cycle budgets of $TN_{1...N}$
$DL_{i...N} \leftarrow OptimalSchedule(CB_{1...N})$
Initialize an empty buffer buf
for $i \leftarrow 1$ to B **do**
    $S_i \leftarrow ScenarioDetect(TN_i)$
    Push S_i to buf
end for
for $i \leftarrow 1$ to N **do**
    $f_{mr} \leftarrow RefinedUpperBound(buf)$
    $f_{opt} \leftarrow LikelyPrediction(buf)$
    $m_i \leftarrow$ the nearest faster mode of $max(f_{mr}, f_{opt})$
    Execute TN_i under mode m_i
    Pop S_i from buf
    if $i + B \leq N$ **then**
        $S_{i+B} \leftarrow ScenarioDetect(TN_{i+B})$
        Push S_{i+B} to buf
    end if
end for

mode. If gas pedals mentioned in Section 3.2.3 exist, the controller also needs to assume that the gas pedal is activated to run for its maximum allowable time and then switch to the normal modes. This procedure is performed for each hard deadline of the TNs in the buffer, and the hardest speed constraint of all is considered in the end.

Second, *likely prediction* is performed to calculate the optimal processor speed that is most likely to minimize the total energy consumption for the TNs in the buffer. This process is shown in Algorithm 3. At the beginning of the procedure, it would predict the real execution time of the TNs in the buffer, based on the statistics of the scenarios acquired in the design-time process. Heuristics can be applied to improve the accuracy of the prediction. In this work, we predict the TN execution time to be the average execution time of the scenario, which is measured during the design-time scenario clustering, as described in Section 4.4.2. Based on the execution time prediction, the scheduler generates an optimal schedule that minimizes the total energy consumption for the TNs in the buffer, and thus the mode to run the next TN is decided. Solving the optimal schedule at run-time will not be feasible since it takes too much computation, so heuristics need to be applied. In this work, our heuristics are designed based on the concept that it is near-optimal to run constantly

Algorithm 2 Refined Upper Bound

```
1:  procedure RefinedUpperBound(buf)
2:      T_cur ← current time
3:      f_max ← maximum frequency
4:      B ← number of TNs in buf
5:      DL_{1...B} ← deadlines of TNs in buf
6:      WC_{1...B} ← worst cycles of scenarios of TNs in buf
7:      PDL_B ← DL_B
8:      for i ← B downto 1 do
9:          PDL_{i-1} ← min(PDL_i − WC_i/f_max, DL_{i-1})
10:     end for
11:     f_mr ← WC_1/(PDL_1 − T_cur)
12:     return f_mr
13: end procedure
```

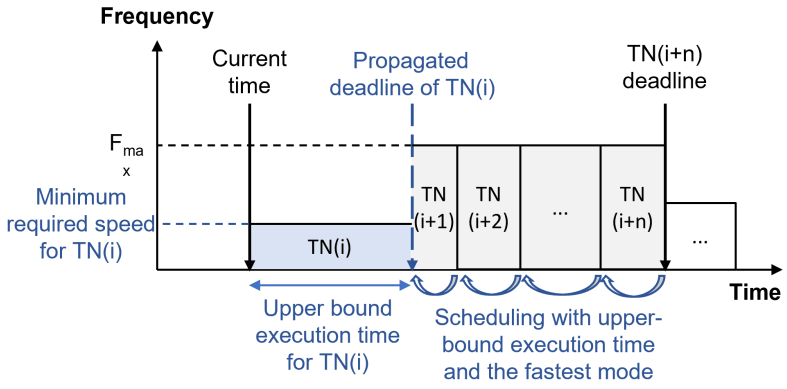


Figure 4.13: Refined upper bound process to calculate the minimum required speed for the next TN for ensuring timing guarantee

at the slowest possible speed. Therefore, the controller calculates the optimal speed which is just enough to run all TNs to meet the deadline of the last TN in the buffer. Finally, the controller finds the slowest mode that is faster than both the optimal speed and the minimum required speed, as shown in Line 12 of Algorithm 1. This becomes the selected mode to run the next TN.

The algorithm above can be easily extended to other contexts and compatible with other scheduling techniques. For example, for scheduling for multiple applications, real-time scheduling [33, 126] can be performed on top of the

Algorithm 3 Likely Prediction

```
1: procedure LikelyPrediction(buf)
2:     T_cur ← current time
3:     B ← number of TNs in buf
4:     DL_1...B ← deadlines of TNs in buf
5:     AC_1...B ← average cycles of scenarios of TNs in buf
6:     f_opt ← sum(AC_1...B)/(DL_B − T_cur)
7:     return f_opt
8: end procedure
```

algorithm, deciding the order of running applications. Then, our algorithm handles fine-grained time resource arrangement and performance variability management. In addition, the maximal latency specification of general applications can be treated as a single deadline. For time-sensitive applications, providing a strict bound on the maximal deadline misses can be realized by statistical analysis at design time and ignoring a number of the hardest deadlines at run time.

Still, there is room for improvement in the algorithm. In this approach, the detection of scenarios at run time is purely based on the monitoring parameters defined at design time, which is viable for simple workloads, such as periodic frame-based applications we use in the experiments later in Chapter 5. However, scenario detection is generally predicting future workload, which is not an easy task in realistic scenarios with complicated interactions among multiple applications and software abstraction layers. In future work, scenario clustering can be potentially generalized and enhanced by predictors based on inference of machine learning models, so it can handle any application context. Moreover, switching scenarios as often as every TN might not be optimal due to switching overheads, so it is crucial to decide when to switch from one scenario to a new one. The conceptual work on switching decisions has been developed in [145], which can be applied to improve the current approach in the future.

4.4.5 Run-Time Scheduling for Multiple Knobs

To exploit the advantages of both HDP and DVFS, the run-time scheduling is adapted for multiple knobs. The flow diagram of the enhanced run-time procedure is shown in Fig. 4.14. In principle, the algorithm sequentially decides the modes of the knobs from the slow-switching ones to the fast-switching ones. In our approach, the DVFS mode is decided first, and then the HDP mode is decided. Moreover, the switching rates of HDP and DVFS are different. HDP

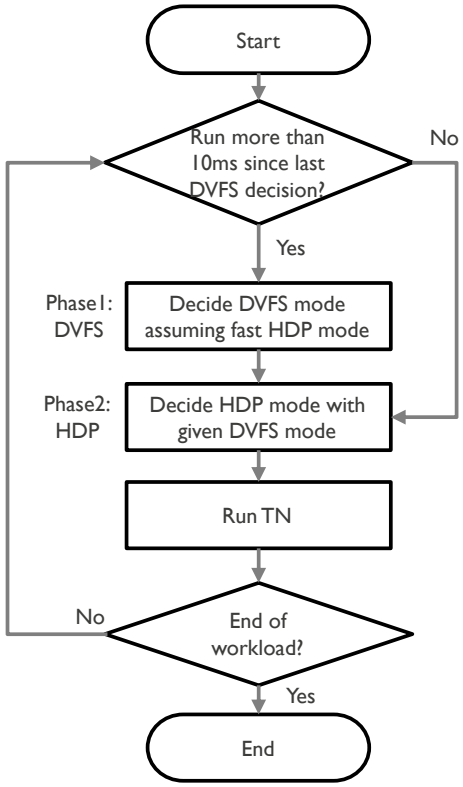


Figure 4.14: Flow diagram of the run-time process of the multi-timescale DS based approach, in which the modes of knobs are decided sequentially from the slow-switching ones to the fast-switching ones

can switch at the start of each TN. On the other hand, DVFS can only switch once over multiple TNs. Otherwise, its switching time overhead would greatly lengthen the execution time.

Here, we limit the interval between two DVFS switches to be larger than a threshold value $T_{th,DVFS}$. Specifically, the controller would consider switching the DVFS knob only when the time since the last time the controller decided the DVFS mode is more than $T_{th,DVFS}$. The choice of $T_{th,DVFS}$ is a balance between the granularity of mitigation and the overhead on execution time. If $T_{th,DVFS}$ is too long, the DVFS might not be able to react to the variability faster than $T_{th,DVFS}$. On the other hand, if $T_{th,DVFS}$ is too short, the DVFS might switch too frequently so that the switching time significantly lengthens

Algorithm 4 Run-time Scheduling for Multiple Knobs

1: **Input**: thread nodes $TN_{1...N}$, buffer size B
Output: DVFS & HDP modes which each TN uses $m_{DVFS,1...N}$, $m_{HDP,1...N}$
2: $CB_{1...N} \leftarrow$ cycle budgets of $TN_{1...N}$
3: $DL_{i...N} \leftarrow OptimalSchedule(CB_{1...N})$
4: Initialize an empty buffer buf
5: **for** $i \leftarrow 1$ to B **do**
6: $S_i \leftarrow ScenarioDetect(TN_i)$
7: Push S_i to buf
8: **end for**
9: **for** $i \leftarrow 1$ to N **do**
10: $DVFS_triggered \leftarrow CheckDvfsTriggered()$
11: $f_{mr} \leftarrow RefinedUpperBound_MT(buf)$
12: $f_{opt} \leftarrow LikelyPrediction_MT(buf, DVFS_triggered)$
13: $f_{res} \leftarrow max(f_{mr}, f_{opt})$
14: **if** $DVFS_triggered$ **or** $i = 1$ **then**
15: $m_{DVFS,i} \leftarrow$ the nearest faster DVFS mode of f_{res} assuming using fastest HDP mode
16: **else**
17: $m_{DVFS,i} \leftarrow m_{DVFS,i-1}$
18: **end if**
19: $m_{HDP,i} \leftarrow$ the nearest faster HDP mode of f_{res} assuming using $m_{DVFS,i}$
20: Execute TN_i under mode $m_{DVFS,i}, m_{HDP,i}$
21: Pop S_i from buf
22: **if** $i + B \leq N$ **then**
23: $S_{i+B} \leftarrow ScenarioDetect(TN_{i+B})$
24: Push S_{i+B} to buf
25: **end if**
26: **end for**

the execution time, as illustrated in the example of Figure 4.1 in Section 4.1. In our approach, we set $T_{th,DVFS}$ to 10 ms, so that the DVFS switching time overhead is less than 1% of the execution time. The principle of our approach can be extended to any knobs of any timescales.

In principle, the procedure of deciding modes for multiple knobs is compatible with any algorithm. Nevertheless, the DS-based algorithm can calculate a tight upper bound of execution time, so it can exploit the benefits of multiple knobs much better than other algorithms. Algorithm 4 shows the detailed flow of the run-time procedure of the DS-based algorithm for multiple knobs. In Line 10,

Algorithm 5 Refined Upper Bound for Multiple Knobs

```
1:  procedure RefinedUpperBound_MT(buf)
2:      T_cur ← current time
3:      f_max ← maximum frequency
4:      N ← number of TNs in buf
5:      DL_{1...B} ← deadlines of TNs in buf
6:      WC_{1...B} ← worst cycles of scenarios of TNs in buf
7:      T_{sw,DVFS} ← DVFS switching time
8:      T_{th,DVFS} ← period between two DVFS switching

9:      // Propagate deadlines
10:     PDL_B ← DL_B
11:     for i ← B downto 1 do
12:         PDL_{i-1} ← min(PDL_i − WC_i/f_max, DL_{i-1})
13:     end for

14:     // Find the last TN of DVFS interval
15:     j ← 1
16:     while PDL_j < T_cur + T_{th,DVFS} + 2T_{sw,DVFS} do
17:         j ← j + 1
18:     end while
19:     f_mr ← sum(WC_{1...j})/(PDL_j − T_cur − 2T_{sw,DVFS})

20:     // Check deadlines in this DVFS interval
21:     for k ← 1 to j do
22:         f_mr ← max(f_mr, sum(WC_{1...k})/(DL_k − T_cur − T_{sw,DVFS}))
23:     end for
24:     return f_mr
25: end procedure
```

the controller would check whether the time since the last DVFS decision point is longer than $T_{th,DVFS}$ to decide whether to decide a new DVFS mode.

In Line 11, the controller calls the enhanced refined upper bound procedure. The detail of the procedure is shown in Algorithm 5. A major change is that the controller would consider that the DVFS mode must run over multiple TNs until the next DVFS switch, as illustrated in Fig. 4.15. Therefore, to generate the case of the most positive slack, the processor is assumed to run in a certain DVFS mode for $T_{th,DVFS}$, and then switch to the fastest DVFS mode until the end of the buffer. The minimum speed requirement (f_{mr}) is derived based on this case, as shown in Line 15–19. In addition, the controller also derives the

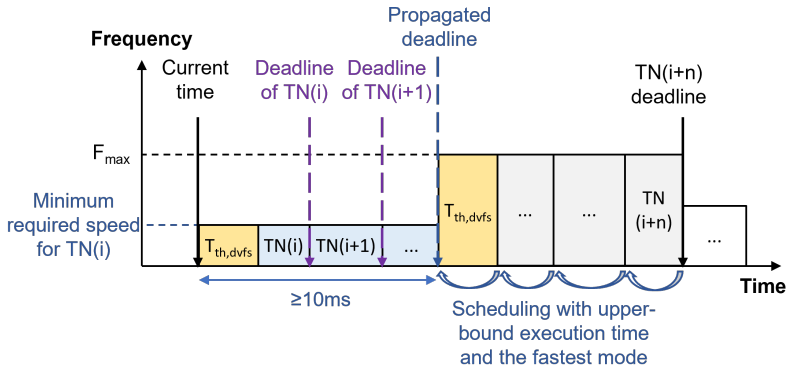


Figure 4.15: Enhanced refined upper bound process to calculate the minimum required speed for the next TN for ensuring timing guarantee in the multiple-knob scheduling algorithm

Algorithm 6 Likely Prediction for Multiple Knobs

1: **procedure** LIKELYPREDICTION_MT(buf, $DVFS_triggered$)
2: $T_{cur} \leftarrow$ current time
3: $B \leftarrow$ number of TNs in buf
4: $DL_{1...B} \leftarrow$ deadlines of TNs in buf
5: $AC_{1...B} \leftarrow$ average cycles of scenarios of TNs in buf
6: $T_{sw,DVFS} \leftarrow$ DVFS switching time
7: **if** $DVFS_triggered$ **then**
8: $f_{opt} \leftarrow sum(AC_{1...B})/(DL_B - T_{cur} - T_{sw,DVFS})$
9: **else**
10: $f_{opt} \leftarrow sum(AC_{1...B})/(DL_B - T_{cur})$
11: **end if**
12: **return** f_{opt}
13: **end procedure**

speed requirements to meet all deadlines before the next DVFS switch. Based on all of these requirements, the controller derives the lowest speed which can ensure all timing guarantees, as shown in Line 21–23.

In Line 12, the controller calls the enhanced likely prediction procedure, which considers DVFS switching time in calculation. The detail of the process is shown in Algorithm 6. Finally, in Line 14–18, the controller would only decide the new DVFS mode when DVFS is triggered. Otherwise, the current DVFS

mode will be used for the next TN. To ensure the timing guarantees, when deciding on the new DVFS mode, the controller assumes the fastest HDP mode is used. Once the DVFS mode is decided, the new HDP mode will be decided based on the condition of the selected DVFS mode.

4.4.6 Overheads of the Algorithm

Running the algorithm could induce execution time overhead. The amount of time overhead depends on the hardware implementation of the system. If the algorithm runs in parallel with the original workload, then most of the execution time overhead can be hidden. Normally this means the algorithm is run in a distinct hardware, such as the power management unit or a dedicated processor core.

If the algorithm and the original workload share the same hardware resource (e.g. they run in the same processor core), then the algorithm costs extra execution time at the beginning of each TN of the original workload. The influence of this time overhead is similar to how the switching time of the knobs affects the time overhead and the time granularity of the mitigation, as discussed in Section 3.2. The total time overhead of the approach for each TN is actually the switching time of the knobs plus the execution time of the algorithm. In order to minimize the performance degradation and improve the time granularity of the mitigation, both the switching time and the execution time of the algorithm should be small compared to the original workload. Since our approach operates at the time granularity of the deca-microsecond level, the execution time generating a decision in the algorithm should be at most hundreds of nanoseconds, approximately a thousand clock cycles for a normal processor of several GHz clock frequency.

Therefore, the algorithm design is restricted by the maximum execution time requirement. The scenario detector and the buffer size are the two main factors which affect the execution time of the algorithm. Although a complicated scenario set and scenario detector can reduce the errors of system cost prediction and bounding, they increase the execution time overhead. Also, a larger buffer size considers more deadline constraints and thus requires more execution time. Hence, the complexity of the scenario detector and the size of the buffer should be limited to meet the execution time constraint. A typical implementation uses several scenarios, a scenario detector based on a look-up table, and a buffer size of at most 20, which are lightweight that the execution time is negligible compared to the execution time of a TN.

4.5 Proof of Timing Guarantees

In the previous work of the conceptual framework of dynamic scenario principles, a mathematical model was presented, where the problem of timing-guaranteed scheduling was formulated as a set of optimization functions and constraints [133]. Our proposed algorithm is instantiated from the conceptual framework, in the context of sub-millisecond-level mitigation of highly dynamic workloads, on the processor equipped with speed-varying knobs like HDP and DVFS. In the following, we will show how the algorithm is transferred from the mathematical model, while the hard constraints are fulfilled in the proposed algorithm. Hence, we prove that the algorithm intrinsically ensures timing guarantees in our specific context.

4.5.1 Summary of the Mathematical Model

The mathematical model assumes typical deterministic scheduling for a number of consecutive jobs with uncertain execution cycles and hard deadlines, executing on a generalized processor model of several operating modes with distinct cycle times. In a nutshell, the processor is assumed to operate in cycles, while the cycle time $CT[m]$ varies with its operating mode m. Switching from a mode m_a and another mode m_b takes switching time of at most $O(m_a, m_b)$. The processor is required to execute a sequence of jobs $J_1 \cdots J_N$, each of which has its own deadline D_i, ready time R_i, upper-bound execution cycles UC_i, and likely execution cycles LC_i. The objective of the scheduler is to find a proper start time s_i and an operating mode m_i for each job, which can ensure that all jobs finish before their deadlines even though the exact execution cycles of each job are unknown before execution. Table 4.2 enumerates the detailed denotations of the parameters in this model. Based on the model, the hard constraints required for a static schedule to ensure timing guarantees are derived as follows:

Constraint 1: The start time of a job cannot be earlier than it ready/arrival time:

$$s_i \geq R_i \tag{4.1}$$

Constraint 2: The start time of a job cannot be earlier than the expected completion time of the previous job plus switching time overhead :

$$s_{i+1} \geq s_i + LC_i \cdot CT[m_i] + O(m_i, m_{i+1}) \quad \forall i \geq 1 \tag{4.2}$$

Boundary case of the above constraint for the first job:

Processor characteristics	
M	Number of distinct operating modes provided by the processor to execute a job
$CT[1 \dots M]$	A constant array where $CT[m]$ is the effective cycle time of the processor when in mode m
$O(m_a, m_b)$	Worst-case time overhead in switching from mode m_a to mode m_b
F	Fastest mode; independent of job and processor state
m_0	Mode setting for the job prior to the first job J_1
Job characteristics	
N	Number of consecutive jobs considered in the joint optimization by the scheduler at a given scheduling instance
R_i	Ready/arrival time of job J_i $\forall 1 \leq i \leq N$; execution of a job cannot be started before this time even if the processor is free
D_i	Hard deadline of job J_i; execution of a job must be completed by this time even when the job has its worst-case workload; each job can have its own deadline
UC_i	Refined upper-bound execution cycles of J_i; scheduler invokes upper-bound refinement model at run-time for each job to get this info
LC_i	Predicted likely execution cycles of J_i (given by the estimator)
$LC_i \cdot CT[m]$	Likely duration of J_i if executed with mode m
$UC_i \cdot CT[m]$	Guaranteed maximum amount of time J_i can take if executed with mode m
LS_i	Latest (or upper-bound on) start time of J_i, i.e., if the job J_i is not started by this time one or more of deadlines $D_k, k \geq i$ might miss which is not acceptable
W	Safe mode of the job subsequent to J_N; if the processor is in mode W by the time D_N, schedulability of subsequent jobs beyond the optimization horizon is guaranteed; $W \leq F$ assuming mode with lower index is faster mode.
PCT	Completion time of the job prior to J_1; namely current time of the processor
Variables to be solved by the scheduler	
$s_1 \dots s_N$	Likely start time of job $J_1 \dots J_N$
$m_1 \dots m_N$	Likely mode setting for job $J_1 \dots J_N$

Table 4.2: Denotations of the mathematical model of scheduling with hard timing constraints

$$s_1 \geq PCT + O(m_0, m_1) \tag{4.3}$$

Constraint 3: A job must complete its execution before the deadline even for the upper-bound workload:

$$s_i + UC_i \cdot CT[m_i] \leq D_i \quad \forall i < N \tag{4.4}$$

Boundary case of the above constraint for the last job:

$$s_N + UC_N \cdot CT[m_N] \leq D_N - O(m_N, W) \tag{4.5}$$

Constraint 4: Even when the current and future jobs pose upper-bound workload, there should be enough time either (i) to switch to the fastest mode and execute all the later jobs using the fastest mode and complete them before their respective deadlines or (ii) selected modes for all jobs are such that all deadlines will be met even for the worst-case workload conditions:

$$\begin{aligned} s_i + UC_i \cdot CT[m_i] \leq \max\{\, & s_{i+1} - O(m_i, m_{i+1}), \\ & LS_{i+1} - O(m_i, F)\} \quad \forall i < N \end{aligned} \tag{4.6}$$

where

$$LS_N = D_N - UC_N \cdot CT[F] \tag{4.7}$$

$$LS_i = \min\{D_i, LS_{i+1}\} - UC_i \cdot CT[F] \quad \forall i < N \tag{4.8}$$

4.5.2 Instantiation of the Mathematical model

On the foundation of the conceptual framework, we develop the proposed algorithm for sub-millisecond-level scheduling in a processor core with DVFS and HDP. Therefore, the variables of the model are mapped into the variables in the algorithm, representing the properties of the real processors and applications. Table 4.3 lists all the relations between the variables of the model and those of the algorithm. In addition, we will elaborate on how the mapping is made as follows:

Variable in the model	Related variable in the algorithm	Note
Processor characteristics		
m	(m_{DVFS}, m_{HDP})	Each operating mode represents a pair of a DVFS mode and an HDP mode
$CT[m]$	$CT[m_{DVFS}, m_{HDP}]$	$CT[m]$ is a function of the DVFS mode and the HDP mode
$O(m_a, m_b)$	$T_{sw,DVFS}, T_{sw,HDP}$	$O(m_a, m_b)$ is related to the switching time of DVFS and HDP, defined by Equation 4.9
F	f_{max}	f_{max} denotes the fastest clock frequency of all modes, equal to $1/CT[F]$
m_0	(n/a)	m_0 is considered to be unknown
Job characteristics		
N	B	The number of jobs to be scheduled is the same as the buffer size B
J_i	TN_i	Each job represent a thread node
R_i	(n/a)	The constraint of the ready time is always fulfilled. Hence, R_i is not required in the algorithm
D_i	DL_i	Hard deadline of TN_i
UC_i	WC_i	UC of each TN is the worst-case execution cycles of the scenario
LC_i	AC_i	LC of each TN is the average execution cycles of the scenario
LS_i	PDL_i	LS_i, PDL_i respectively denote the latest start time and the latest end time for J_i. Their relation can be expressed by Equation 4.15
W	(n/a)	The algorithm should guarantee the TNs subsequent to TN_B cam meet their deadlines regardless of the mode setting of m_B. Hence, the safe mode W is not required in the algorithm
PCT	T_{cur}	Current time of the processor
Variables to be solved by the scheduler		
$s_1 \dots s_N$	$s_1 \dots s_B$	Start time of $TN_1 \dots TN_B$
$m_1 \dots m_N$	$(m_{DVFS,i}, m_{HDP,i})$, $1 \leq i \leq B$	Mode setting for $TN_1 \dots TN_B$

Table 4.3: Relations between the variables in the math model and in our proposed algorithm

- Each operating mode m represents a pair of a DVFS mode and an HDP mode (m_{DVFS}, m_{HDP}). Therefore, $CT[m]$ corresponds to the clock period when the processor is running under this combination of DVFS mode and HDP mode, which can be denoted as $CT[m_{DVFS}, m_{HDP}]$

- $O(m_a, m_b)$ is dependent on whether DVFS mode switching or HDP mode switching is involved. Assuming the worst DVFS switching time is $T_{sw,DVFS}$ and the worst HDP switching time is $T_{sw,HDP}$, then $O(m_a, m_b)$ can be denoted as follow:

$$O(m_a, m_b) = \begin{cases} 0 & \text{if } m_{DVFS,a} = m_{DVFS,b} \\ & \text{and } m_{HDP,a} = m_{HDP,b} \\ T_{sw,DVFS} & \text{if } m_{DVFS,a} \neq m_{DVFS,b} \\ & \text{and } m_{HDP,a} = m_{HDP,b} \\ T_{sw,HDP} & \text{if } m_{DVFS,a} = m_{DVFS,b} \\ & \text{and } m_{HDP,a} \neq m_{HDP,b} \\ T_{sw,DVFS} + T_{sw,HDP} & \text{otherwise} \end{cases} \tag{4.9}$$

  In our specific case, $T_{sw,HDP} = 0$ is assumed, since HDP switching time is negligible as compared to the time granularity of scheduling. Hence, $O(m_a, m_b)$ can be simplified as follow:

$$O(m_a, m_b) = \begin{cases} 0 & \text{if } m_{DVFS,a} = m_{DVFS,b} \\ T_{sw,DVFS} & \text{if } m_{DVFS,a} \neq m_{DVFS,b} \end{cases} \tag{4.10}$$

- In the algorithm, f_{max} denotes the fastest clock frequency of all modes. Hence, it can be defined as follows:

$$f_{max} = \frac{1}{CT[F]} \tag{4.11}$$

- The number of jobs to be scheduled (N) equal to the size of the buffer (B), in which each job J_i represents a thread node TN_i. Not all TNs have hard deadlines, since the end of a TN can be merely an arbitrary point in the middle of the application. However, as mentioned in Section 4.4.4, at the start of the algorithm, an optimal schedule is drafted, in which provisional deadlines are given for all TNs without hard deadlines, serving as checkpoints of the workload process. These deadlines become additional constraints that need to be fulfilled besides the original hard deadlines. In the mathematical representation, the deadlines $DL_1, \ldots DL_B$ include both original deadlines and provisional deadlines.

- A thread node TN_i should be always ready at the end time of the previous thread node TN_{i-1}. Hence, Constraint 1 $s_i \geq R_i$ is always fulfilled. In practice, this means that all input data required for TN_i is either already stored in the processor core before TN_1 starts or provided by the computation results of previous TNs. There is no need to wait for the arrival of the data from the external components.

- In the context of dynamic scenarios, UC_i, LC_i are determined by the scenario to which TN_i belongs. In our approach, the UC and LC of each scenario are the worst and the average execution cycles of this scenario, denoted by WC_i and AC_i. These numbers are acquired from design-time process, as described in Section 4.4.2. By definition, the following statement is always true:

$$WC_i \geq AC_i \quad \forall 1 \leq i \leq B \tag{4.12}$$

- In the algorithm, PDL_i denotes the latest end time for TN_i to ensure that the deadlines of TN_i and the following TNs can always be met. It is the counterpart of LC_i with respect to the start time and the end time of a job. Hence, similar to LC_i, PDL_i can be defined as follows:

$$PDL_B = DL_B \tag{4.13}$$

$$PDL_i = \min\{PDL_{i+1} - \frac{WC_{i+1}}{f_{max}}, DL_i\} \quad \forall i < B \tag{4.14}$$

  Line 10–13 of Algorithm 5 implements the calculation process of PDL_i. It is also worth noting that the following relation between LS_i and PDL_i can be easily proved by mathematical induction:

$$PDL_i = LS_i + \frac{WC_i}{f_{max}} \quad 1 \leq i \leq B \tag{4.15}$$

- The TNs and mode settings before TN_1 are considered to be unknown, and the scheduler should ensure the timing guarantees regardless of the mode setting prior to TN_1. Hence, the switching time at the start of the first TN ($O(m_0, m_1)$) is defined as the maximum switching time ($T_{sw,DVFS}$) for safe scheduling.

- For the same reason as above, for the TNs subsequent to TN_B, the scheduler should guarantee that they can meet their deadlines regardless

of the mode setting of TN_B. Hence, we assume that the switching at the end of TN_B occurs after D_B, and there is no need for the requirement of the safe mode W. Mathematically it is equivalent to defining $O(m_N, W) = 0$.

To avoid ambiguity, in the following sections, we will adopt the denotations of the algorithm in the proof of timing guarantees, except for $CT[m], O(m_a, m_b)$, and F, which do not have equivalent denotations in the algorithm.

Based on the variable mapping as above, the hard constraints of timing guarantees in Section 4.5.1 can be translated in the context of the algorithm as follows:

Constraint 1: The start time of a TN cannot be earlier than it ready time. As discussed above, this constraint should be always fulfilled in our specific context.

Constraint 2: The start time of a TN cannot be earlier than the likely time of the previous TN plus switching time overhead:

$$s_{i+1} \geq s_i + AC_i \cdot CT[m_i] + O(m_i, m_{i+1}) \quad \forall i \geq 1 \tag{4.16}$$

Boundary case of the above constraint for the first job:

$$s_1 \geq T_{cur} + T_{sw,DVFS} \tag{4.17}$$

Constraint 3: A TN must end before its deadline even for the worst-case execution cycles:

$$s_i + WC_i \cdot CT[m_i] \leq DL_i \tag{4.18}$$

Constraint 4: Even when the current and future TNs pose the worst-case execution cycles, there should be enough time either (i) to switch to the fastest mode and execute all the later TNs using the fastest mode and complete them before their respective deadlines or (ii) selected modes for all jobs are such that all deadlines will be met even for the worst-case execution cycles:

$$s_i + WC_i \cdot CT[m_i] \leq \max\{\, s_{i+1} - O(m_i, m_{i+1}), \\ PDL_{i+1} - \frac{WC_{i+1}}{f_{max}} - O(m_i, F)\} \quad \forall i < B \tag{4.19}$$

4.5.3 Proof of Hard Constraints

Instead of the static schedule theorized in the model, our approach uses a dynamic schedule, which changes at run time over the progress of the workload and is updated at the start of each TN. To ensure timing guarantees, the dynamic schedule should always fulfil the hard constraints within the buffer no matter how the schedule changes. In our proposed approach, at the start of every job, the Refined Upper Bound process of Algorithm 5 is invoked to formulate a schedule that strictly fulfils the hard constraints. With this schedule, we can derive the available modes for the next TN that can ensure no deadline misses for the next job and the following TNs in the buffer. Then, the next TN is executed with one of these modes until the end of the TN when the schedule is updated again. In this way, we can ensure that every mode setting for each TN is based on a schedule that fulfils the hard constraints, so the timing guarantees are ensured at any time of the workload progress.

The schedule formulated by the Refined Upper Bound process is composed of two phases, illustrated in Figure 4.15. In the first phase, the workload runs under the mode m_A, in which the speed is equal to or greater than the minimum required speed f_{mr}. Then, for the rest of the TNs, the fastest mode F is used. At the start of both phases, mode switching occurs with the worst-case switching time $T_{sw,DVFS}$. Mathematically, the schedule can be expressed as follows:

- Assuming the first phase contains $TN_1 \dots TN_K$ and the second phase contains jobs $TN_{K+1} \dots TN_B$. Then, the mode setting for all jobs and the switching time overhead between jobs can be expressed as follows:

$$m_i = \begin{cases} m_A & 1 \leq i \leq K \\ F & i > K \end{cases} \tag{4.20}$$

$$O(m_i, m_{i+1}) = \begin{cases} T_{sw,DVFS} & \text{if } i = 0, K \\ 0 & \text{otherwise} \end{cases} \tag{4.21}$$

- In the algorithm, the interval between two DVFS switches must be at least $T_{th,DVFS}$. The boundary of the two phases is defined by this constraint. To ensure the duration of the first phase is at least $T_{th,DVFS}$, the first TN in the second phase (J_{K+1}) is defined as the first TN whose start time is at least $T_{th,DVFS} + T_{sw,DVFS}$ later than the start time of TN_1. Since from Equation 4.28 we know that $s_{K+1} = PDL_K$, the location of TN_K is constrained by the following relations:

$$PDL_{K-1} < T_{cur} + T_{th,DVFS} + 2T_{sw,DVFS} \leq PDL_K \tag{4.22}$$

The process of calculating the TN index K is implemented in Line 15–18 of Algorithm 5.

- f_{mr} denotes the minimum required speed for the first phase that can still ensure the timing guarantees of all TNs. Based on Line 19–23 of Algorithm 5, the minimum required speed can be expressed as follows:

$$f_{mr} = \max_{1 \leq i \leq K} \left\{ \frac{\sum_{j=1}^{i} WC_j}{DL_i - T_{cur} - T_{sw,DVFS}}, \frac{\sum_{j=1}^{K} WC_j}{PDL_K - T_{cur} - 2T_{sw,DVFS}} \right\} \tag{4.23}$$

- Based on the definition of f_{mr}, the speed of the selected mode for the first phase (m_A) must be equal to or faster than f_{mr}. This principle is implemented in Line 11–20 of Algorithm 4, in which the algorithm always selects a mode faster than f_{mr} if possible. Hence, the following constraint holds:

$$CT[m_A] \leq \frac{1}{f_{mr}} \tag{4.24}$$

- In the first phase, each TN starts right after its previous TN ends, assuming each TN requires its own worst-case execution cycles. Therefore, the start time of these TNs can be expressed as follows:

$$s_1 = T_{cur} + T_{sw,DVFS} \tag{4.25}$$

$$s_i = s_{i-1} + WC_{i-1} \cdot CT[m_A] \quad 1 < i \leq K \tag{4.26}$$

From the above equations we can derive the start time of the TNs as follows:

$$s_i = T_{cur} + T_{sw,DVFS} + CT[m_A] \sum_{j=1}^{i-1} WC_j \quad 1 < i \leq K \tag{4.27}$$

- In the second phase, each TN starts at the time of the propagated deadline of the previous TN, which can be expressed as follows:

$$s_i = PDL_{i-1}, \quad K < i \leq N \tag{4.28}$$

In summary, the start time of each TN is defined in Equation 4.25, 4.27, 4.28, and the mode setting of each TN is defined in Equation 4.20. Next, we will show that the schedule always fulfils the hard constraints described in Section 4.5.2, and thus the timing guarantees of the algorithm are proven. The proof of Constraint 1 is dismissed since the TN is always considered ready when the previous TN ends, as mentioned in Section 4.5.2. The proofs of other constraints are shown as follows:

Proof of Constraint 2: The constraint can be divided into four different cases: (i) the boundary case of Equation 4.17 (ii) $1 \leq i < K$ (iii) $i = K$ (iv) $K < i < B$. Each of these cases will be proven respectively as follows:

(i) With Equation 4.25, the constraint is automatically fulfilled:

$$s_1 = T_{cur} + T_{sw,DVFS} \geq T_{cur} + T_{sw,DVFS} \tag{4.29}$$

(ii) When $1 \leq i < K$, from Equation 4.20 and 4.21, we get that $m_i = m_A$ and $O(m_i, m_{i+1}) = 0$. Based on these relations and Equation 4.26, 4.12, we can prove the constraint by the following derivation:

$$\begin{aligned} s_{i+1} &= s_i + WC_i \cdot CT[m_A] \\ &\geq s_i + AC_i \cdot CT[m_i] + O(m_i, m_{i+1}) \end{aligned} \tag{4.30}$$

(iii) When $i = K$, from Equation 4.23, 4.28, 4.24, and 4.27, we can acquire the following relations respectively:

$$f_{mr} \geq \frac{\sum_{j=1}^{K} WC_j}{PDL_K - T_{cur} - 2T_{sw,DVFS}} \tag{4.31}$$

$$PDL_K = s_{K+1} \tag{4.32}$$

$$\frac{1}{CT[m_A]} \geq f_{mr} \tag{4.33}$$

$$\sum_{i=1}^{K} WC_i = \sum_{i=1}^{K-1} WC_i + WC_K = \frac{s_K - T_{cur} - T_{sw,DVFS}}{CT[m_A]} + WC_K \tag{4.34}$$

Then from the above relations we can derive:

$$\frac{1}{CT[m_A]} \geq f_{mr} \geq \frac{s_K - T_{cur} - T_{sw,DVFS} + WC_K \cdot CT[m_A]}{CT[m_A](s_{K+1} - T_{cur} - 2T_{sw,DVFS})} \tag{4.35}$$

$$s_{K+1} - T_{cur} - 2T_{sw,DVFS} \geq s_K - T_{cur} - T_{sw,DVFS} + WC_K \cdot CT[m_A] \tag{4.36}$$

$$s_{K+1} \geq s_K + WC_K \cdot CT[m_K] + T_{sw,DVFS} \tag{4.37}$$

$$s_{K+1} \geq s_K + AC_K \cdot CT[m_K] + O(m_K, m_{K+1}) \tag{4.38}$$

(iv) When $K < i < N$, from Equation 4.20 and 4.21, we get that $m_i = F$ and $O(m_i, m_{i+1}) = 0$. Based on this relation and Equation 4.14, 4.28, 4.12, we can prove the constraint by the following derivation:

$$PDL_{i-1} \leq PDL_i - WC_i \cdot CT[F] \tag{4.39}$$

$$s_i \leq s_{i+1} - WC_i \cdot CT[F] \tag{4.40}$$

$$s_{i+1} \geq s_i + WC_i \cdot CT[F] \tag{4.41}$$

$$s_{i+1} \geq s_i + AC_i \cdot CT[m_i] \tag{4.42}$$

Proof of Constraint 3: The constraint can be divided into four different cases: (i) $1 \leq i < K$ (ii) $i = K$ (iii) $K < i < B$ (iv) the boundary case of Equation 4.5. Each of these cases will be proven respectively as follows:

(i) When $1 \leq i < K$, from Equation 4.23, 4.24, and 4.27, we can acquire the following relations respectively:

$$f_{mr} \geq \frac{\sum_{j=1}^{i} WC_j}{DL_i - T_{cur} - T_{sw,DVFS}} \tag{4.43}$$

$$\frac{1}{CT[m_A]} \geq f_{mr} \tag{4.44}$$

$$\sum_{j=1}^{i} WC_j = \frac{s_{i+1} - T_{cur} - T_{sw,DVFS}}{CT[m_A]} \tag{4.45}$$

From the above relations we can derive:

$$f_{mr} \geq \frac{s_{i+1} - T_{cur} - T_{sw,DVFS}}{CT[m_A](DL_i - T_{cur} - T_{sw,DVFS})} \tag{4.46}$$

$$f_{mr} \geq \frac{f_{mr}(s_{i+1} - T_{cur} - T_{sw,DVFS})}{DL_i - T_{cur} - T_{sw,DVFS}} \tag{4.47}$$

$$DL_i - T_{cur} - T_{sw,DVFS} \geq s_{i+1} - T_{cur} - T_{sw,DVFS} \tag{4.48}$$

$$DL_i \geq s_{i+1} = s_i + WC_i \cdot CT[m_i] \tag{4.49}$$

(ii) When $i = K$, from Equation 4.14 and 4.28 we can derive:

$$DL_K \geq PDL_K = s_{K+1} \tag{4.50}$$

Then, from Equation 4.37 we can further derive:

$$DL_K \geq s_K + WC_K \cdot CT[m_K] + T_{sw,DVFS} \tag{4.51}$$

$$DL_K \geq s_K + WC_K \cdot CT[m_K] \tag{4.52}$$

(iii) When $K < i < N$, from Equation 4.14 we can derive:

$$PDL_i - \frac{WC_i}{f_{max}} \geq PDL_{i-1} \tag{4.53}$$

$$PDL_i \geq PDL_{i-1} + \frac{WC_i}{f_{max}} \tag{4.54}$$

$$DL_i \geq PDL_{i-1} + \frac{WC_i}{f_{max}} \tag{4.55}$$

From Equation 4.28 and 4.20 we know that $PDL_{i-1} = s_i$ and $m_i = F$, so we get:

$$DL_i \geq s_i + WC_i \cdot CT[m_i] \tag{4.56}$$

(iv) When $i = B$, from Equation 4.13 and 4.14 we can derive:

$$PDL_B - \frac{WC_B}{f_{max}} \geq PDL_{B-1} \tag{4.57}$$

$$PDL_B \geq PDL_{B-1} + \frac{WC_B}{f_{max}} \tag{4.58}$$

$$DL_B \geq PDL_{B-1} + \frac{WC_B}{f_{max}} \tag{4.59}$$

From Equation 4.28, 4.20, we know that $PDL_{B-1} = s_B$ and $m_B = F$, and in Section 4.5.2 we define $O(m_B, W) = 0$. Hence, we can acquire:

$$DL_B \geq s_B + WC_B \cdot CT[m_B] + O(m_B, W) \tag{4.60}$$

Proof of Constraint 4: In the process of the proof of Constraint 2, from Equation 4.30, 4.37, and 4.41, we can observe that the following statement is true for all $i < B$:

$$s_{i+1} \geq s_i + WC_i \cdot CT[m_i] + O(m_i, m_{i+1}) \tag{4.61}$$

Hence, we can derive:

$$\begin{aligned} s_i + WC_i \cdot CT[m_i] &\leq s_{i+1} - O(m_i, m_{i+1}) \\ &\leq \max\{s_{i+1} - O(m_i, m_{i+1}), PDL_{i+1} - \frac{WC_{i+1}}{f_{max}} - O(m_i, F)\} \end{aligned} \tag{4.62}$$

Since we have proven that all hard constraints are fulfilled in the dynamic schedule kept in the algorithm, we prove that the approach can ensure the timing guarantees.

4.6 Applicability to other architectures and technologies

Although the presented approach is implemented on a single-core processor, the principles of multi-timescale knobs and dynamic scenarios are generic. Therefore, they can be instantiated to develop mitigation approaches of other architectures. Each architecture has its unique knobs, such as core mapping in multi-core processors and thread mapping in GPUs. Nevertheless, in Section 4.2 & 4.3 we have shown the method of exploring time-energy trade-offs and switching overheads of the knobs. The principles of the method can be used on the other knobs to know their capability and limitation to be exploited in mitigation approaches. Once the knobs are explored, the DS principles described in Section 4.4.1 & 4.4.3 can be instantiated to develop a mitigation approach that can tackle the architecture-specific variability sources, such as cache protocol in multi-core processors and thread management in GPUs.

Moreover, the principles of the presented approach are not specific to any technology, so they can be applied to different technologies. To transfer the approach to a new technology, one should explore the variety of the devices and circuits of the technology which can result in performance-energy trade-offs, such as voltage scaling in CMOS technology. The trade-offs can be utilized to design knobs. With a variety of knobs of different ranges and switching time, the mitigation approach can be developed on the same principles.

In summary, the instantiation of the multiple-knob and DS principles to other architectures and technologies is a whole new research direction that is yet to be explored. Hence, research should be performed in the future to close the gap of timing guarantees for modern complicated architectures and advanced technologies.

4.7 Summary

The workload-dependent multi-timescale mitigation approach is developed for ensuring timing guarantees of highly dynamic workloads. The proposed approach improves the way to derive the upper bound of execution time at run time with the DS-based algorithm. Furthermore, it incorporates both the HDP and DVFS knobs to push the timescale of mitigation down to the sub-millisecond level.

Chapter 5

Workload-Dependent Multi-Timescale Mitigation Approach for Performance Variability: Experiments

In this chapter, we experiment on the effectiveness of the workload-dependent multi-timescale mitigation approach for performance variability, with a cycle-accurate simulation flow we have developed. The targeted applications for the approach are highly dynamic workloads with strict real-time requirements. The state-of-the-art approaches fail to provide practically usable full guarantees for these workloads, so we consider them the most suited benchmarks. We select two representative workloads from widely different application domains. One is the lower sub-band quantization block of the ADPCM-encoding application from TACLeBench [11]. The other one is the JPEG-decoding application from MiBench [146]. Both workloads are typical real-time signal and data processing applications, where the input of workload is a sequence of frames. Deadlines for processing consecutive frames are periodic, where the interval between two consecutive deadlines is called *frame period*.

In conventional real-time scheduling, the finest unit is the job of processing a single frame, but in our approach, more fine-grained mitigation can be achieved by cutting the job of processing a frame into multiple TNs, each of which has an execution time of tens to hundreds of microseconds. The execution time of these workloads is highly dynamic with different input data. Therefore, they

are representative of the highly dynamic workloads that our approach targets.

This chapter is structured as follows: Section 5.1 elaborates on the experiment setup, in which the cycle-accurate simulation flow is introduced. Then, Section 5.2 and 5.3 discuss the case studies with the workloads of lower-band quantization of ADPCM encoding and JPEG-decoding. The comparison of the results of the case studies is discussed in Section 5.4. The original work presented in this chapter has been published in the authored work of [136, 137].

5.1 Experimental Setup

We develop a simulation flow which can simulate the entire process of mitigation approaches, including both the design-time and run-time processes. In the simulation flow, cycle-accurate processor simulation of execution time and energy consumption is used. The details of the simulation flow are elaborated in the following sections.

5.1.1 Processor Simulation Flow

Accurate simulation of the execution time and energy consumption of the processor is required in many steps of the simulation of the DS-based methodology, such as the design-time profiling and the run-time workload benchmarking. Therefore we established a cycle-accurate processor simulation flow for this purpose, which is shown in Figure 5.1.

In the processor simulation flow, the energy consumptions of different HDP modes are simulated separately. To accurately simulate the energy consumption of HDP, first we create the physical design of each ALU of the HDP. The circuit design and physical design of the HDP knob are already detailed in Section 4.2. Then, each ALU of the HDP has gone through the entire digital design flow, including logic synthesis and place & route, to generate the full physical design. The netlists of the physical designs are further used in gate-level simulation to obtain detailed activities inside the HDP when running a workload.

Then, we perform RTL-level simulation on the RTL design of the entire processor core, running the target workload. Execution time results are acquired in this step in cycle accuracy. Meanwhile, the input vectors of the HDP are extracted during the simulation. Then, gate-level simulation is performed on the netlist of each ALU of the HDP. The input vectors from the RTL simulation are applied to the ALU design in the simulation to acquire the activities of each net in the ALUs. The activities record every rising and falling transition of each net

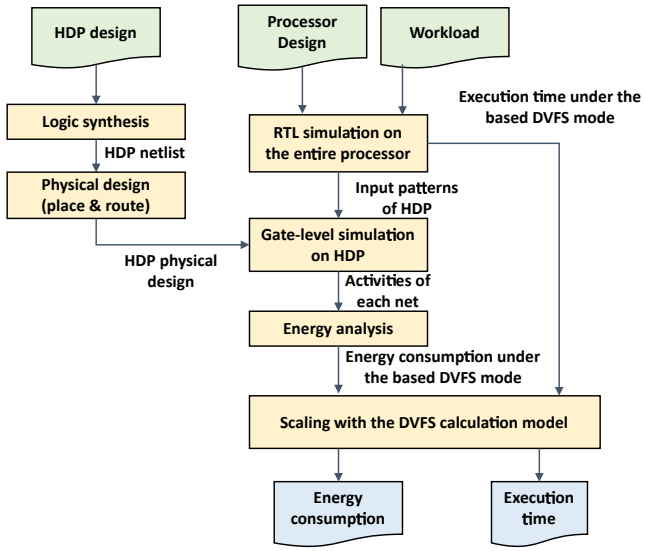


Figure 5.1: The processor simulation flow of benchmarking the execution time and HDP energy consumption for running a certain workload

during the workload at the picosecond level. Afterwards, these activities are used by the commercial power analysis tool to compute the energy consumption of the ALUs.

In our energy simulation, the time and dynamic energy overheads of HDP switching are not incorporated, since the HDP switching time is only tens of cycles, which is only about 0.1% of execution time in our sub-millisecond mitigation context, so it would only have negligible influence on time and energy. Moreover, the energy overheads from the idle ALUs are also very small compared to the energy of the active ALU. Even though the duration of power-gating can be as short as tens of microseconds, it is still thousands of times longer than the break-even point, which is approximately ten cycles [141]. Therefore, the energy consumption of the idle ALUs is also left out of the analysis.

On the other hand, the timing and energy consumption of different DVFS modes are calculated from the ones of the base DVFS mode with a timing and energy calculation model. We create five DVFS modes, with voltages ranging from 0.5V to 0.9V. The clock frequency of each mode is calculated by static timing analysis on the ALU design, while the energy consumption of each mode is calculated by scaling from the energy consumption of the 0.7V mode, which is the base mode of our DVFS knob. The energy scaling is based on the V^2

Voltage (V)	Frequency (GHz)			Energy*(nJ)		
	Fast ALU	Mid ALU	Slow ALU	Fast ALU	Mid ALU	Slow ALU
0.9	4.67	4.46	3.33	22.37	19.01	16.07
0.8	4.24	3.98	2.97	17.68	15.02	12.69
0.7 (base mode)	3.69	3.39	2.46	13.53	11.50	9.72
0.6	2.80	2.08	1.60	9.94	8.45	7.14
0.5	1.79	0.98	0.79	6.90	5.87	4.96

* The actual energy consumption is workload-dependent by the cycle-accurate simulation flow. The numbers shown here are an example of simulation with a JPEG-encoding workload.

Table 5.1: Summary of DVFS model

scaling relation according to Equation 3.2. Here, we do not consider static energy, since in simulation we found it smaller than 3% of the dynamic energy due to all the leakage reduction techniques (high-V_{th} library, power gating). A summary of the DVFS model is shown in Table 5.1, and the speed-energy relations of all available DVFS and HDP modes are shown in Figure 4.8. In addition, we can realistically assume that the switching time of DVFS is a constant of 100 μs regardless of the modes the processor switches from and to. This switching time is representative of the empirical numbers measured from modern processors [25].

5.1.2 Overall Simulation Flow

Figure 5.2 shows the overall simulation flow for the experiments of the mitigation approaches. Both the design-time scenario clustering and the run-time scheduling parts in the approach are simulated. During the simulation of design-time scenario clustering, we profile the workload with different input files to acquire the statistics of execution time and energy, using the processor simulation flow described in the last section. Then, based on the execution flow of the workload, the profiling traces are segmented into TNs of tens to hundreds of microseconds in length. Once the execution time and energy of all TNs are acquired, we analyze the correlation between these data and application variables to search for suitable monitoring parameters for scenario clustering. Scenarios are generated based on the monitoring parameters, and the statistics of each scenario are calculated and stored in a look-up table. The execution flow analysis, TN segmentation, discovery of monitoring parameters, and scenario clustering are all performed manually.

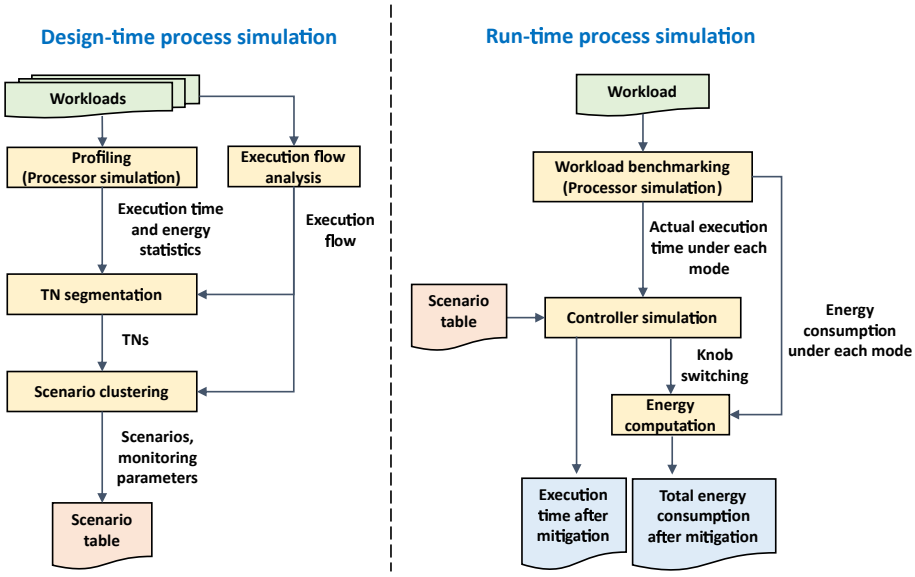


Figure 5.2: The cycle-accurate simulation flow to acquire the execution time and energy consumption when running a workload under mitigation

During the simulation of run-time scheduling, arbitrary inputs are selected for testing the algorithm. Then, we benchmark the workload with the inputs to acquire their execution time and energy under different modes of the knobs, using the processor simulation flow described in the last section. By applying the inputs of different frame contents, performance variability is injected, leading to differences in execution time for processing each frame. The exact execution time of each TN is captured by the processor simulation. These results are read by a controller simulator in which the adaptive control algorithm is implemented in the Perl language. With the scenario table and the benchmarking results, the simulator derives the knob-switching events issued by the controller along the entire workload. The results of these knob-switching events are combined with the execution time results and the energy analysis results to calculate the execution time and total energy consumption after mitigation.

In the experiment, we compare six run-time mitigation approaches of different algorithms and knobs. In terms of the algorithms, we select three different configurations, each of which is representative of an algorithm type: best-effort (BE), WCET-based, and DS-based. The BE algorithms are represented by the algorithm of Pering et al. [110], in which the prediction of execution cycles is the exponential moving average of the executed cycles of the past frames. Though

this algorithm cannot ensure timing guarantees, it represents the algorithm of the best energy consumption one can achieve. WCET-based algorithms are represented by the cycle-conserving algorithm of Pillai et al [120]. This configuration represents the conventional algorithm for timing guarantees.

In terms of the knobs, we have two different configurations. The first one is the single-DVFS configuration, which only has the DVFS knob. The only ALU in the processor is assumed to be the same as the fast ALU in the HDP. This configuration represents the conventional approach which only has the DVFS knob and one datapath designed for high speed. The BE and the WCET-based algorithms of this configuration switch DVFS modes only at the beginning of each frame, regardless of the TNs. On the other hand, the DS-based algorithm can switch at the start of each TN, while still following the restriction that the interval between two DVFS switches must be longer than $T_{th,DVFS}$. The second one is the multi-timescale configuration, which uses both the DVFS and the HDP knob. This configuration represents the proposed approach. For BE and the WCET-based algorithms, their multi-timescale versions are implemented according to the flow in Fig. 4.14. Therefore, they can switch the knobs at the start of each TN. Still, their execution time predictions are frame-based and are unaware of the run-time situation, so they cannot fully take advantage of the finer timescale. Combining different configurations of the knobs and the algorithms, we have six different approaches for comparison. By comparing the results of different configurations, we can decouple the impacts from two different features of our methodology: either from the knobs or from the algorithm.

5.2 Case Study I: Lower-Band Quantization of ADPCM Encoding Application

Adaptive Differential Pulse Code Modulation (ADPCM) is an audio wave encoding method which is widely used in speech codec, featuring sound compression with variable quantization steps. Figure 5.3 shows the block diagram of the encoding progress of the G.722 standard of ADPCM [19]. In a nutshell, the algorithm takes frames of audio signal samples as the inputs, and generates frames of compressed bit streams of the audio signals as the outputs. In the encoding process, the audio signals are first separated into features high-frequency and low-frequency sub-band signals. For both sub-bands, an adaptive predictor predicts the incoming samples based on the previous samples. The real incoming samples are compared with the predicted samples, and the error of prediction is recorded and then compressed by an adaptive quantizer. The quantized values of prediction errors of both sub-bands are then aggregated to be the output stream.

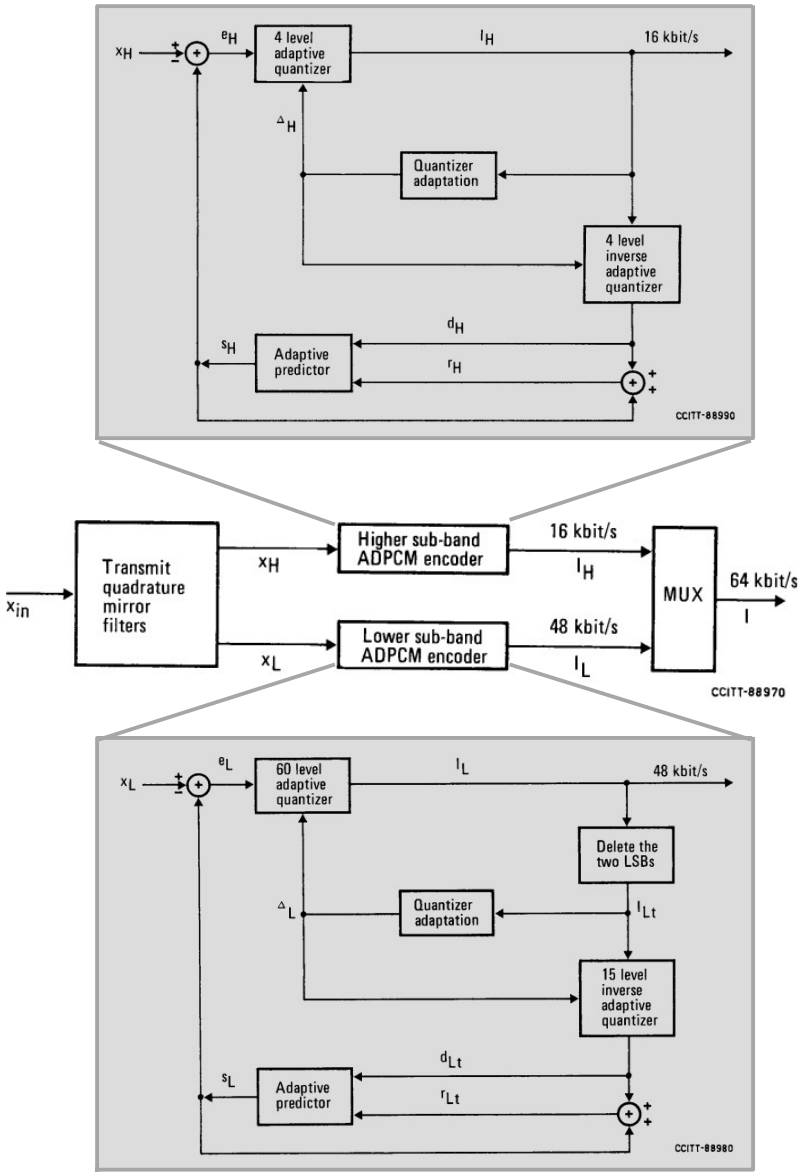


Figure 5.3: Block diagram of the encoding process of ADPCM G.722 standard [19]

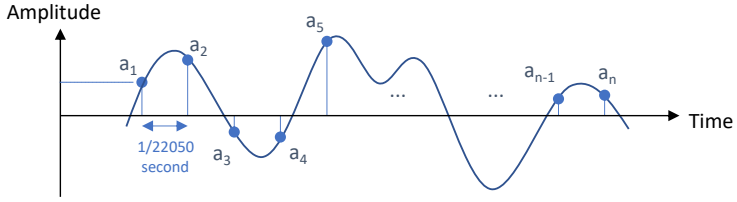


Figure 5.4: Illustration of the samples of an acoustic wave of 22050 Hz, used as the input data of the workload of the lower sub-band quantization block of ADPCM-encoding

Here, we assume a workload of the lower sub-band quantization block of ADPCM-encoding. Though the original application includes the entire ADPCM encoding flow, we only experiment on the quantization block of the low sub-band, which is the most dynamic part of the flow, to maximize the dynamism of the workload. It is feasible in practice to run the quantization block alone since one can easily modify the ADPCM encoding application to parallelize the quantization part from the other parts. Technically, we simulate the entire ADPCM encoding application but only record the execution time and energy consumption of the lower sub-band quantization block. The execution time and energy consumption of different frames are summed up to represent the case that the quantization parts are run alone. The inputs of the application are audio wave samples of 22050 Hz sample rate, as illustrated in Figure 5.4. The frame size of the audio stream is 2000 samples.

5.2.1 Design-Time Scenario Clustering

First, we profile the lower-band quantization workload with the inputs of six different audio files of 10-second long to obtain the statistics of the execution time and energy consumption of the workload. These audio files are segments of music songs of different genres. Then, we divide the profile traces in a way that each TN represents the processing of 100 samples, which means that processing one frame requires 20 TNs. In this way, the execution time of each TN is around tens to hundreds of microseconds. Figure 5.5 shows the distribution of the number of instructions of all profiled TNs and the WCET condition from program flow analysis. From the distribution, we can observe that the workload is highly dynamic. In addition, we calculate the WCET by profiling a modified workload, which always goes through the maximum loop bounds of all loops in the execution flow. The WCET is also shown in Figure 5.5. It clearly shows that the number of instructions of the WCET condition is much greater than

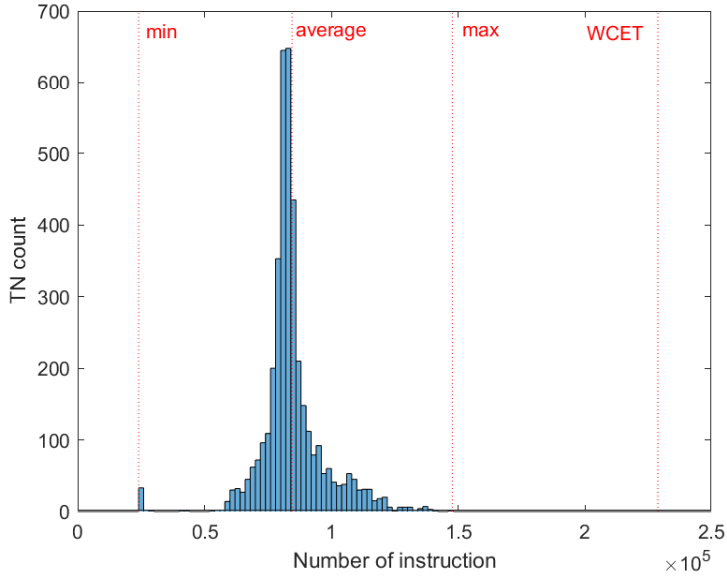


Figure 5.5: Distribution of the instruction count of the TNs of the ADPCM quantization workload, in which the great difference between WCET and realistic cases are shown

the maximum number of instructions we observe. Therefore, it demonstrates that WCET is too pessimistic to represent the real execution time.

In scenario clustering, we use the *volume* as the monitoring parameter. Assuming the amplitudes of all acoustic wave samples processed in a TN are denoted by $a_1, a_2, ..., a_n$, then the volume of this TN is defined as the equation below:

$$Volumn = \sum_{i=1}^{n} |a_i| \tag{5.1}$$

In the profiling results, the volume is shown to have a high correlation to the number of instructions, as shown in Figure 5.6, which makes the volume suitable to be the monitoring parameter. Then, six scenarios are created manually based on the understanding of the workload. Each scenario corresponds to a different range of the volume. Scenario 1 corresponds to the TNs of zero volume, a special case in which all samples are zero. The zero-volume TNs are frequent in the quiet periods of the audio file, which justifies the need for a specific scenario

Scenario	Volume range	Avg cycle count	Max cycle count
1	0	42276	42276
2	$1 - 10^4$	56180	76640
3	$10^4 - 10^5$	100451	124669
4	$10^5 - 4{\times}10^6$	147584	169068
5	$4{\times}10^6 - 7{\times}10^6$	164788	206888
6	$>7{\times}10^6$	202253	265514

Table 5.2: Scenario Look-up Table for the ADPCM Quantization Application

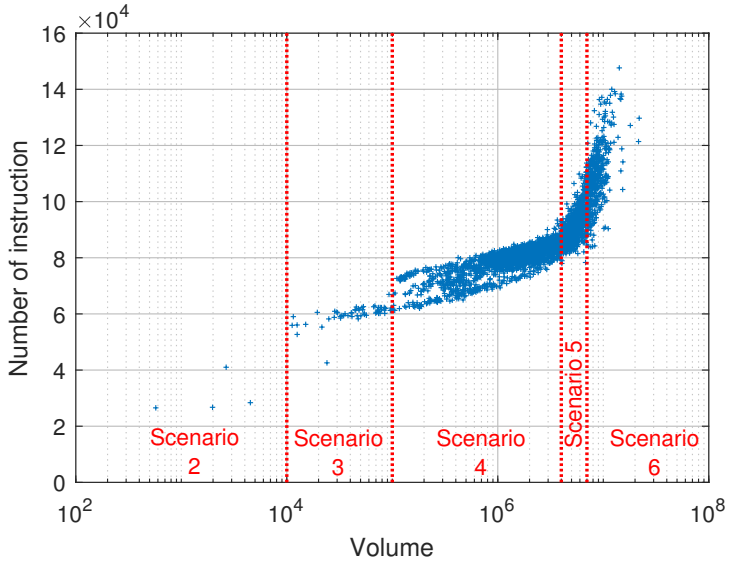


Figure 5.6: The relation between the volume and the number of instructions for each TN. TNs are clustered into scenarios by volume.

for these TNs. Scenario 2 and 6 are designed to include the TNs of extremely low and high volume respectively. Scenario 3–5 include the TNs of normal volume ranges. They are separated in a way that the difference in the number of instructions in each scenario is limited by approximately 3×10^{-4}. After the scenarios are created, the look-up table of the statistics of these scenarios is generated, shown in Table 5.2.

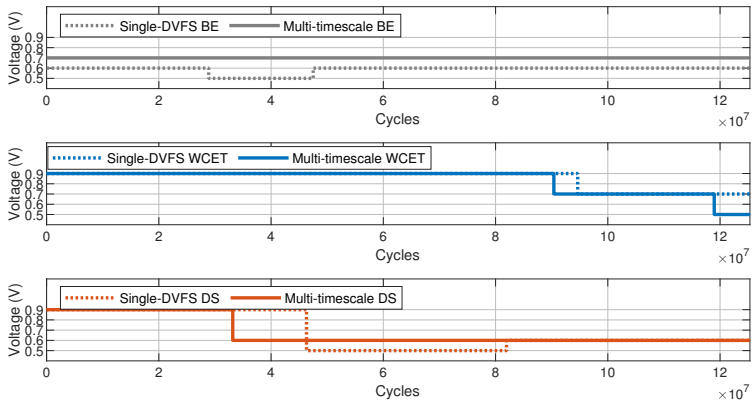
5.2.2 Run-Time Mitigation Results

At run time, we selected an arbitrary audio wave of a music song with 500 samples and repeated it ten times to form the input of the workload. By applying the input, the performance variability is injected, in which the execution flow of processing each frame depends on the content of the samples. As shown in Figure 5.5, the different input contents lead to variability in execution time, and the exact execution time cannot be calculated before execution. It requires 1000 TNs to run the whole workload, and the total execution time is 34 ms under the 0.7V DVFS mode and the fast HDP mode. For each TN encountered at run time, the controller calculates the volume of the TN and then references the look-up table (Table 5.2) to find the scenario of the TN. As a result, the controller obtains the average cycle count and the upper-bound cycle count of the TN from the table. These cycle count data are used by the scheduling algorithm (Algorithm 4) to derive the suitable modes to run each TN in the workload.

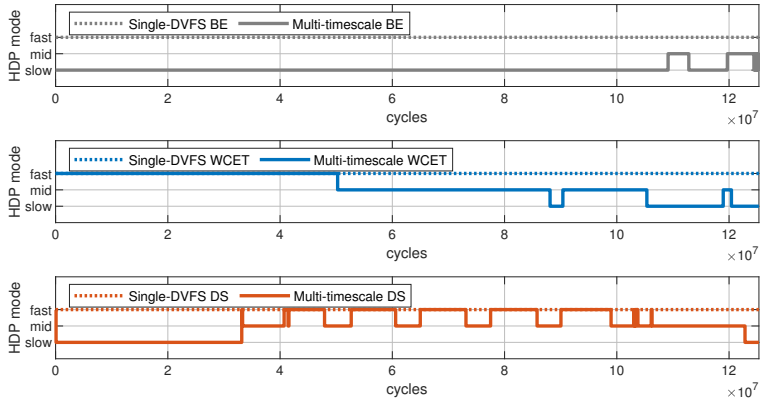
Figure 5.7 shows the selected DVFS and HDP modes during the entire workload of the 1.0-ms frame period case, and Figure 5.8 shows the traces of slack of all mitigation approaches. The slack is calculated as the amount of time by which a TN ends before its deadline. The optimal behaviour of the slack trace is to always be positive and as close to zero as possible, which means that the processor runs at the slowest possible speed without any deadline violations. For the BE approaches, the slack is close to zero all along the workload, which is near-optimal for energy saving. However, as shown in Figure 5.8, their slacks can sometimes be negative, which means potential deadline violations. Therefore, these methods cannot ensure timing guarantees.

For the approaches that can ensure timing guarantee (WCET and DS), we identify three phases of mitigation over the entire workload from the traces of slack. First, at the beginning of the workload, fast modes are used to accumulate slack to ensure a safe margin to avoid deadline violations. Then, when sufficient slack is secured to ensure timing guarantees, the processor runs at a speed at which the slack keeps at a stable level. In this phase, the processor switches among neighbouring modes to fine-tune the running speed or react to small variability. Finally, when the workload is about to finish, the processor exploits the slack to save energy by running in slow modes.

While all four curves generally follow the three phases of mitigation, their behaviours are quite different. Comparing the traces of different knob configurations, we observe that the traces of the single-DVFS configurations could have sharp turns, such as the turn at about 4.6×10^{-7} cycles of the single-DVFS DS curve. These turns indicate occurrences of DVFS switching



(a)



(b)

Figure 5.7: Selected modes over the entire run of the ADPCM quantization workload with the 1.0-ms frame period. (a) DVFS mode (b) HDP mode

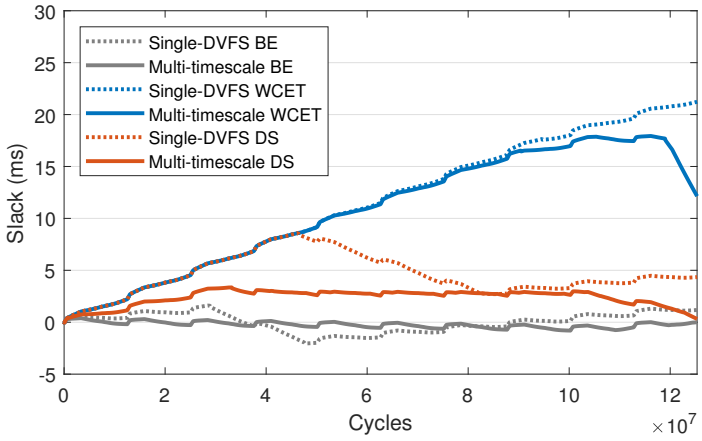


Figure 5.8: Trace of slack over the entire run of the ADPCM quantization workload with the 1.0-ms frame period

which causes dramatic changes in processor speed. As a result, the trend of slack increasing or decreasing also changes dramatically before and after the DVFS switching. It is worth noting that depending on the decisions of the algorithm, there might not be any DVFS switching during the whole workload. In this case, there would not be any sharp turns in the curve, such as the single-DVFS BE configuration in the result. However, these curves are still far from the ideal behaviour that the slack should be kept at a level that is only required for timing guarantees. These characteristics show that with the coarse-grained modes and long intervals between switchings of DVFS, the control of the slack is far from optimal.

In contrast, the traces of the multi-timescale configurations are much smoother and kept at a constant slack level most of the time. This shows that the fine-grained modes and the small timescale enabled by HDP enable the controller to precisely control the slack and immediately tackle the variability. Therefore, the workload progress can be kept at a stable slack required for timing guarantees.

Furthermore, comparing the traces of the two approaches of the DS-based algorithm shows that the multi-timescale one requires less slack to ensure timing guarantees, which results in shorter time and slower modes in the first phase. Also, the multi-timescale one starts the third phase earlier, so it can exploit more slack by the end of the workload. These behaviours show the benefits of multi-timescale mitigation in providing more energy reduction.

Moreover, the benefit of the DS-based algorithm is shown Figure 5.7a and

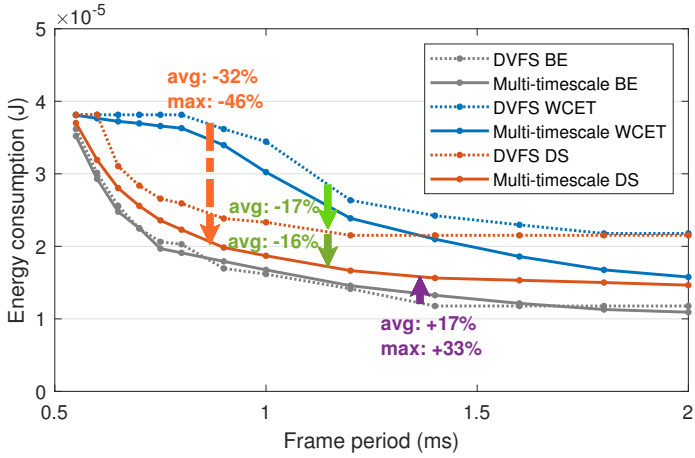


Figure 5.9: ADPCM quantization workload energy consumption of different methods under different time-criticality conditions

Figure 5.8 by comparing the results of the approaches with the single-DVFS configuration. The approach with the DS-based algorithm starts the second phase much earlier than the one with the WCET-based algorithm. This shows that the DS-based algorithm requires less slack to ensure timing guarantees due to the tight upper bound of execution time. The smaller slack indicates that the processor runs at a slower speed, which is beneficial to energy saving. Finally, combining the benefits of both features, the proposed approach has a smooth mitigation process over the whole workload with only the least amount of slack required to be secured. Its curve is always positive and close to zero, which is near the optimal behaviours we described above. This means it is the best approach among all in keeping the processor at the slowest possible speed without violating any deadlines.

Figure 5.9 shows the energy consumption of different approaches under different time-criticality situations, represented by different frame periods. The energy differences among all approaches are more prominent in the middle of the curves. At the left end of the curves, the timing is so strict that all approaches use the fastest mode most of the time, while at the right end of the curves, the timing is so relaxed that all methods use the slowest mode most of the time. In these conditions, the choice of the mitigation approach is less relevant.

Comparing different methods, Figure 5.9 shows that the proposed approach has a energy saving of 32% on average and 46% on maximum, compared to the conventional WCET-based single-DVFS approach. Also, the benefit is

significant in all time-criticality situations. As compared to the single-DVFS BE approach, which is near-optimal for energy saving, the energy consumption of the proposed approach is larger by only 17% on average and 33% on maximum. Regarding the timing guarantees, the single-DVFS BE method misses 13% deadlines on average, while the proposed approach has zero deadline misses. Since the BE approaches cause potential deadline misses, it is not an available solution for time-critical and time-sensitive applications. For these applications, the proposed approach is the best solution, in which a small energy overhead is a reasonable price to pay to ensure the timing guarantees.

The separate contributions of each feature can also be derived from Figure 5.9. The benefit of the DS-based algorithm can be evaluated from the comparison between the single-DVFS WCET-based approach and the single-DVFS DS-based approach. It shows that the algorithm improves energy consumption by 17% on average. Its benefit is more prominent in the time-critical region, since its advantage of deriving a tighter execution time upper bound is more important in the time-critical situation. The benefit of the multi-timescale approaches using multiple knobs can be evaluated from the comparison between the single-DVFS DS-based method and the multi-timescale DS-based method. It shows the improved knob configuration saves extra 16% energy on average. Its benefit is more prominent when the timing is loose, since its advantage of allowing the processor to run in slower modes for a longer time, is more prominent in the non-time-critical situation. In conclusion, it is the combined effect of both features which makes our approach have a significant energy saving in all time-criticality situations.

5.3 Case Study II: JPEG Decoding Application

JPEG is a widely used image compression standard. A JPEG encoder can compress a bitmap digital image into a shorter bit stream, while a JPEG decoder can transform the bit stream into the original digital image with little loss in image quality. In this study, we assume the inputs of the JPEG decoding application are streams of JPEG images extracted from sample QCIF videos (174 x 144 resolution). Each JPEG image represents a frame of the video. The JPEG images are sequentially provided to the application to represent a multimedia streaming workload of a constant frame rate.

The program flow of the JPEG decoding application is shown in Figure 5.10, which generally consists of two main parts. The first part is the initialization phase, in which the application builds up the data structure and reads the input image. Then in the second part, the application decodes the image by rows of

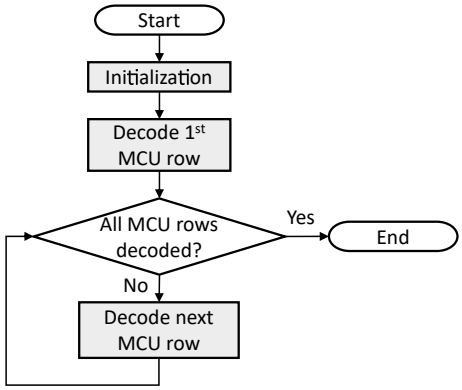


Figure 5.10: Program flow of the JPEG decoding application. In the diagram, each step (grey square block) of the program flow is defined as a TN.

Minimum Coded Units (MCU). Each MCU row is a 174 x 16 fraction of the image.

5.3.1 Design-Time Scenario Clustering

First, we profiled the JPEG decoding workload with the inputs of 12 different QCIF video streams of various contents, to obtain the statistics of the execution time and energy of the workload. The number of JPEG images in each video ranges from 150 to 494. Then, the profiling traces are segmented into TNs of hundreds of microseconds. In this case, a TN is defined as the process of initialization or decoding an MCU row. Since one image is composed of nine MCU rows, the process of decoding one image is divided into 10 TNs in this way.

Two parameters are selected as the monitoring parameters for the scenario detector. The first one is the different steps in the program flow (initialization, decoding the first MCU row, decoding a middle MCU row, decoding the last MCU row). Each of these steps exhibits distinct characteristics of execution time and energy consumption. Since the execution flow of the initialization step is the same for all images, its execution time and energy consumption are almost insensitive to the image content, which differentiates the characteristics of this step from the others. Then, in the main loop of decoding the image by MCU rows, the costs of processing the first and the last MCU rows are systematically different from the middle MCU rows. Therefore, the steps of processing the

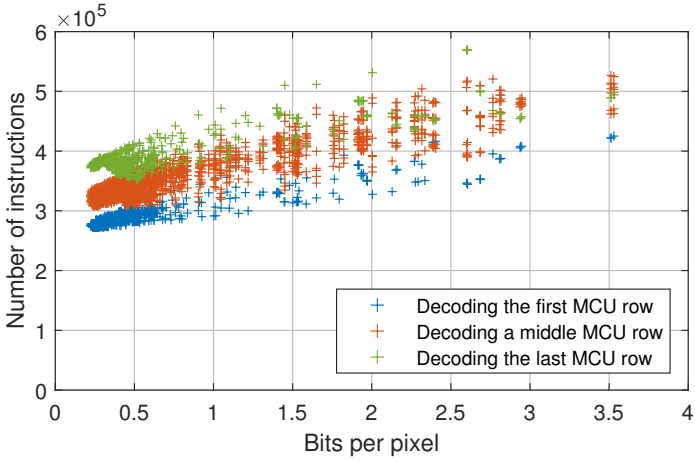


Figure 5.11: Relation between the number of instructions and the BPP for the TNs of decoding an MCU row. It shows the high influence of BPP and MCU row location on the number of instructions.

first and the last MCU rows are distinguished from the middle MCU rows to create their own scenarios.

The second monitoring parameter is the *bit per pixel* (BPP) of the image, which is defined as the length of the encoded bit stream divided by the number of pixels. BPP represents the data density of the image, which is related to the computation effort to decode an image. In the profiling, BPP is highly correlated to the execution time of decoding the MCU rows in the image. Also, the calculation of BPP is simple and can be done before reading the content of the encoded bit stream. Therefore, the controller can calculate the BPP of the upcoming TNs in the buffer to detect their scenarios. These properties make BPP a suitable monitoring parameter for scenario detection.

Figure 5.11 shows the relation between the number of instructions and the BPP for the TNs of decoding an MCU row. In general, the number of instructions increases with the BPP. Moreover, decoding the first MCU row takes more instructions, while decoding the last MCU row takes fewer instructions. In contrast, Figure 5.12 shows that the execution time of the TNs of the initialization step is almost constant regardless of the input image content and the BPP. Based on these properties, 9 dynamic scenarios are created, each corresponding to a certain step in the program flow and a BPP range. Based on the statistics of the cycle count of each scenario, the look-up table for scenario detection is created, which is shown in Table 5.3.

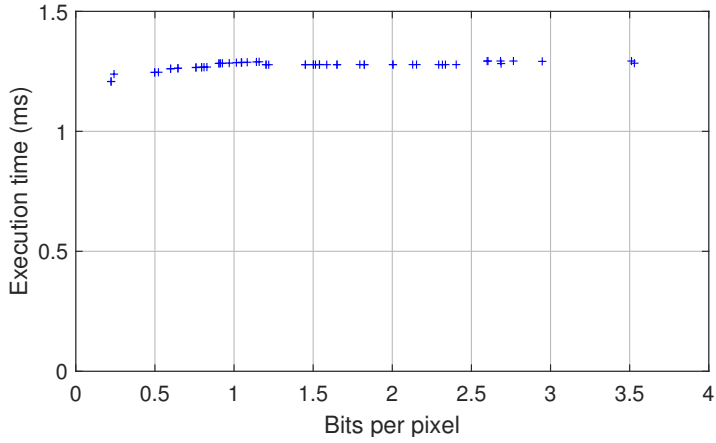


Figure 5.12: Relation between the execution time under the 0.7V/fast HDP mode and the BPP for the TNs of the initialization step. It shows the execution time is almost constant regardless of the input image content.

Monitoring parameters		**Scenario**	**Cycle count**	
Program flow	**BPP range**		**max**	**avg**
Initialization	0 – 4.00	1	4771771	4703069
Decode the first MCU row	0 – 1.35	2	1423793	1271159
	1.35 – 4.00	3	1614701	1434089
Decode a middle MCU row	0 – 1.35	4	1609520	1414888
	1.35 – 2.10	5	1864506	1569669
	2.10 – 4.00	6	2175664	1767012
Decode the last MCU row	0 – 1.45	7	1838664	1661917
	1.45 – 2.50	8	2055572	1790937
	2.50 – 4.00	9	2266653	1991362

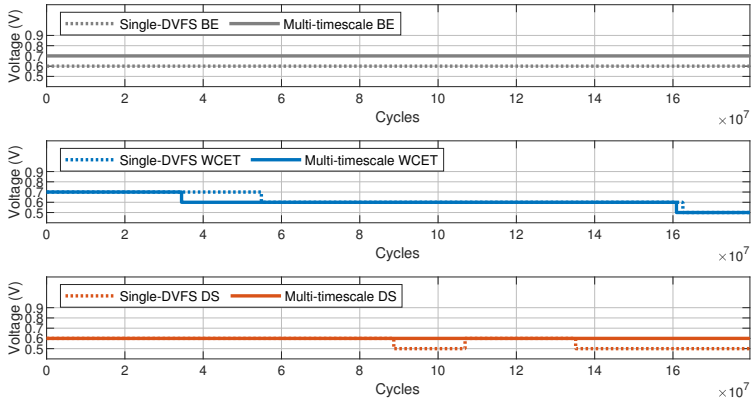
Table 5.3: Scenario Look-up Table for the JPEG-Decoding Application

5.3.2 Run-Time Mitigation Results

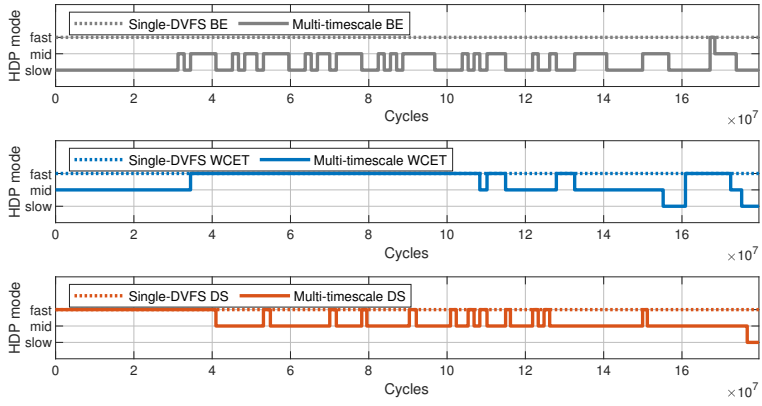
At run time, we selected an arbitrary QCIF video stream composed of 10 JPEG images as the input of the workload. By applying the input, the performance variability is injected, in which the execution flow of processing each MPU row of each frame depends on the image content, so the execution time can be different and cannot be exactly calculated before execution. It requires 100 TNs for the JPEG decoders to process the input. For each TN encountered at run time, the controller identifies which program flow step the TN is in and the BPP of the MCU row that the TN is processing. With these parameters, the controller can look up Table 5.3 to obtain the scenario of each TN and its average cycle count and upper-bound cycle count, which is used by the controller algorithm (Algorithm 4) to derive the suitable modes to run each TN in the workload.

Figure 5.13 shows the DVFS and HDP modes selected during the entire run under the 8.0-ms frame period condition, and Figure 5.14 shows the traces of slack of all approaches. The behaviours of each approach in these figures are similar to Section 5.2.2, though the workload is different. This shows that the characteristics of the approaches we discussed in Section 5.2.2 are not specific to a certain application. They are universal across workloads. Under a different workload, the multi-timescale DS-based approach can still run in a smooth slack curve which is always positive and close to zero. This proves that the merits of this approach are general and independent of workload.

Figure 5.15 shows the energy consumption of different approaches under different frame periods. Compared to the conventional single-DVFS WCET-based approach, the multi-timescale DS-based approach can save 16% energy on average and 25% energy on maximum. And compared to the DVFS BE approach, the energy consumption of the multi-timescale DS-based approach is about the same and 5% greater in the worst case. In some cases, the multi-timescale DS-based approach consumes even less energy, which is 6% in the best case. This result does not mean that the proposed approach can ensure timing guarantee without an energy penalty, since the energy consumption can be affected by multiple factors, such as the energy efficiency of the selected modes and the activities in the workload. Nevertheless, it shows that in some contexts the energy penalty can be minimal. Also, the energy differences among the approaches are smaller than in Case I in Section 5.2.2, which indicates that the JPEG-decoding workload is less dynamic, so the potential of energy saving by mitigation is smaller, further showing that the energy saving achieved by mitigation is highly workload dependent.



(a)



(b)

Figure 5.13: Selected modes over the entire run of the JPEG decoding workload with a 8.0-ms frame period. (a) DVFS mode (b) HDP mode

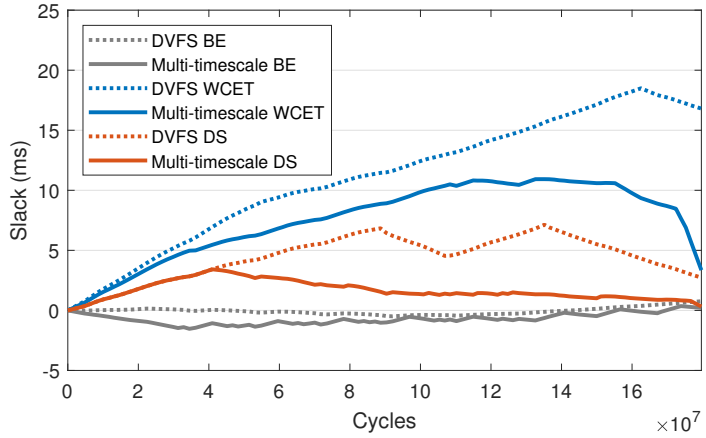


Figure 5.14: Trace of slack over the entire run of the JPEG decoding workload with a 8.0-ms frame period

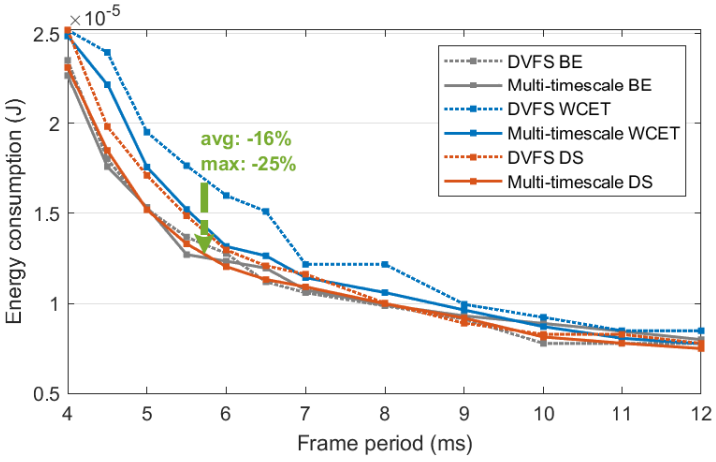


Figure 5.15: JPEG decoding workload energy consumption of different methods under different time-criticality conditions

Approach	Case I: Quantization of ADPCM encoding		Case II: JPEG decoding	
	Deadline miss rate	Energy ratio* (min/avg/max)	Deadline miss rate	Energy ratio* (min/avg/max)
Single-DVFS BE	13%	0.47/0.59/0.95	52%	0.74/0.84/0.93
Multi-timescale BE	29%	0.49/0.58/0.92	46%	0.72/0.85/1.00
Single-DVFS WCET	0%	1.00/1.00/1.00	0%	1.00/1.00/1.00
Multi-timescale WCET	0%	0.72/0.90/1.00	0%	0.82/0.91/0.99
Single-DVFS DS	0%	0.66/0.83/1.00	0%	0.80/0.89/1.00
Multi-timescale DS	0%	0.54/0.68/0.97	0%	0.75/0.84/0.92

* Energy ratio is calculated w.r.t. the energy of the single-DVFS WCET method.

Table 5.4: Mitigation result summary with different methodologies and benchmarks

5.4 Discussion of the Case Studies

We summarize the mitigation results of both benchmarks under all frame periods in Table 5.4, in which the average energy ratio is calculated with respect to the energy of the DVFS BE-based approach. The differences in the results of both benchmarks may be attributed to several factors. First, the degree of dynamism of the workload could affect the performance of mitigation approaches. A workload which is more dynamic in execution time, like Case I, could provide more potential for mitigation benefits. This explains why the energy consumption differences of methodologies are more prominent in Case I. Second, for the BE algorithm, the deadline miss rate varies between benchmarks, depending on the differences between the prediction and the real execution time. On the other hand, the WCET-based algorithm and the DS algorithm can both achieve zero deadline misses regardless of the benchmark.

5.5 Summary

We compare our proposed approach with conventional BE and WCET-based approaches by cycle-accurate simulation with two representative workloads (quantization of ADPCM encoding, JPEG decoding) running on the Ariane core. Results show that when running a highly dynamic workload, the proposed approach can ensure meeting all deadlines, and at the same time reduce the dynamic energy consumption by 32% on average and 46% on maximum compared to the conventional WCET-based method with a single DVFS knob.

Chapter 6

Cross-Layer Reliability and Variability Mitigation Framework: Theory

In this chapter, we further discuss how to extend the state-of-the-art mitigation approaches to tackling device- and circuit-level performance variability. To achieve this, multiple mitigation approaches need to be integrated to form a cross-layer mitigation framework which can deal with different kinds of variability and errors. In the framework, each layer tackles a type of variability or errors produced from the lower layers and generates variability or errors which should be tackled by the upper layers. The framework can not only ensure timing guarantees but also ensure functional correctness at the same time. Here, the principles and theoretical basis of the framework are elaborated. Then, Chapter 7 will discuss the experiments on testing the capability, cost, and limitations of this mitigation framework.

This chapter is structured as follows: Section 6.1 describes the motivation and overview of the cross-layer framework, which consists of three layers. Then, we further elaborate on the details of each layer respectively from the bottom to the top in Section 6.2 – 6.4. The first layer, mitigation of device-level variability (process, voltage, and short-term aging effects), is described in Section 6.2. Then, the second layer, mitigation of errors, is described in Section 6.3. After that, we describe the last layer, mitigation of performance variability, in Section 6.4. Then, Section 6.5 will describe how the overall reliability of the system can be evaluated with the cross-layer framework. Finally, we discuss

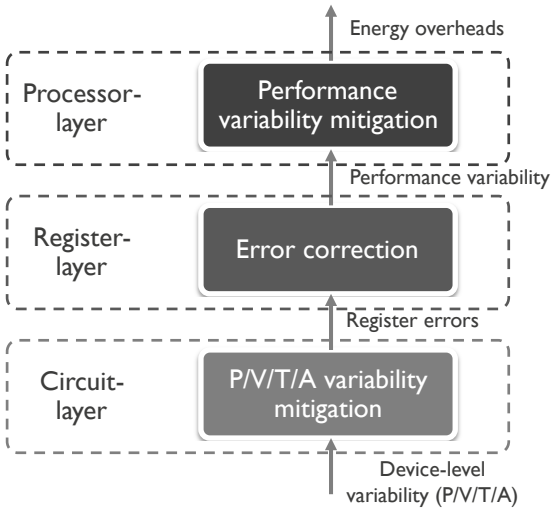


Figure 6.1: Overall structure of the cross-layer mitigation framework

some extensive questions about the framework in Section 6.6, including the comparisons with other cross-layer reliability frameworks and the applicability to other architectures and technologies. Part of the work presented in this chapter has been published in the co-authored work of [147].

6.1 Motivation and Principles

In Section 2.1, we have described variability sources which cause performance changes in devices. However, these variability sources do not directly cause changes in the execution time of workloads, since the performance of circuits is defined by the clock frequency, which is controlled by the phase lock loop, but not the intrinsic device performance. Nevertheless, the device-level variability is manifested by two effects. First, device-level variability causes timing violations, which causes errors in registers. To ensure functional correctness, error correction mechanisms need to be applied, which induces execution time and energy overhead. Second, to reduce errors, modern processors have adopted mitigation approaches for device-level variability, as described in Section 2.4.2. The knobs used in these approaches, such as throttling and adaptive clocking, introduce extra execution time. Thus, mitigation for device-level performance variability is essentially compensating for the execution time overhead indirectly caused by the devices through these error correction and reduction mechanisms.

Therefore, we propose a cross-layer mitigation framework which integrates all of the mechanisms mentioned above. The framework can ensure functional correctness and timing guarantees in presence of the errors and performance variability caused by the devices. The overall structure, shown in Figure 6.1, is composed of three layers. Each layer deals with different kinds of errors and variability at different levels.

First, in the circuit layer, run-time mitigation approaches are applied to reduce the variability of process, voltage, and short-term aging effects in devices. With these approaches, the performance fluctuation of devices is reduced, and so are timing violations and errors in registers. However, these mitigation approaches have their limits, so they cannot prevent all errors from happening. Therefore, there are still error occurrences which need to be tackled by error correction mechanisms in the upper layers. Moreover, these approaches cause performance fluctuation in the circuits, which also needs to be compensated by the performance variability mitigation approaches in the upper layers.

Second, in the register layer, error correction mechanisms fix the errors generated from the circuit layer, so functional correctness can be ensured. These mechanisms also generate execution time overhead, which contributes to performance variability. The performance variability needs to be tackled in the upper layer. The amount of time overhead increases with the number of errors required to be corrected. Hence, if we would leave out the device-level variability mitigation in the first layer, this layer could become overloaded with such overhead all the time, which is in practice too costly for the performance variability mitigation approach in the upper layer to deal with. Therefore, the first layer is essential to prevent this overloading of errors.

Last, in the processor layer, performance variability generated in the circuit and register layers is tackled by the performance variability mitigation approach, so timing guarantees can be ensured. Performance variability of the architecture level and the software level can also be tackled at the same time. The mitigation approaches operated at the cost of extra energy consumption, which is the final cost of the overall framework. The energy cost and the validity of timing guarantees depend on the amount and the frequency of performance variability. Only the fewer cases of performance variability needed to be tackled, the less energy cost and probability of failing timing guarantees. This shows the importance of the mitigation and error correction techniques in the lower two layers.

In short, the framework can ensure both functional correctness and timing guarantees in the presence of variability in device performance. The cost of the framework is only the extra energy consumption and the chip area required for the hardware of each mitigation approach.

6.2 Mitigation of Device-Level Variability

Mitigation approaches in the circuit layer aim to reduce the very fast device-level variability in process variation, voltage fluctuation, short-term aging effects, etc. The purpose is to keep the devices operating in a stable condition to minimize fluctuation in device performance. In this way, the circuit can work within its timing specification, so errors can be reduced. Though temperature fluctuation and long-term aging effects also cause variability in devices, they are normally tackled in higher layers due to their long time constants. In the following sections, we will discuss the state-of-the-art of these approaches as well as their limitations and overheads.

6.2.1 State-of-the-Art Mitigation Approaches

Voltage droop is the main target to deal with in this layer due to its nanosecond-level timescale, as described in Section 2.1.2. Voltage droop means occurrences of extraordinary drops of supply voltage that might cause errors in the circuit. It is normally caused by abrupt high demand for switching current from the devices that exceed the capability of the voltage regulation of the voltage supply module. As mentioned in 2.4.2, voltage droop mitigation approaches are proven to be effective in preventing voltage droop from producing errors.

As a representative example, here we demonstrate a typical voltage droop prevention technique. To prevent voltage droop, voltage sensors or critical path monitors are distributed across the whole chip design to detect local events of large voltage decreases. Figure 6.2 shows an example of how critical path monitors are placed inside each core of IBM POWER7™ processor [20]. The readings of the sensors and monitors are passed to a dedicated controller to decide whether a potential voltage droop event is approaching. Once a potential voltage droop event is identified, the controller intervenes to cut the switching current demand in nanoseconds, either by throttling, adaptive clocking, or halting the processor cores. In this way, further supply voltage decrease is prevented, and so are errors in the circuit. As a result, Figure 6.3 shows an example of measurements of a droop event with and without the voltage droop prevention approach. It clearly demonstrates the capability of the approach in cutting down voltage droop.

Instead of proactively eliminating variability, another concept is error-resilient design techniques [148]. In these techniques, in-situ error detection circuits reactively detect errors in the circuit, which trigger on-site microarchitectural recovery mechanisms. Normally, the recovery process will cost one to several extra clock cycles, which is called “cycle noise”. The designs of Razor [148–150]

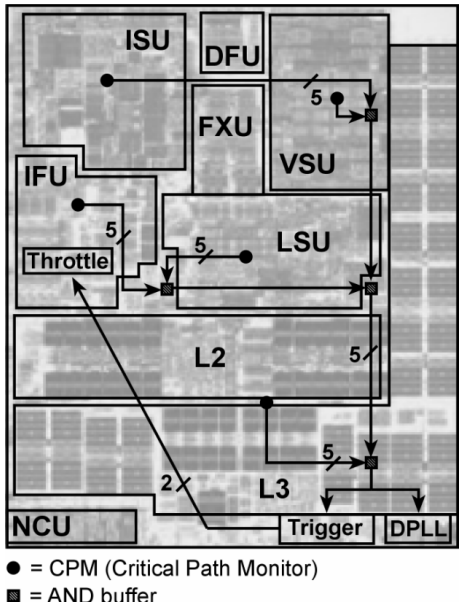


Figure 6.2: Critical path monitor distribution in each core of IBM POWER7™ processor [20]. The AND buffers are used to aggregate the monitor signals and the final signals could trigger knobs such as throttling or the digital phase-locked loop (DPLL).

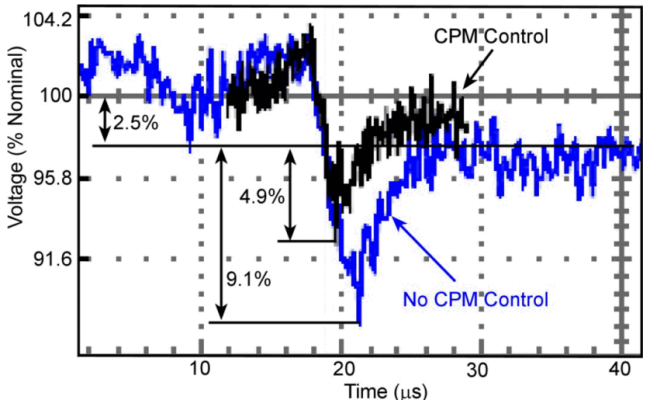


Figure 6.3: Measurement of a droop event with and without the voltage droop prevention approach [20]

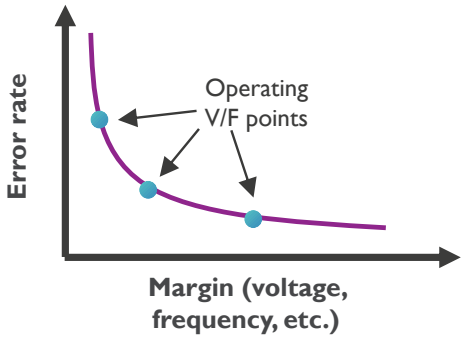


Figure 6.4: Relation between the error rate and the margin

and tunable replica circuits (TRC) [47, 75] fall into this category. However, the extra circuitry leads to area and power overheads which could reach 6.9% and 9.4% respectively [151]. The cycle noise also brings a performance loss that could reach 18% [152]. Apart from the overheads, a prominent drawback is that these techniques cannot guarantee that errors are detected under all circumstances, since the error detection is not realized by catching the wrong bits, but by sensing variability conditions which could generate errors, such as potential timing violations. Therefore, error-resilient designs are less popular than other mitigation approaches.

While mitigation approaches can effectively reduce variability to prevent errors, they have their limitations in the scale and the gradient of the variability they can tackle. As a result, guardbanding still needs to be used on top of the mitigation. However, due to the complexity of the underlying mechanisms, variability is not fully predictable, nor does it have a guaranteed bound. Therefore, the existing state-of-the-art approaches cannot guarantee to contain variability within their specifications and prevent all errors. It is still possible that a large and sharp variability exceeds the capability of mitigation and the guardband, causing errors in circuits. However, as motivated earlier in Section 6.1, the state-of-the-art circuit-level techniques proposed in the literature are still very useful and even essential to prevent the overload of errors in the register layer which will be discussed later in the next section.

6.2.2 Register-Level Errors

In presence of variability, the change in device performance can exceed the timing specification of the circuit, causing timing violations. In consequence, registers in the circuit might catch incorrect bit values, which are called register-

level errors. The error rate of the registers is highly related to the imposed margin of the guardbanding technique. In general, the relation between the error rate and the margin is depicted in Figure 6.4, in which the larger the margin, the smaller the error rate. Due to recent trends of shrinking margins and increasing hardware complexity in advanced technologies, register-level errors are more and more frequent. Recent research in large-scale datacenters shows that silent data corruption induced by register-level errors is observed in one in thousands of processor cores [153, 154]. These errors are unpredictable and difficult to be prevented by testing in advance, so the costs of tackling them are very high.

The margin can be adjusted at run time by changing the operating conditions, such as supply voltage and clock frequency. Ideally, the circuit is best to operate at the point of the smallest margin that does not cause any error. However, it is formidable to accurately calculate this ideal operating point since it depends on numerous factors, such as operating conditions, the amount of variability, workload, etc. Therefore, modern circuits normally operate at the point with a reasonable margin at which errors are rarely observed, and error correction mechanisms are applied to ensure functional correctness. The amount of margin affects the system costs in terms of speed, power, and area. On one hand, a large margin means a high supply voltage or a slow clock frequency, which degrades circuit speed and power consumption. On the other hand, a small margin means more errors, which increases the time, energy, and circuit area invested in error correction. Therefore, the operating point of the circuit (supply voltage, clock frequency) has great influences on the system reliability (error rate, etc.) and system costs (speed, power, etc.) for ensuring reliability, so it should be carefully decided when designing the mitigation approaches for system reliability.

Estimation of the error rate is essential to evaluate the overall system costs of margining and mitigation approaches in the presence of register-level errors. However, very few studies have addressed the methods of error rate estimation in the past. Run-time measurements were performed in error-resilient design research to analyze the error rate dependency on voltage and frequency [47, 75, 149, 150]. Apart from measurements, some research has explored the idea of error rate evaluation by either design-time simulation or run-time estimation, In [155], the error rates detected by different types of monitors were compared in both simulation and experiments. In addition, the relation among the error rate of in-situ monitors, voltage, and aging was simulated. In [156, 157], measurements were performed to find the relations between real circuit failures and error flags reported by critical path monitors, in order to minimize the voltage margin at run time. In [158], error probability is evaluated by gate-level simulation with degradation-aware cell libraries based on variations of voltage, temperature,

and aging. In our work, we also simulate the error rate dependency on clock frequency later in Section 7.2.

In addition to register-level errors, mitigation approaches also bring performance variability. First, running the controller routine causes execution time overhead. Second, using system knobs to mitigate variability lengthens the execution time by increasing the number of clock cycles or reducing clock frequency. These execution time overheads need to be noted in the mitigation for performance variability in upper layers.

6.3 Register-Layer Error Correction

6.3.1 Error Correction Techniques

To ensure functional correctness, register-level errors generated in the circuit layer need to be tackled by error correction approaches. Many hardware (HW) and software (SW) approaches are developed in the past. However, most approaches either cannot provide full guarantees, or come with significant timing, power, and energy overheads. Therefore, designing a highly-reliable and low-cost error correction approach is still a challenging task. Time overhead is especially critical in time-critical and time-sensitive systems since it raises timing guarantee concerns. Hence, the time overhead needs to be compensated by the mitigation approaches for performance variability in the higher abstraction layers. However, if the error rate is too high, the time overhead could increase to unacceptable values, exceeding the capability of performance variability mitigation.

Many HW-based approaches exploit the principle of redundancy. This type of approach includes error correction circuitry (ECC) used at different memory levels [159–161]. Other HW-based approaches are developed with dual modular redundancy (DMR) or triple modular redundancy (TMR) in different hardware granularities [162–164]. However, these approaches come with large timing, power, and area overhead, which is too costly for systems which are sensitive to either of these metrics. For instance, previous works show that ECC-protected L1 SRAM could have high overheads of 15% in area [165], 70% in energy [21], or more than 100% in time [21]. Also, a processor chip with full DMR shows more than two times the area of the original processor [166]. While these overheads can be reduced by using these techniques only on circuits of critical functions, reliability is also reduced. Another HW-based concept is error-resilient designs discussed in Section 6.2.1, which also suffer from the problems of high overhead and compromised reliability.

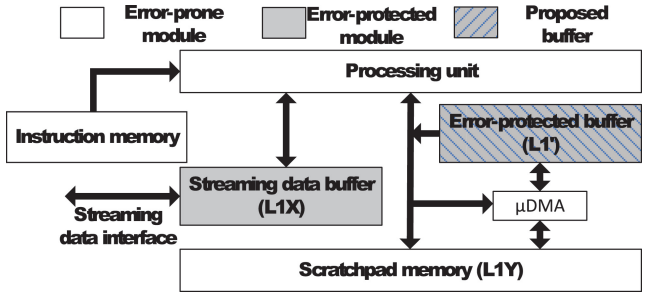


Figure 6.5: Schematic diagram of OCEAN, the optimized HW/SW reliability mitigation approach [21]

SW-based error correction approaches can be classified into techniques that apply checkpointing and rollback-recovery (CRR), or resource redundancy. In the former techniques, checkpointing is used to save the software states on a regular basis, In cases of an error, a recent checkpoint state is restored by recovery processes. While this technique has been used in plenty of works [167–170], it introduces significant overheads that could reach 70% in energy and more than 100% in execution time [21]. In cases of high error rates, the recovery events can even overflood the system, so that the time overhead is prohibitive to be mitigated.

In the resource redundancy approaches, both temporal and spatial redundancy can be exploited, As for temporal redundancy, workloads can be duplicated in the same core before their results are used as the source operands of the follow-up instructions [167, 171]. As for spatial redundancy, workloads are duplicated in a nearby idle core [167]. The granularity of the duplication can vary from the instruction level [171] to the process level [167]. However, the operation of these techniques relies on spare resources such as idle time and cores, which are not always available. Therefore, their reliability cannot be fully guaranteed.

To sum up, most of the proposed error correction techniques face the dilemma of introducing significant overheads or degrading reliability. Recovery-based approaches tend to have high time overheads, while redundancy-based approaches tend to have high area and energy overheads. To reduce overheads, many approaches turn to partially protected circuits or data, which compromise reliability.

6.3.2 HW/SW Reliability Mitigation with OCEAN

To overcome the drawbacks and limitations mentioned in the last session, the concept of integrated HW/SW error correction has been proposed [172, 173]. Cross-layer error correction design can combine and optimize the error-resilient approaches in different abstraction layers, such as propagating information on reliability from the HW level to the SW level to reduce cost. Although cross-layer designs are promising to ensure reliability with minimized total overheads, their challenge lies in deciding the proper techniques and granularities which can globally optimize the interlayer communications and the holistic costs. Sub-optimal solutions will lead to one optimized parameter (e.g., time) at the expense of another suboptimal parameter (e.g., energy).

To have a low-overhead error correction approach, our framework adopts OCEAN, the optimized HW/SW reliability mitigation approach [21], developed by Sabry et al. It provides a generalized framework for cross-layer optimization, achieving optimal error correction. A hybrid HW/SW mitigation approach is realized with this framework. In this approach, the HW reliability is enhanced by using an error-protected buffer, as shown in the schematic in Figure 6.5. The buffer is utilized to protect a portion of the processed data used to restore from run-time errors. By optimizing the buffer size, energy overhead can be minimized under timing and area constraints. In addition, a hardware module (referred to by μDMA) resembling the direct memory access function is dedicated to the transfer between the memory module and the buffer, further removing the latency in data movement.

In the SW layer, OCEAN applies the checkpointing and rollback-recovery (CRR) technique. An example is shown in Figure 6.6. In this approach, each workload is segmented into computation phases. The collection of computation results generated in each computation phase is called *data chunk*. At the end of each computation phase, a checkpoint is inserted to check the errors in the data chunk with error detection techniques such as signature comparison. If no error is detected, the data chunk is considered error-free, and it is copied to the error-protected buffer as a backup by the μDMA. Meanwhile, the main program can proceed to the next computation phase. The main program can run in parallel with the copy of the data chunk, so the time overhead is minimized.

On the other hand, if errors are detected by the checkpoint routine, rollback-recovery occurs, as illustrated in P3 of Figure 6.6. First, a rollback routine is performed, in which the main program is recovered to the state of the start of the current phase, and the data chunk of the last phase is recovered by copying back from the error-protected buffer. Then, the main program reruns the current phase. At the end of this phase, a new checkpoint is inserted to

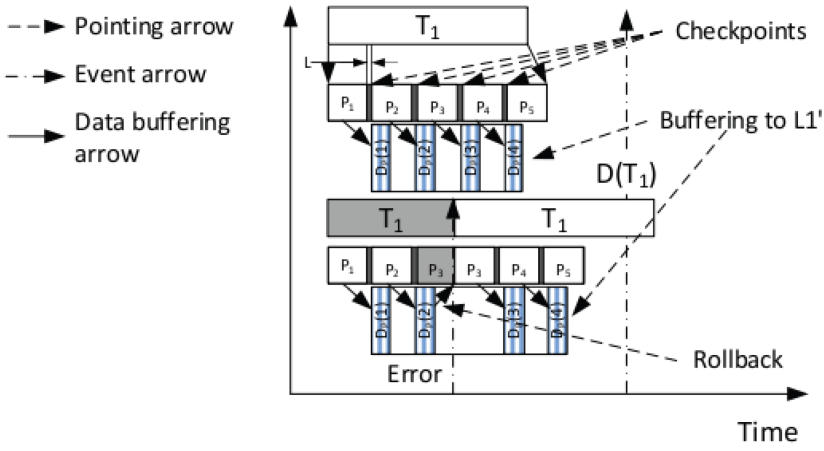


Figure 6.6: A CRR example of dividing a task T1 to five phases (P1-P5), showing the impact on error mitigation. The faulty task/phase is shaded with light grey, the checkpoints are shaded with dark grey, and the data chunks are shaded in stripe-patterned blue [21]

check the errors in the recomputed data chunk. The phase can roll back and recompute repeatedly until the data chunk is proven correct by the checkpoint routines. Therefore, there is no bound for the number of rollbacks required for each phase.

In experimental results, OCEAN achieves full error mitigation with 10.1% average energy overhead compared to the baseline operation that does not include any error correction capability, and 65% energy saving compared to a cross-layer error mitigation mechanism without optimization [21].

6.3.3 Theoretical Analysis of Time Overhead

Our work further extends the use of OCEAN to be integrated into the cross-layer reliability and variability mitigation framework. In order to have a global view of the impact of the time overhead of error correction on timing guarantees, it is crucial to analyze the interplay between errors and timing guarantees. Therefore, we derive the relations between errors and time overhead under the CRR mechanism. Moreover, the limit within which the system can ensure reliability can be estimated by this analysis. Our analysis in this section is workload-independent, assuming a static error rate and a constant size of each computation phase. We will further perform workload-dependent analysis with variable segment sizes in Section 7.3.

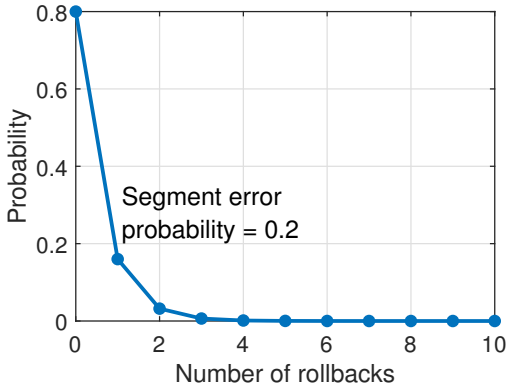


Figure 6.7: Probability distribution of the number of rollbacks of a segment, assuming the error probability of the segment is 0.2

In our error model, a cycle is erroneous if any register of a pipeline stage contains a wrong value. The probability that a cycle is erroneous is static over time, which represents the effect of variability in logic circuits during the execution of the application. Assuming a certain segment of the workload is concerned, the probability that there is no error during this segment is as follows:

$$Pr(N_e{=}0) = (1-p)^{C_{seg}} \tag{6.1}$$

where N_e is the number of erroneous cycles, C_{seg} is the total number of cycles in this segment, and p is the probability of a cycle to be erroneous, respectively. The number of rollbacks for each segment follows Geometric distribution, derived with Equation 6.1 as follows:

$$Pr(N_{rb}{=}n_{rb}) = (1-(1-p)^{C_{seg}})^{n_{rb}}\,(1-p)^{C_{seg}} \tag{6.2}$$

Figure 6.7 shows the probability distribution of the number of rollbacks of a segment, assuming the error probability of the segment to be 0.2. The distribution shows that multiple rollbacks are possible but highly unlikely. Moreover, by the probability distribution of Equation 6.2, we can derive the expected number of rollbacks as follows:

$$E[N_{rb}] = \frac{1-(1-p)^{C_{seg}}}{(1-p)^{C_{seg}}} \tag{6.3}$$

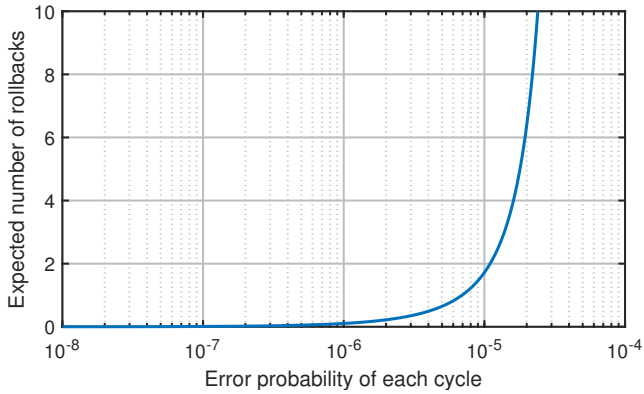


Figure 6.8: Expected number of rollbacks for a segment of 10^5 cycles, under different error probability of cycles

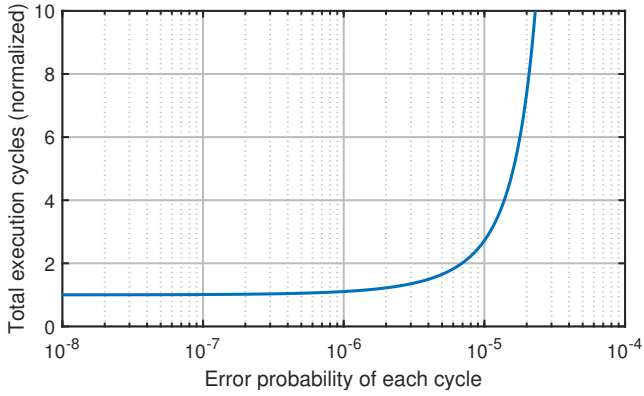


Figure 6.9: The expected value of the total execution cycles normalized to the original execution cycles for the segment (10^5 cycles), under different error probability of each cycle.

Figure 6.8 shows the relation between the expected number of rollbacks and the error probability of each cycle, assuming $C_{seg} = 10^5$. The figure indicates that in a small error probability window, the number of rollbacks changes rapidly from almost zero to more than ten. Therefore, the number of rollbacks is highly sensitive to the error rate, and with the exponential growth in error rate, the number of rollbacks will quickly increase to unacceptable values.

The time overhead of CRR can also be evaluated with the error model. Here we

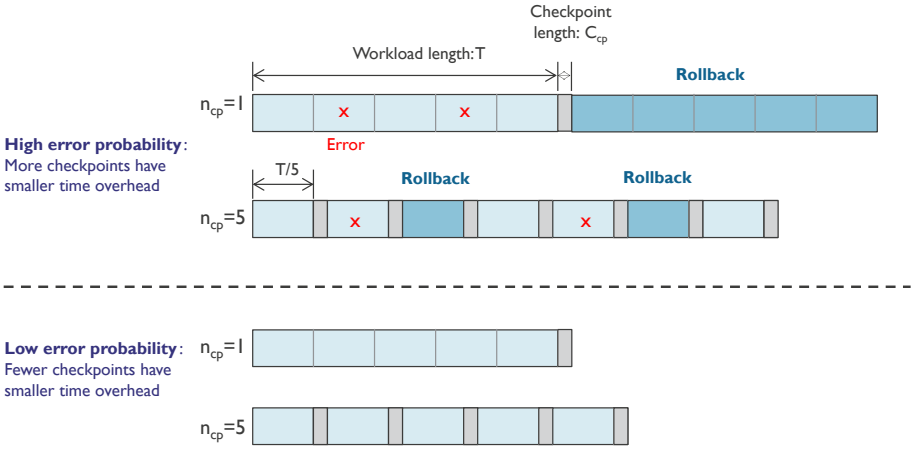


Figure 6.10: Example of how the number of checkpoints affects the time overhead under different error probability conditions

assume a time model in which a checkpoint routine is executed at the end of each segment, and a rollback routine is executed at the start of each rollback-recovery occurrence. Therefore, we can derive the time overhead of each segment by cycles as follows:

$$C_{ov} = C_{cp} + n_{rb}(C_{rb} + C_{seg} + C_{cp}) \tag{6.4}$$

where n_{rb} is the number of rollbacks, and C_{ov}, C_{cp}, C_{rb} are the number of cycles of the time overhead, checkpoint routine, and rollback routine respectively. Here, we assume a CRR system model by abstracting the OCEAN approach described in Section 6.3.2, in which a checkpoint routine contains 100 cycles and a rollback routine contains 48 cycles. Then, combined with Equation 6.3 and 6.4, the expected value of time overhead can be calculated in cycles. Figure 6.9 shows the relation between the expected value of the total execution cycles and the error probability of each cycle, assuming $C_{seg} = 10^5$. Similar to the number of rollbacks shown in Figure 6.8, time overhead quickly increases to unfeasible values with the exponential growth in error rate.

6.3.4 Optimal Checkpointing

In CRR schemes, the amount of time overhead is significantly affected by the number of checkpoints and rollbacks performed. However, there is a trade-off between the time overhead generated from checkpoints and rollbacks. As shown

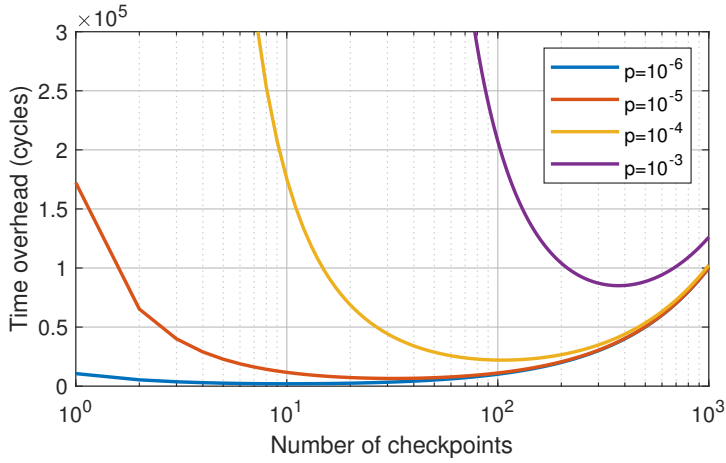


Figure 6.11: Relations between time overhead and the number of checkpoints for a segment of 10^5 cycles, under different error probability of cycles (p)

in the example of Figure 6.10, more checkpoints increase the total time spent in checkpointing routines. Nevertheless, more checkpoints also mean smaller segment sizes, which reduce the rollback and recovery time spent in each error occurrence, and thus reduce the total time spent in rollback-recovery.

Figure 6.11 shows the relations between time overhead and the number of checkpoints for a segment of 10^5 cycles. For different error probabilities of cycles (p), there is an optimal number of checkpoints which can minimize total time overhead. In general, more checkpoints are favoured at high error probability, since the time overhead is dominated by the rollback-recovery time. On the other hand, at low error probability, time overhead is dominated by checkpointing routines, so fewer checkpoints are preferred to reduce the total time spent in checkpointing routines.

If an ideal CRR scheme is assumed, in which checkpoints can be inserted anywhere in the workload and the lengths of checkpoint routines are constant, then there is an optimal number of checkpoints that could minimize the expected value of total time overhead. Based on the derivation in [174], the optimal number of checkpoints is calculated as follows:

$$n_{cp,opt} = -\frac{1}{2}\ln P + \sqrt{\frac{1}{4}(\ln P)^2 - \frac{C_{wl}\ln P}{C_{cp}}} \tag{6.5}$$

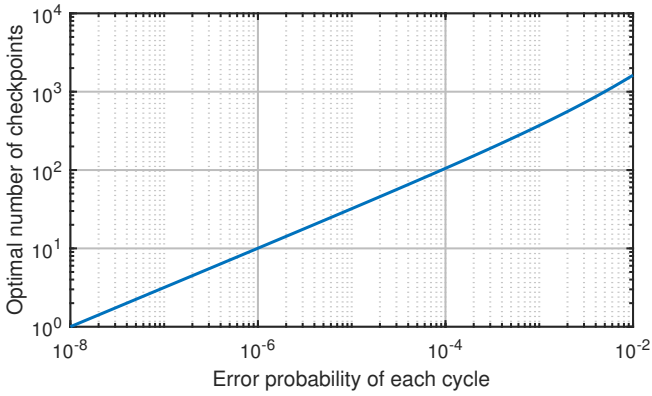


Figure 6.12: Optimal number of checkpoints under different error probabilities of each cycle

where $n_{cp,opt}$ is the optimal number of checkpoints. C_{wl}, C_{cp} are the number of cycles of the workload and the checkpoint routine respectively. P is the probability that there is no error in every cycle of the workload, which can be calculated as follow:

$$P = (1-p)^{C_{wl}} \tag{6.6}$$

where p is the probability of a cycle being erroneous. The time required for the rollback routine is not considered in the calculation. Nevertheless, the rollback routine in our CRR scheme is relatively short (48 cycles) compared to the workload length, so the difference in the calculation is negligible. Figure 6.12 shows the optimal number of checkpoints under different error probabilities of each cycle. It indicates that the optimal number of checkpoints increases with the growth of error probability.

Figure 6.13 shows the minimized time overhead with the optimal number of checkpoints, compared to the scheme of one checkpoint per segment in Figure 6.9. With variable numbers of checkpoints, the total execution cycles increase by a slower rate with respect to the error probability. Moreover, the optimal checkpointing scheme reaches a significantly higher error probability before the surge of the total execution cycles, showing much better system reliability.

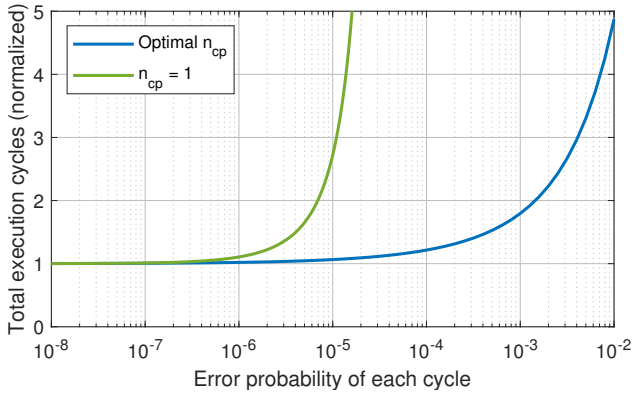


Figure 6.13: Expected value of total execution cycles with the optimal number of checkpoints compared to the scheme of one checkpoint per segment, under different error probabilities of each cycle

6.4 Processor-Layer Mitigation of Performance Variability

In the processor layer, mitigation approaches are used to tackle the performance variability to ensure timing guarantees. In Chapter 4 and 5, we have presented the workload-dependent multi-timescale mitigation approach to deal with performance variability of the architecture and software levels. Here, the use of the approach is extended to include performance variability from the lower levels, such as the time overhead of the CRR mechanism. As we will demonstrate in Section 7.3, by enlarging the time budget of each TN, the mitigation approach can absorb the time overhead of the CRR mechanism to prevent deadline misses.

In general, the time budget assigned to each TN in the adaptive control algorithms affects its capability of ensuring timing guarantees in the presence of register-level errors. Conservative algorithms supply TNs with larger budgets, which could absorb the time overhead. Therefore, conservative algorithms are better at meeting deadlines at high error probability. However, the high processor speed used by conservative algorithms consumes more energy, which is unnecessary if errors are rare. On the other hand, aggressive algorithms assign smaller budgets to each TN, which might fall short of meeting deadlines. However, it is better at energy saving. Therefore, it is important to schedule TNs with the right amount of time budgets which can ensure timing guarantees

and save energy.

Furthermore, since errors occur randomly, there is no mathematical bound of the time overhead, which makes the timing guarantees impossible to be fully ensured. However, conditions of extremely large time overhead, like multiple errors in a very short time, are not realistic in the real world. Therefore, the number of errors can be bound by realistic specifications. Also, in robust system designs, fail-safe mechanisms should be applied on top of the mitigation approaches to prevent severe consequences and recover to normal states under extreme conditions. Thus, functional correctness and timing guarantees can still be fully ensured.

6.5 System Reliability Evaluation

While effective in handling variability and errors, every mitigation approach has its own limits. If the amount of variability and errors exceeds the capability of the mitigation approach, it could fail the guarantees it promises and lead to the loss of system reliability. In the mitigation of device-level variability, an excessively large and sharp voltage droop could make the voltage droop prevention approach unable to raise the supply voltage to a sufficient voltage level, causing circuit failure. In the error correction approaches, too many errors could overwhelm the error correction mechanism, so it cannot fix all the errors in a reasonable amount of time. In performance variability mitigation, the capability of absorbing the time overhead is constrained by the maximum processor speed. Excessive execution time overhead could make the mitigation approach unable to offset the performance variability even with the highest processor speed, causing deadline misses.

Therefore, the reliability of the whole system is constrained by the limit of each mitigation approach. To evaluate the overall system reliability, it is required to analyze the boundary conditions of the mitigation approaches and interdependencies among different approaches under these conditions. With such analysis, we discovered that the system reliability of the presented framework can only be ensured within a limit of the error probability per cycle, which we refer to as the "error rate wall" (ERW). ERW can be calculated by analyzing the interdependencies of the error correction and the performance variability mitigation, which we will describe as follows.

Figure 6.14 depicts a basic example of performance variability mitigation dealing with a simple workload with one deadline and an amount of execution cycle overhead. Suppose the workload has C_{wl} cycles originally, C_{ov} cycles of overhead, and C_{tot} cycles in total including the overhead. We assume C_{max} is the maximum

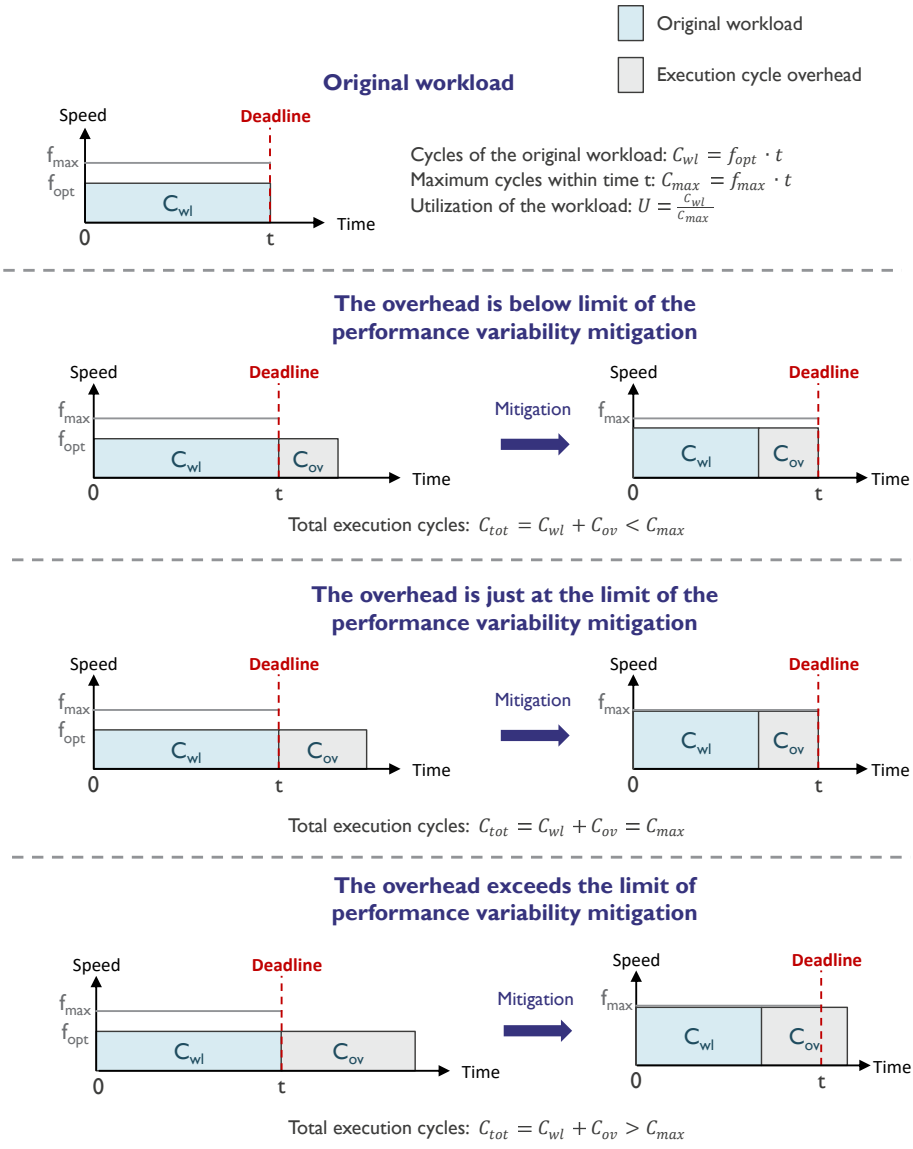


Figure 6.14: Illustration of the limit of the performance variability mitigation in mitigating the execution cycle overhead of a workload

number of cycles the processor can run with the highest speed between the start of the workload and the deadline, which represents the maximum available computing resource of this interval. Whether the workload with overhead can finish before the deadline depends on the relation of C_{tot} and C_{max}. When C_{ov} is small so that $C_{tot} < C_{max}$, the execution time overhead can be offset by raising the processor speed. If C_{ov} increases so that $C_{tot} = C_{max}$, then the processor is just fast enough to finish the workload at the deadline with maximum processor speed. If C_{ov} further increases so that $C_{tot} > C_{max}$, the total execution cycles are so large that even the maximum processor speed cannot save the workload from violating the deadline. Therefore, it requires $C_{max} \geq C_{tot}$ for the performance variability mitigation to ensure timing guarantees.

The above relation shows that the sizes of the original workload and the overhead both affect the viability of performance variability mitigation. The ratio of C_{wl} to C_{max} is known as the *utilization* (U) of the workload, representing the proportion of resource the original workload occupies out of the total computing resource. Therefore, the aforementioned criterion can be expressed by the ratio of C_{tot} and C_{wl} as follows:

$$\frac{C_{tot}}{C_{wl}} \leq \frac{1}{U} \tag{6.7}$$

Equivalently, the relation can be expressed with C_{ov} and C_{wl}:

$$\frac{C_{ov}}{C_{wl}} \leq \frac{1-U}{U} \tag{6.8}$$

The ratio of C_{ov} to C_{wl} represents the allowed amount of the overhead compared to the original workload which can still ensure timing guarantees. The allowed overhead also depends on the utilization of the original workload.

In our framework, the execution cycle overhead is generated by the CRR, in which the execution cycles are dependent on the error probability of each cycle, as shown in Figure 6.13. Therefore, combining the above equation with the relation between the total execution cycles and error probability, we can estimate the error rate limit within which the timing guarantees can be ensured, namely the ERW. Beyond the ERW, the reliability of functional correctness and timing guarantees can no longer be ensured simultaneously.

For example, assuming the CRR schemes of Figure 6.13 with the utilization of the original workload is 0.5, by Equation 6.8 we derive that the C_{tot}/C_{wl} is 2. Under these conditions, the ERW can be obtained by finding the crossing point of the curve with the y-axis level of 2, yielding approximately 7×10^{-6} for the

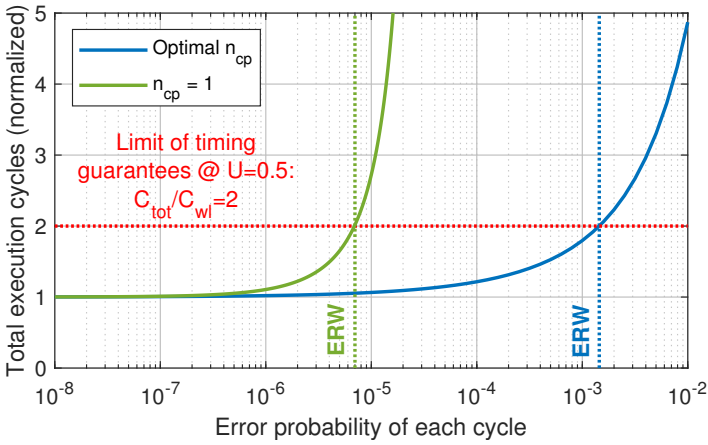


Figure 6.15: Error rate walls under 0.5 utilization of the original workload in the CRR schemes of Figure 6.13

optimal checkpointing scheme and 1.4×10^{-3} for the scheme of one checkpoint per segment. The ERW is improved by more than 200 times for the optimal checkpoint compared to the case of one checkpoint per segment, which shows the potential of enhancing reliability by optimization of checkpointing.

While the utilization can change over time, the highest utilization during the entire workload should be used in the calculation, since it is the bottleneck of timing guarantees. With this analysis, we can define the feasible region of the system specification in which reliability can be ensured. The validity of this analysis can be further proven by simulation with real workload demonstrated later in Chapter 7.

6.6 Discussion

6.6.1 Comparison with Other Cross-Layer Mitigation Approaches

In addition to our framework, cross-layer approaches have been researched by others in various layers for different reliability constraints. These approaches mainly aim to optimize for multiple goals (i.e. minimizing energy) under certain reliability constraints. Though distinct in the applied mitigation techniques,

most of these approaches are complementary to our framework in that they are deployed in higher layers, such as the OS and application layers [175].

In the error correction field, many works have explored the benefits of cross-layer approaches [172, 173], apart from the OCEAN approach applied in our framework. For example, Lee et al. [176] presents a cross-layer soft error mitigation approach for multimedia, which combines error detection codes in hardware, drop and forward recovery in middleware, and error-resilient video encoding at the application level. Moreover, Henkel et al. [175] proposed a multi-layer dependency framework of functional correctness spanning from the microarchitecture layer to the application layer. These approaches operate on similar principles but in distinct layers and application fields from our framework, so they can be combined with our framework to cover more abstraction layers and fields.

In real-time scheduling fields, plenty of work addresses the problem of scheduling in the presence of recovery events due to hardware errors [177–181]. However, these works often assume oversimplified mathematical models of hardware, error occurrences, and CRR mechanisms, which make them inadequate to reflect real-world scenarios. For example, the assumptions of the error model can be a single error [178], a bounded number of errors [177], or Poisson distribution [179], and they can assume the recovery processes to be free of errors [178]. These limitations result in discrepancies between the model and the real statistics of error occurrences. Hence, our framework can complement these insufficiencies by providing realistic hardware and error models. Regardless of the aforementioned flaws, these approaches can serve as multi-task scheduling solutions which work in a higher layer on top of our framework.

In addition to functional correctness and timing guarantees, many approaches also address other reliability objectives (i.e. thermal constraints and lifetime), as discussed in Section 2.4.2. These techniques are also included in cross-layer approaches of multiple reliability objectives. For example, Ansari et al. [182] proposed an adaptive CRR approach under thermal design power (TDP) constraints. In addition, previous works have proposed real-time scheduling approaches while considering thermal constraints [34] or lifetime [183] at the same time. These approaches can be complementary to our work in addressing more reliability requirements, completing the framework with the full reliability scope.

6.6.2 Applicability to other architectures and technologies

Although the presented framework is instantiated for a single-core processor, the principles of this framework can be applied to other architectures and

technologies. The first layer, the mitigation of device-level variability, can be any domain-specific mitigation approaches which target the error-generating mechanisms prominent for that technology and domain. For example, in a space application, the Single Event Upset (SEU) caused by radiation could be the dominant source of errors, which is mainly mitigated by the radiation-hardened circuits to reduce the error probability of the system. The second layer, the register-layer error correction, could also be realized by alternative error correction techniques which are cost-effective for that architecture and application. For example, the error correction codes (ECC) in the communication domain can be also used for checking the register errors. Finally, for the last layer, the applicability of the performance variability to the other architectures and technologies has been elaborated in Section 4.6. In summary, the suitable mitigation approaches of each layer can vary with technologies, architectures, applications, and domains. Therefore, the system designers should find the most cost-effective combination of mitigation approaches for the targeted system to implement the cross-layer mitigation framework on the same principles.

6.7 Summary

We propose a cross-layer mitigation framework to tackle the device-level and circuit-level performance variability. The framework can ensure both functional correctness and timing guarantees by three mitigation approaches in a layered structure. In the circuit layer, mitigation approaches decrease device-level variability (process, voltage, temperature, aging) to reduce register-level errors. Then, in the register layer, error correction mechanisms such as CRR fix register-level errors at the cost of time overhead. Last, in the processor layer, mitigation approaches ensure timing guarantees in the presence of performance variability generated from the lower layers. Our framework is extendable with other mitigation approaches in different layers, achieving a full reliability framework across all layers and objectives.

Chapter 7

Cross-Layer Reliability and Variability Mitigation Framework: Experiments

To evaluate the effectiveness, constraints, and costs of the cross-layer reliability and variability mitigation framework, a holistic simulation flow needs to be developed, in which the simulations of each layer are incorporated. Here, we developed a workload-dependent simulation framework for each level of mitigation. Combining individual simulations, we can perform cross-layer simulations to reveal the interdependency among layers, as well as the figures of merits in the reliability of the holistic system. The simulation framework can be further used for co-simulation and co-optimization of system parameters across all layers for maximizing reliability while minimizing cost.

This chapter is structured as follows: the overall simulation framework of the cross-layer reliability and variability mitigation framework is introduced in Section 7.1. The following sections show the experiments on the system reliability with different layers of the framework. In Section 7.2, we show a workload-dependent error simulation flow to evaluate the effect of the circuit-layer mitigation. Then, Section 7.3 demonstrates a cross-layer co-simulation combining the register layer and the processor layer to reveal their interdependencies. Finally, the effect of optimal checkpointing on the system is analyzed in Section 7.4, showing how changing system parameters can improve the reliability of the system. Part of the work presented in this chapter has been published in the co-authored work of [147].

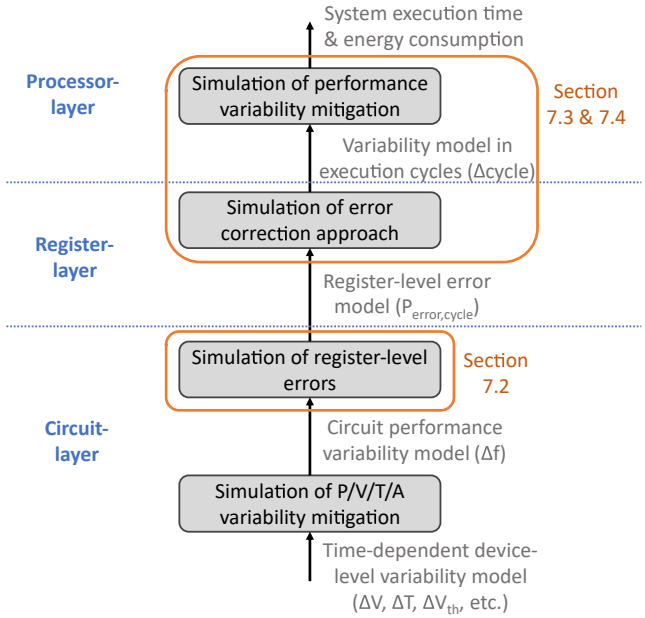


Figure 7.1: Simulation framework of the cross-layer reliability and variability mitigation

7.1 Cross-layer Simulation Framework

To simulate the effect of the entire cross-layer reliability and variability mitigation framework, multiple simulation methods for different purposes in different layers need to be integrated. Figure 7.1 shows the overall simulation framework required for the experiments on the whole framework. In our work, the essential parts of the simulation framework are implemented, while the rest of the framework and the integration of the holistic framework are yet to be developed. Nevertheless, the principles of establishing the cross-layer simulation framework are demonstrated, and important properties of cross-layer mitigation approaches are revealed through the simulation, which we will demonstrate in the following section.

At the beginning of the simulation framework, device-level variability models are required to generate the time-zero and time-dependent variability in process, voltage, temperature, and aging. The variability models vary with the addressed variability sources, expressed in their own physical metrics (ΔV, ΔT, ΔV_{th}, etc.). They can be either statistical models such as the V_{th} distribution of the

process variation [3], or time-dependent traces of variability metrics such as BTI-induced V_{th} fluctuation over time [184].

Then, the simulation of the device-level variability mitigation is required to simulate the resulting circuit performance variability under the applied device-level variability. The simulation approach depends on the types of device-level variability and the applied mitigation approaches. In general, most of the device-level variability and mitigation approaches result in the time-dependent clock frequency change, so the circuit performance variability can be modelled as frequency shift (Δf) over time. For time-zero variability, variation-aware static timing analysis has been developed to accurately simulate the performance variability of logic gates and digital circuits [185, 186]. However, there is a lacking of mature approaches to accurately simulate the influence on the circuits from the time-dependent variability such as aging effect. Therefore, research has been conducted in order to fill this gap [184, 187], and we look forward to the development of an accurate time-dependent variability simulation flow.

After the performance variability of the circuits is known, the simulation of register-level errors is used to obtain the occurrence rate of the errors in the registers, which is modelled by the error probability per cycle ($P_{error,cycle}$). To serve this purpose, in Section 7.2 we will demonstrate a workload-dependent simulation flow which can simulate the register errors in a circuit when running a certain workload. In the simulation, a constant clock frequency shift (Δf) is injected into the circuit, causing register errors which can be read out as the results.

Once we know the amount of register-level errors in the circuit, the simulation of error correction approaches is used to simulate the error correction events in the system, which we will demonstrate in Section 7.3. In the simulation, the register-level errors are generated based on a constant error probability ($P_{error,cycle}$) for each cycle, and injected into the workload to be corrected by the error correction approach. As a result, the execution time overhead caused by the error correction is obtained and modelled as the variability in the execution cycles ($\Delta cycle$). In theory, the execution cycle overheads of correcting each segment of the workload follow the Geometric distribution which we have derived in Section 6.3.3.

Afterwards, the variability in execution cycles ($\Delta cycle$) is injected into the workload used as the input of the simulation of performance variability mitigation. The simulation flow has been described in Section 5.1. Finally, the results of this simulation show the execution time and the energy consumption of the system running the workload if all the addressed variability and errors are properly handled.

To fully simulate the interdependencies across layers, the individual simulation flows described above need to be integrated, in which the output variability and models of the former layer should be applied as the input of the latter layer. In the simulation framework, we have accomplished the integration of the simulations of the error correction approach and the performance variability mitigation, in which the exact execution cycle overhead of each workload segment from error correction is applied directly to the inputs of the performance variability mitigation. The integration of both simulation flows allows us to experiment with the interdependencies of the two mitigation approaches, which we will elaborate on in Section 7.3 & 7.4.

7.2 Simulation of Register-Level Errors

To analyze the reliability of systems, it is crucial to evaluate the number of register-level errors in the system under different variability and operating conditions, either in design time or run time. Therefore, we developed a workload-dependent error simulation flow to acquire the error occurrences during the running of a certain workload. The overall simulation flow is shown in Figure 7.2.

At the beginning, variability-aware timing analysis should be performed to how much the gate delays and the clock frequency of the circuit are shifted by the time-zero/time-dependent device-level variability. As mentioned in Section 7.1, the time-zero/time-dependent variability models depend on the addressed variability sources, and are yet to be integrated into our simulation flow. Instead, in this simulation a constant clock frequency shift is assumed to be the impact of the variability on the circuit. While the source of the frequency shift is not specified, it could originate from device-level variability with a timescale longer than the workload, such as process variation, voltage shift, temperature change, or long-term aging effect. It could also occur as a result of the modulation by the mitigation approaches of the device-level variability, such as adaptive clocking for voltage droop prevention.

Meanwhile, RTL-level simulation is performed on the RTL design of the target circuit running the target workload, where the golden input and output vectors of all registers in the design are extracted. Then, gate-level simulation is performed, based on the gate delays and the clock frequency after the effect of the variability. In the simulation, the gate delay propagation is captured in ps-level precision, so timing violations and register-level errors are generated and caught in the output vectors. Finally, by comparing the gate-level output

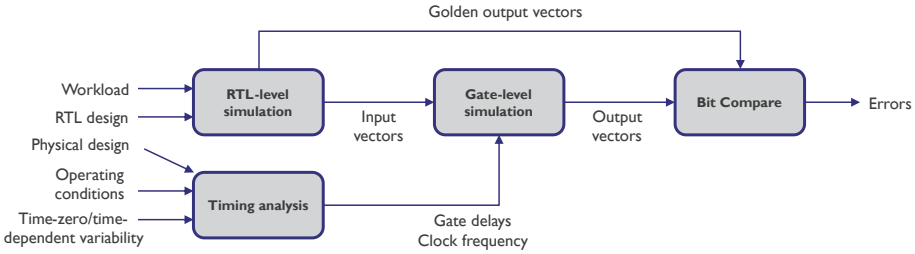


Figure 7.2: Workload-dependent error simulation flow

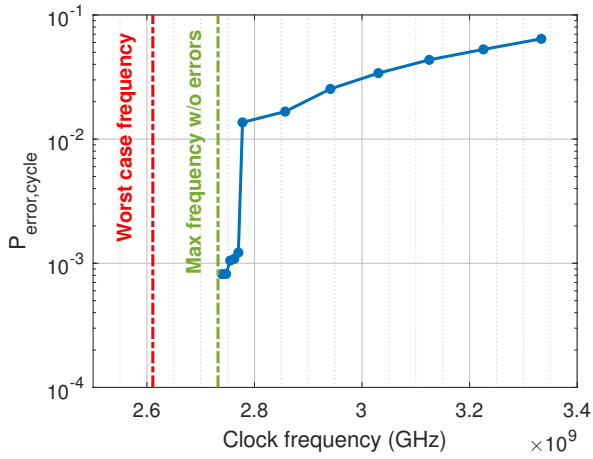


Figure 7.3: Error rate per cycle under different clock frequencies

vectors and golden output vectors, the number and the locations of register-level errors are acquired.

The error simulation flow enables error rate analysis under different variability and operating conditions. As an example, we analyzed the impact of the design margin on the error rate of the ALU pipeline stage of the Ariane core. In this experiment, we swept across different clock frequencies to represent the scenario in which the clock frequency is changed. No other variations in voltage, temperature, or aging are applied. The RTL-level model used in the simulation is the same as in 5.1, and the gate-level model and physical implementation are implemented in commercial 28nm technology. The applied workload is the same lower-band quantization of ADPCM encoding workload as in 5.2, in which the length is about 3.6 million cycles.

In our analysis, we define the cycle-based error rate ($P_{error,cycle}$) as the number of erroneous cycles to the total number of cycles. A cycle is erroneous if any bit in the output vectors has the wrong value in this cycle. Figure 7.3 shows the simulation results. The worst-case frequency calculated by static timing analysis is 2.61GHz, defined by the critical path delay. However, if the critical path is not activated during the entire workload, the circuit can run at a higher clock frequency without any errors. In our case, the clock frequency can be increased without observing any errors until 2.73GHz, which is 4.6% higher than the worst-case frequency. Then, if the clock frequency is further increased, the error rate is increased rapidly. At 2.78GHz, a cycle-based error rate is already larger than 1%. In summary, the error rate is shown to be highly sensitive to clock frequency, which modulates the design margin of the circuit. The high sensitivity of error rate to operating conditions also shows the importance of accurate error rate evaluation, since any tiny discrepancy in operating conditions could result in significant degradation of reliability.

7.3 Cross-layer Interdependency Analysis

To evaluate the overall effect when multiple mitigation techniques work together, it is essential to understand the interdependencies across layers and different mitigation approaches. As an example, we study and simulate the effect of combining CRR in the register layer and performance variability mitigation in the processor layer. Specifically, Figure 7.4 shows how the two mitigation approaches can collaborate to meet the dual reliability requirements of functional correctness and timing guarantees. When CRR is applied to correct the errors in the workload, the execution time overhead can result in deadline misses. Therefore, performance variability mitigation is applied to increase processor speed to offset the execution time overhead, so that timing guarantees can be ensured.

The interdependency of the two approaches resides in the execution time overhead of CRR. When the error rate is high, the execution time overhead will be large, so the processor speed needs to be increased more dramatically to compensate for the overhead, and thus the energy consumption will be large. Furthermore, if the error rate is too high, the workload can be flooded with rollback events, so that even the highest processor speed cannot offset the time overhead. In this scenario, timing guarantees cannot be ensured anymore and deadline misses can occur. In summary, Figure 7.5 shows the outcomes of the collaboration of the two approaches, in which the energy is increased with the error rate, and the deadline hit rate is decreased with the error rate. In the following subsections, we will show how the interdependences can be simulated.

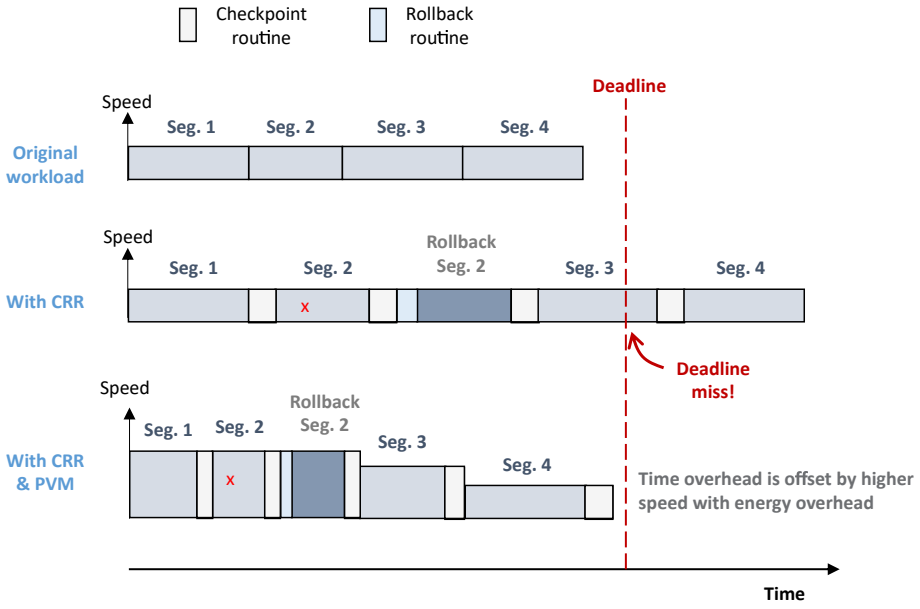


Figure 7.4: Illustration of an example of how CRR and performance variability mitigation can collaborate to meet the dual reliability requirements

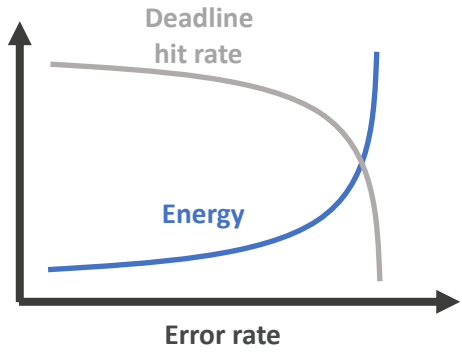


Figure 7.5: Interdependency of CRR and performance variability mitigation

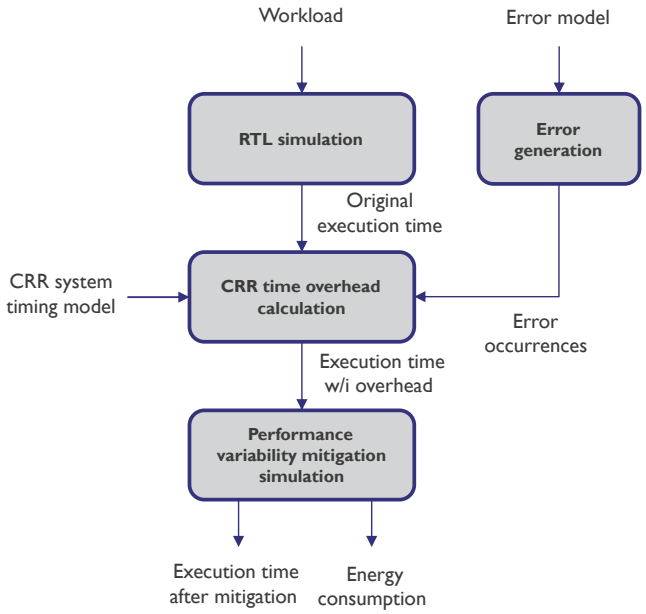


Figure 7.6: Cross-layer simulation flow incorporating the register-layer CRR and the processor-layer performance variability mitigation

7.3.1 Simulation Flow and Experimental Setup

To analyze the effectiveness and cost of the holistic system in meeting the dual reliability requirements simultaneously, it is necessary to establish a cross-layer simulation framework which integrates the simulation of every individual technique. Therefore, we developed a simulation framework which combines the simulation of the circuit-layer CRR approach and the processor-layer performance variability mitigation approach.

Figure 7.6 shows the overall flow of the simulation framework. First, the target workload is profiled with RTL-level simulation to the original execution time without errors. Then, errors are generated inside the workload based on the error model. Here, we assume a static error rate model in which the error probability is the same for each cycle over the entire workload, while the error simulation flow in Section 7.2 can also be integrated for more sophisticated simulation.

Afterwards, to simulate the effect of the CRR approach, a system timing model is created by abstracting the OCEAN approach described in Section 6.3.2 [21].

In this model, the checkpoint routine and the rollback routine are assumed to be 100 cycles and 48 cycles respectively, regardless of the computation phases. And the recomputation is assumed to take the same execution time as the original computation phase. The number of rollbacks is randomly generated for each computation phase with the probability distribution of Equation 6.2, based on the error probability specified by the error rate model. By summing up the execution time of the checkpoint routines, the rollback routines, and the recomputation, the time overhead of each computation phase of the workload can be calculated.

Next, the execution time after CRR is used as the input of the simulation of performance variability mitigation. For this purpose, we use the workload-dependent multi-timescale mitigation approach in Chapter 4 and the cycle-accurate simulation flow in Chapter 5. The simulation flow generates the execution time and energy consumption after the mitigation as the final outcomes of the whole framework. In the calculation of the energy consumption, the energy of the checkpoint and rollback routines are considered to consume the same energy per cycle as the TN they belong to.

In the experiment, we use the same Ariane core processor and ADPCM lower-band quantization workload and Ariane core processor as the experiment in Chapter 5. The segmentation of workload is the same for both mitigation approaches. In other words, a TN in performance variability mitigation is the same as a computation phase in CRR. We performed simulations across different levels of error probability. For each level, we performed Monte Carlo simulations of 100 runs and calculated the average results.

In the scheduling algorithm, timing budgets can be deliberately enlarged to absorb the CRR time overhead, so that deadline misses can be prevented. To compare different algorithms and policies of time budget assignment, four algorithms are selected in our simulation, ranging from aggressive ones to conservative ones.

- **DS**: dynamic-scenario based ***(most aggressive)***
- **DS 1.5x**: dynamic-scenario based, budgets scaled by 1.5x
- **DS 2x**: dynamic-scenario based, budgets scaled by 2x
- **WCET**: worst-case execution time ***(most conservative)***

The DS-based approach is the most aggressive algorithm, which can derive a tight budget with full guarantees under the error-free condition at run time. On the other hand, worst-case execution time is the most conservative algorithm.

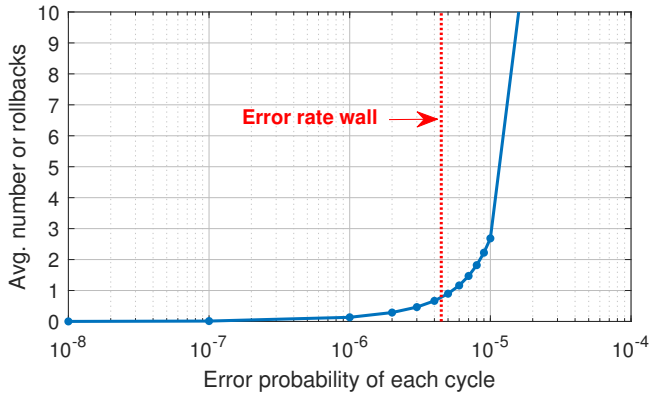


Figure 7.7: Average rollbacks of every computation phase under different error probability conditions

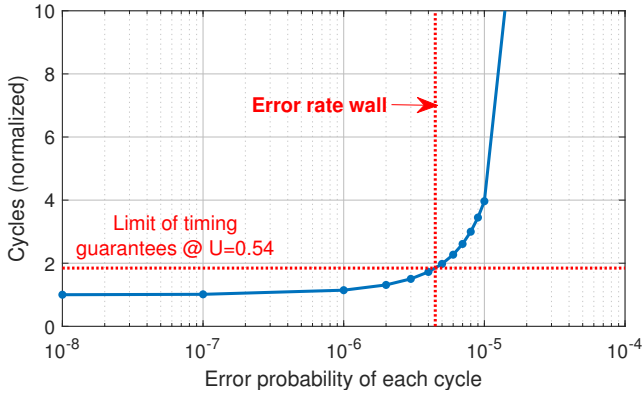


Figure 7.8: Total executed cycles normalized to the cycles of the original workload, under different error probability conditions

Moreover, two additional variants of the DS-based algorithms are created, in which the budgets for all TNs are enlarged by 1.5 times and 2 times respectively. These algorithms aim to reflect the influence of the aggressiveness of algorithms on reliability and costs, which has been discussed in Section 6.4.

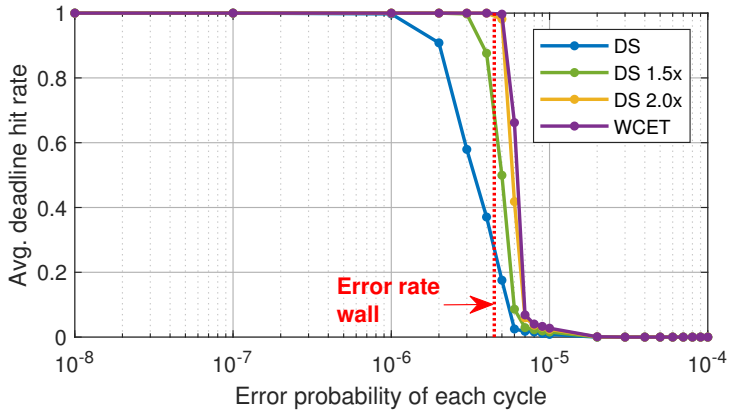


Figure 7.9: Deadline hit rate under different error probability conditions

7.3.2 Results and Discussion

Figure 7.7 shows the number of rollbacks increases with the error probability in our demonstrated system. The rollbacks result in increased numbers of executed cycles, shown in Figure 7.8. The rollbacks and the execution cycles are only influenced by the CRR systems, regardless of the algorithm used in performance variability mitigation. The extra cycles apart from the original workload are approximately linear to the number of rollbacks. The trend of the rollbacks and the total cycles is consistent with the execution time overhead of Figure 6.8 & 6.9, which proves the correctness of the theoretical calculation in Section 6.3.3.

The error rate wall of this system can be derived with the execution cycles and the highest utilization over the original workload, as described in Section 6.5. In our ADPCM lower-band quantization workload, the highest utilization is calculated based on the frame with the highest processing time over the entire workload, which results in a utilization of 0.54. With Equation 6.7 and Figure 7.8, the error rate wall is estimated to be at the error rate of 4.5×10^{-6} each cycle.

Figure 7.9 shows the deadline hit rate under different error probability conditions in our demonstrated system. It shows that the error rate wall is approximately located at the boundary of whether the 100% hit rate can be ensured with conservative algorithms (DS 2.0x, WCET). Before the error rate wall, the execution time overhead could be offset by higher processor speed, so deadline misses could still be completely prevented. However, beyond the error rate wall, the rollbacks are so frequent that even the highest processor speed cannot save

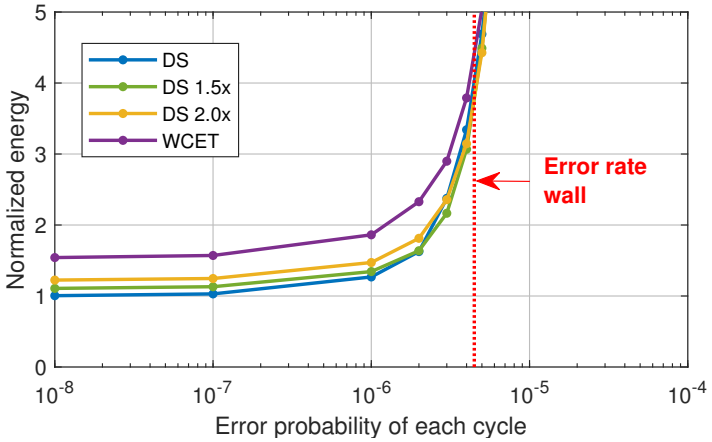


Figure 7.10: Total energy consumption normalized to the energy consumption of the original workload, under different error probability conditions

the deadlines from being missed. Therefore the hit rate rapidly drops to almost zero in a small window of error probability, which shows the high sensitivity of timing guarantees to error probability. If the error probability is even higher, the number of rollbacks could increase to more than 10 rollbacks per computation phase, which means that the workload would be flooded with rollbacks. Dealing with such a huge number of rollbacks is impractical for a reasonable system. These results show that the error rate wall defines the inherent system limit of the conditions that both reliability requirements can be fulfilled, which cannot be improved by the algorithms.

Comparison among scheduling algorithms shows that conservative algorithms bring higher deadline hit rates. Aggressive algorithms (DS, DS 1.5x) can have deadline misses even before the error rate wall is reached, due to the insufficient time slack left for absorbing the execution time overhead. However, beyond the error rate wall, all deadline hit rates converge to zero regardless of the algorithms. These results also prove that the error rate wall is the inherent system limit independent of the algorithms.

Figure 7.10 shows the energy consumption under different error probability conditions. Near the error rate wall, the energy consumption increases rapidly so that it is about 4 times the original energy consumption at the error rate wall. The sources of the energy overhead are twofold. First, running the checkpoints, rollbacks, and recomputations all result in extra energy. Second, the higher processor speed for offsetting time overhead leads to higher power dissipation

and thus consumes more energy. As for the algorithms, conservative algorithms have higher energy overheads when the error probability is low. However, when the error probability is high, aggressive algorithms may have deadline misses, which make the processor speed up to recover from the negative slack. Therefore, the advantage of aggressive algorithms in using slower processor speed diminishes, so the energy consumption of different algorithms becomes similar.

The position of the error rate wall is strongly dependent on system parameters, such as the processor speed, the granularity of checkpointing, etc. To know whether a system can ensure its reliability requirement, one can check whether the position of the error rate wall is further than the error probability, which can be simulated by the flow we showed in Section 7.2. Since the system reliability depends on both the error rate wall and the error probability, the ways of improving system reliability are involved with affecting these two factors. One way is to reduce the error probability through technology improvement (e.g. advanced lithography) or circuit design (e.g. ECC or DMR). However, as discussed in Section 6.3.1, these error-tolerant circuit design techniques can lead to significant area, performance, and energy overheads. The other way is to improve the tolerance of errors (i.e. moving the error rate wall further) by optimizing the system. For example, minimizing execution time overhead by optimizing the number of checkpoints can improve reliability, as we will show in the next section. Co-simulation and co-optimization of all system parameters across all layers and techniques are required in order to maximize the reliability and minimize the cost of the system.

7.4 Analysis with Optimal Checkpointing

In Section 6.3.4, we have shown that to minimize execution time overhead, the optimal numbers of checkpoints are different for different error probability conditions. With the minimum execution time overhead, both the deadline misses and the energy costs are reduced in performance variability mitigation, so reliability can be improved. Here, we will analyze how the error rate wall can be moved by selecting different numbers of checkpoints, which is an example of how system parameters affect reliability.

7.4.1 Experimental Setup

We used the same simulation flow and experiment setup as Section 7.3. However, in this experiment, a TN is segmented into multiple computation phases. In

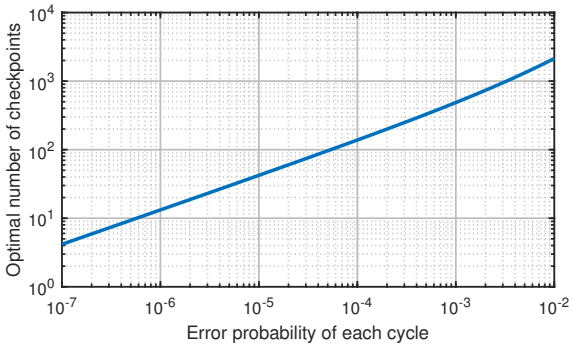


Figure 7.11: Optimal number of checkpoints per TN under different error probability conditions

the calculation of execution time and energy consumption, computation phases in the same TN are assumed to be homogeneous. For example, if a TN is segmented into 10 computation phases, each phase takes 10% cycles and 10% energy consumption of the original TN under the same processor speed. As in the original CRR mechanism, a 100-cycle checkpoint routine is added at the end of each computation phase. If any error occurs, a 48-cycle rollback routine is added, and recomputation is performed. By Equation 6.5, we calculated the optimal number of checkpoints per TN based on the average length of all TNs. Besides the case of the optimal number of checkpoints, we also simulated the case of 1, 10, 100, 1000, and 2000 checkpoints per TN for comparison. The DS 2x algorithm of the last section is used in this experiment, which is a relatively conservative algorithm that can approach the system limit in mitigating the execution time overhead.

7.4.2 Results and Discussion

In the simulation results, Figure 7.12 shows that the numbers of total executed cycles are highly dependent on the number of checkpoints. When the number of checkpoints increases from 1 to 100, the overhead of cycles is remarkably reduced at the high error probability over 10^{-6} per cycle, since the recomputed workload segment required for each error occurrence is largely reduced by 100 times. As a result, the error tolerance is approximately improved by two orders.

However, when the number of checkpoints further increases to 1000 and 2000, the cycles of checkpoints become significant and add up to 80% and 160% cycle overhead respectively to the original workload, significantly increasing the total

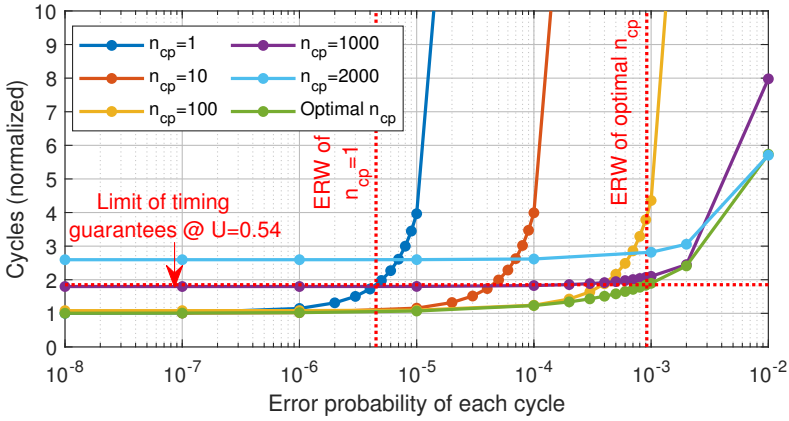


Figure 7.12: Total executed cycles with different numbers of checkpoints per TN (n_{cp}), under different error probability conditions. The numbers of executed cycles are normalized to the cycles of the original workload. The red dash line indicates the error rate wall under the optimal n_{cp} condition.

execution cycles even at low error probability. Such a high overhead would greatly degrade the performance and energy efficiency of a system. Even though these schemes have smaller overheads at the high error probability over 10^{-3} per cycle, the overhead is still too high for a feasible system. Furthermore, in the 2000-checkpoint scheme, the high overhead makes the number of cycles exceed the limit of ensuring timing guarantees in all error probability ranges. Thus, the 2000-checkpoint scheme is not feasible for system reliability. Also, any schemes of more than 2000 checkpoints only increase the overhead even more, so they cannot meet the system reliability requirements.

Nevertheless, if the optimal number of checkpoints is applied for each error probability condition, the cycle overhead can be minimized. Also, the trend of the number of cycles does not rise as quickly with the error probability as in the cases of the fixed number of checkpoints. The smaller sensitivity to the error probability means that the system is more robust to miscalculation and variation of the error rate, which also benefits the overall reliability of the system.

The minimized cycle overhead leads to better deadline hit rates as shown in Figure 7.13. With the increase in the number of checkpoints from 1 to 1000, the error rate wall is pushed forward to 9×10^{-4} error probability per cycle, about 200 times to the original ERW of $n_{cp} = 1$. The results are consistent

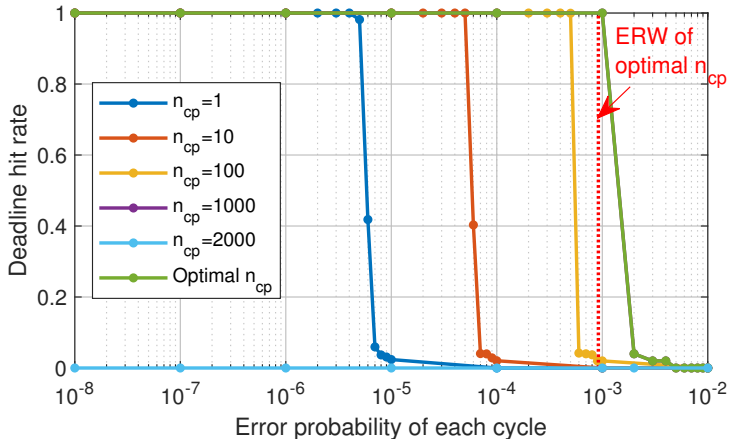


Figure 7.13: Deadline hit rate with different numbers of checkpoints per TN (n_{cp}), under different error probability conditions. The curves of $n_{cp} = 1000$ and optimal n_{cp} overlaps with each other, yielding similar deadline hit rates.

with the observation of Figure 7.12 that the 1000-checkpoint scheme has the smallest overhead among all fixed-checkpoint schemes at around the 10^{-3} error probability condition. However, when the number of checkpoints further increases to 2000, the deadline hit rate drops to 0 in all error probability conditions, since the overhead from the checkpointing routines is too large to be mitigated by the highest processor speed even without any errors. The failure to meet deadlines in the 2000-checkpoint scheme is in agreement with the observation in 7.12 that the execution cycles exceed the limit. Furthermore, we suppose that any schemes above 2000 checkpoints only increase the overhead, so the deadline hit rate will still stay at 0.

With the optimal number of checkpoints, the deadline hit rates can always be maximized in any error probability condition, and the error probability the system can operate without deadline misses is the highest. While the case of 1000 checkpoints also yields similar deadline hit rates, the overhead of checkpointing routines increases the execution cycles too much under the low error probability condition, consuming significant energy which is discussed below.

The energy consumptions in Figure 7.14 show similar effects. Increasing the number of checkpoints to 100 reduces energy overhead at high error probability conditions, so the system can operate at 100× higher error probability without energy rising sharply. However, with 1000 checkpoints per TN, the large

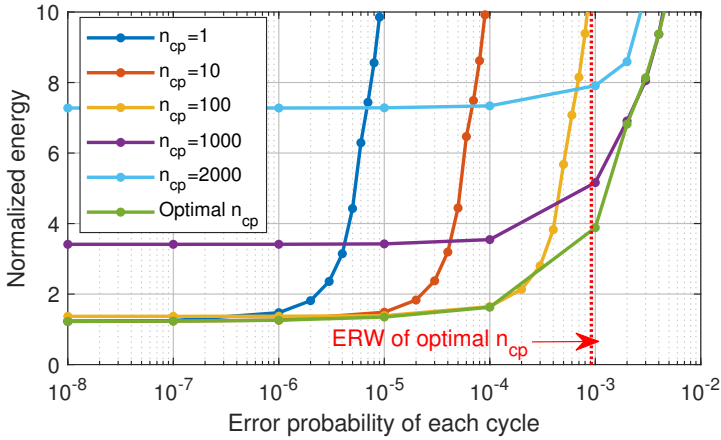


Figure 7.14: Total energy consumption with different numbers of checkpoints per TN (n_{cp}), under different error probability conditions. The energy consumptions are normalized to the one of the original workload

overhead of checkpoints increases energy consumption to more than three times the original energy consumption, which greatly degrades the energy efficiency of the systems. For the 2000-checkpoint scheme, the energy consumption is even more than seven times, and further increasing the checkpoints only enlarges the energy consumption. With optimal checkpointing, energy consumption can always be minimized at any error probability condition. Also, the sensitivity of energy consumption to the error probability is smaller.

In conclusion, we demonstrated how optimal checkpointing can remarkably improve reliability and reduce system costs in execution time and energy consumption. Therefore, checkpointing needs to be adaptive to the error rate in run time. To realize this, it is crucial to develop well-supported software and firmware structures for adaptive checkpointing, and accurate run-time error rate estimation in hardware.

7.5 Summary

We developed a workload-dependent simulation framework which combines individual simulations of different layers to reveal the holistic reliability and costs of the system. First, we demonstrated how a workload-dependent error simulation flow can evaluate the error rate of the registers under different

variability and operation conditions. Then, we showed a cross-layer simulation flow that incorporates register-level CRR and processor-level performance variability mitigation. The flow enables analysis of system reliability in meeting both functional correctness and timing guarantees, as well as the underlying energy overheads in meeting these requirements. The analysis proves the concept of the "error rate wall," the error rate limit in which the system can ensure the dual reliability requirements. Finally, we showed the error rate wall can be moved forward can be improved by using the optimal number of checkpoints, which is an example of how optimizing system parameters can improve the reliability of the entire system.

Our work can be further extended to a holistic framework incorporating all mitigation approaches from the device to application layers to ensure multiple reliability requirements. This framework could enable co-simulation and co-optimization of system parameters across all layers to maximize reliability and minimize costs. In this way, we can improve the designs of highly reliable systems by holistically considering all reliability requirements.

Chapter 8

Performance Variability Evaluation of the Carbon Nanotube Technology

There is wide and general applicability of the theoretical framework proposed in earlier chapters in dealing with any kind of variability-related problems. For example, the framework can be extended to analyze and mitigate the impact of technology-specific device-level variability on the speed and energy of the processors. To substantiate this statement, we have adopted the framework to tackle the variability of the emerging technology of Carbon Nanotube Field-Effect Transistors (CNFETs) on processor-level integrated circuit designs. We will show that the potentially large device-level variability of CNFETs [188,189] leads to significant speed and energy variations at the full processor level. Moreover, the non-negligible performance variability of CNFET processors can be effectively mitigated by our novel framework with reasonable system-level energy overheads, as shown in this chapter.

This chapter is structured as follows: first, we will give an introduction to the CNFET in Section 8.1. Section 8.2 will discuss the main variability sources of CNFETs. Then, the evaluation of variability impact on processor-level designs of both CNFET and silicon CMOS technologies is elaborated in 8.3. Thereafter, the mitigation cost estimation of CNFET variability in processor-level designs is discussed in 8.4. Finally, we will discuss the conclusions of this research and the prospect of further CNFET variability research in 8.5.

This research on performance variability evaluation of CNT technology is a

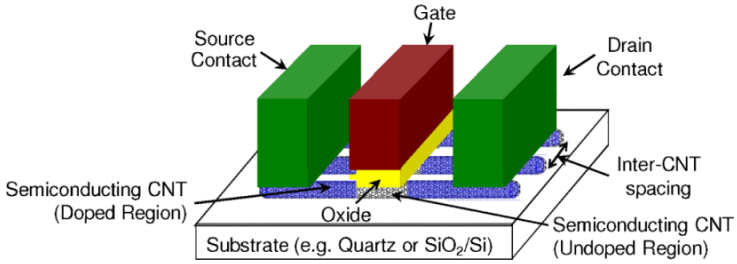


Figure 8.1: 3D view of the planer CNFET structure. Replotted from [22]

joint effort with Stanford University. In particular, the variability distribution generation, compact model creation and library characterization of the CNFET technology were accomplished by Carlo Gilardi of Stanford University. Here, we would like to acknowledge the contribution of our colleagues at Standard University to this work.

8.1 Introduction to Carbon Nanotube Field-Effect Transistors (CNFET)

The diminishing return of downscaling in advanced CMOS technologies has resulted in disruptive technology options that could provide further benefits of performance and energy efficiency after the end of CMOS scaling. Among numerous technology options, the usage of carbon nanotubes (CNT) is considered promising. CNTs are nano-materials in which the hexagonal lattice of carbon atoms is rolled into a tube shape with about one-nanometer diameter. They possess plenty of exceptional mechanical, thermal, and electrical properties, including extremely high tensile strengths, high moduli, large aspect ratios, low densities, good chemical and environmental stabilities, and high thermal and electrical conductivities. Since the discovery of CNTs, they have found their way into many applications, especially for applications in catalysis, energy, and environmental areas. In particular, CNTs have been used as fillers in advanced battery electronic additives, supercapacitors and battery electrodes, and lightweight high-strength composites [190].

In the electronic field, CNTs have been proposed as a channel material for transistors, which is known as Carbon Nanotube Field-Effect Transistors (CNFET). Figure 8.1 [22] shows a typical CNFET structure, which resembles a planer silicon MOSFET except the channel is composed of an aligned array of

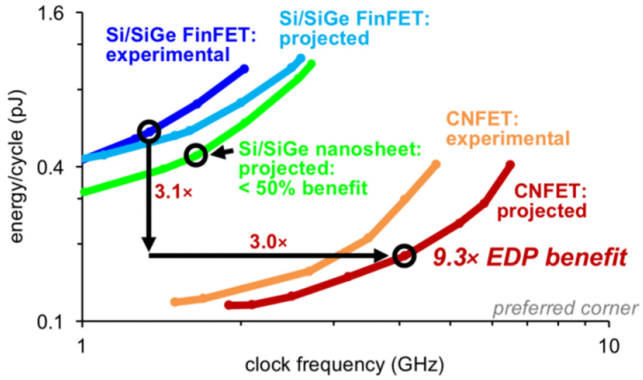


Figure 8.2: Energy vs. clock frequency results across different CNFET and silicon technology options of 5 nm node on 32-bit commercial processor core. Replotted from [23]

multiple CNTs with the spacing of several nanometers. The source and drain regions of CNTs are heavily doped, and the gated regions can be undoped [189].

Since the first report of CNFETs in 1998 [191], their superior geometrical and electrical properties have made them a promising candidate for next-generation transistor devices, replacing silicon CMOS after it eventually reaches its physical limits. It is reported that the hole carrier mobility of CNFETs is more than eight times higher than that of silicon and maintains superior carrier transport even at very thin (1–2nm) body thickness. These advantages are translated into a sharp sub-threshold slope (SS) of 70 mV/decade and a 25% higher effective drive current (I_{eff}) than that of silicon at the same level of the off-current (I_{off}). When compared to other advanced technology options of silicon, CNFET offers six times the energy-delay product (EDP) benefit in the experiment, while the projected EDP benefit can even reach nine times, as shown in Figure 8.2 [23]. Despite tremendous challenges yet to be overcome in manufacture, recent work has shown a successful design and fabrication of a 16-bit microprocessor consisting of 14,000 CNFETs [192], which has paved the way toward large-scale electronic systems.

8.2 Variability of CNFET Devices

Despite the advantages of CNFETs mentioned in the last section, large device variability is proved to be a major threat on the path to mass production of CNFETs. It is reported among the basic devices, each with 1–2 nanotubes as

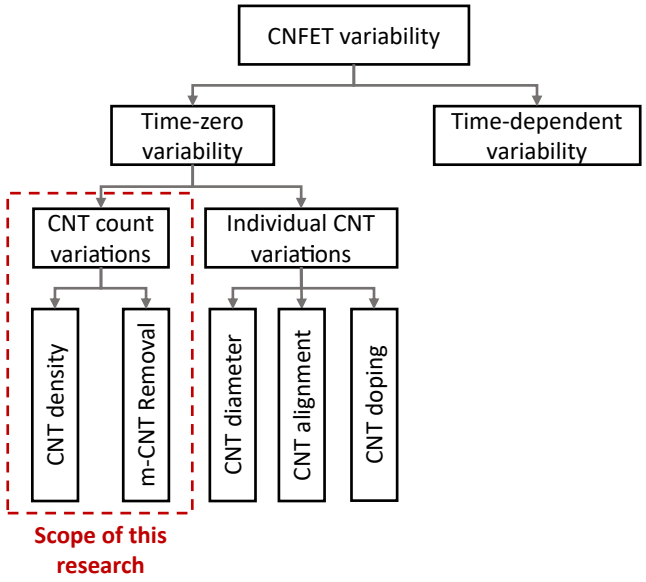


Figure 8.3: Taxonomy of variability in CNFETs. Only the main sources are enumerated and the miscellaneous sources are left out for clarity.

the channel, there is large variability in multiple device metrics, including two orders of magnitude in the on-current (I_{on}), varied V_{th} of a range above 1 V, and the device SS ranging from 60 to 800 mV/decade [188]. The huge variability necessitates having multiple CNTs in one channel to mitigate variability by averaging effects, while in turn it introduces the CNT count variations as the new dominant source of variability [22]. Thus, improving uniformity is critical for CNFETs to be qualified for integrated circuit applications. This problem has urged a great deal of research conducted over the years, which strives to eliminate the imperfections and variations in the CNT fabrication process to meet the high standard of yield and variability for large-scale integration.

In general, the variability sources of CNFETs can be classified as Figure 8.3. The main sources of time-zero variability have been identified and characterized in past research, while there are only limited studies on time-dependent variability [193]. The sources of time-zero variability can be further categorized into two groups: CNT count variations and individual CNT variations. The sources of CNT count variations are the sources that lead to unequal numbers of CNTs from one CNFET device to another, such as CNT density variations and the removal of metallic CNTs (m-CNT), which we will introduce as follows:

- **CNT density variations**: Due to the imprecise control over the positions of individual CNTs during the CNT growth process, the inter-CNT spacing (as depicted in Figure 8.1) is significantly non-uniform, which is referred to as CNT density variations. Such non-uniformity directly leads to variations in CNT count among CNFETs, consequently introducing larger variations in the drive current. It also brings the possibility of non-functional CNFETs with no CNT in their channels, resulting in yield loss. In the stochastic model of the density variations, the random CNT growth can be modelled by a 1-D renewal process. Consequently, the CNT count distribution induced by density variations can be approximated by Gaussian distribution as derived in [22].

- **m-CNT removal**: Depending on how the CNT is hypothetically formed by wrapping a graphene sheet, the geometry of CNTs has numerous flavours, also known as chirality. However, only two-thirds of the chiralities form semiconducting CNTs (s-CNT), while the other one-third lead to zero or near-zero bandgap, forming metallic CNTs (m-CNT) [194]. In CNFETs, these undesirable m-CNTs are randomly distributed among s-CNTs, causing source-to-drain shorts which result in excessive leakage current and incorrect circuit functionality. Thus, techniques have been developed to selectively remove m-CNTs in fabrication, while a small portion of s-CNTs are removed at the same time. The removal causes extra CNT count variations on top of the aforementioned density variations [189]. In modelling, the remaining s-CNTs still follow the renewal process, so the Gaussian CNT count distribution still holds [22]

The sources of individual CNT variations are the sources that lead to different electrical properties from one CNT to another, such as variations in diameter, alignment, and doping, which we will introduce as follows:

- **CNT diameter variations**: Since the diameters of CNTs are determined by chirality, CNT diameter variations can affect many CNFET properties, such as the bandgap, V_{th}, and I_{on}. The I_{on} for a single CNT is highly sensitive to CNT diameter variations. Nevertheless, for multi-CNT devices, the variability induced by diameter variations is largely reduced due to the averaging effect [189].

- **Other sources**: Due to the imperfections of the CNT fabrication process, there is plenty of non-uniformity among individual CNTs, such as alignment and doping. However, the impacts of these variations on device parameters are mitigated by the averaging effect of multiple CNTs, so they only account for a negligible portion of the total variability.

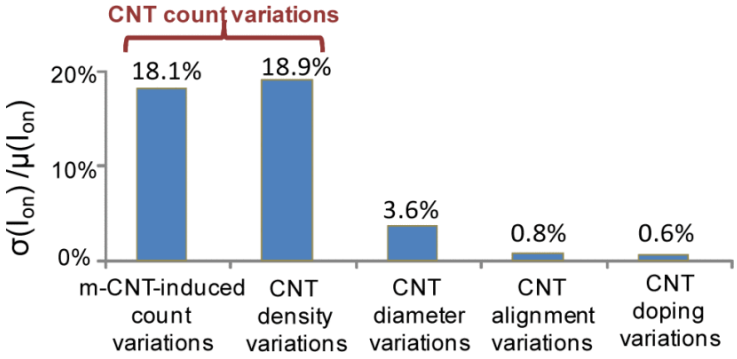


Figure 8.4: Simulated I_{on} variability of minimum-width CNFET at 32nm technology node, contributed by various variability sources. Replotted from [24]

Figure 8.4 shows the simulated variability contribution in I_{on} by different sources [24]. The numbers show that the sources of CNT count variations are the dominant sources of variability, in which both density variations and m-CNT removal have significant contributions. In contrast, the contributions of the sources of individual CNT variations are relatively small since their impacts are alleviated by the averaging effect of multiple CNTs [189].

Though the goal of this research is to analyze the total impact of all CNFET variability sources on processor performance, it is not feasible up to now since practical variability models have yet to be developed for many of the addressed variability sources. Nevertheless, the analytical model of the CNT count variations, which are the major variability sources, has been well established in [22]. The CNT count variability model is based on 1-D renewal process, including both the density variation and m-CNT removal in the same mathematical model. Thus, the following research utilizes this model and focuses only on the impacts of the sources of CNT count variations (CNT density variations, m-CNT removal), as indicated in Figure 8.3. The impacts from the sources of individual CNT variations are neglected. Due to the dominance of CNT count variations (88% among the variability sources evaluated in Figure 8.4), the results will only have limited error margins compared to the total CNFET variability. Hence, the comparison with the silicon CMOS technology still makes sense, so it will be provided as well. The details of the experiment are given in the following sections.

8.3 Processor-level Evaluation of Performance Variability

Variability has been identified as the main challenge to the realization of large-scale CNFET electronic systems. However, to the best of our knowledge, there is no published analysis of the CNFET variability impact on large-scale circuit designs. To fill this gap, we have experimented on the time-zero performance variability of processor-level CNFET integrated circuit designs induced by device-level CNFET variability. Moreover, we compared the performance variability of the CNFET processor-level designs with the counterparts of silicon CMOS technology, to evaluate the feasibility of CNFET as the next-generation technology of large-scale electronic systems replacing silicon CMOS. The experiments are set up based on projected technologies of the most advanced 2nm node with statistical device models and interconnect assumptions.

8.3.1 Technology Set-ups and Variability Models

Both CNFET and silicon CMOS technologies are set up to reflect the most advanced 2nm technology node with projections to the best of our knowledge. To have fair comparisons between both technologies, they are assumed to use the same middle-end-of-line (MEOL) and back-end-of-line (BEOL) interconnect technology and dimensions. The front-end-of-line (FEOL) is set up to their respective projections of the most probable technology option for the 2nm or equivalent node. The variability models are set up based on the measurements of existent technologies, scaled with the established variability scaling models for CNFET and silicon CMOS.

The reference silicon CMOS technology is based on the IMEC 2nm nanosheet technology [195], with the assumptions of 4 stacked sheets per device (N_{NS}), 12nm sheet width (W_{NS}), 5nm sheet height (H_{NS}), and 14nm gate length (Lg). The nominal supply voltage is 0.7V and the devices opt for the low threshold voltage (LVT) option. The cell libraries were generated based on the 5-Track standard cell layouts and the buried power rail (BUR) structure, specified in [196]. The cell library has 114 cells, containing the essential cells required for digital circuits. The BEOL assumptions also reflect the projection of the 2nm technology, shown in Table 8.1.

The time-zero variability of silicon CMOS addressed in this experiment is the variability in V_{th}, which is the dominant contributor to the device performance variations. The aggregated effects of V_{th} variations from all variability sources (e.g. RDF, MGG, as discussed in Section 2.1.1) can be modeled by Gaussian

Layer	Pitch (nm)	Width (nm)	Spacing (nm)
MBUR	90	33	57
MINT	18	10	8
M1	28	16	12
M2	18	10	8
M3	28	16	12
M4	28	16	12
M5	28	16	12
M6	80	40	40
M7	80	40	40
M8	80	40	40
M9	80	40	40
M10	80	40	40
M11	80	40	40
M12	80	40	40

Table 8.1: BEOL assumptions in physical designs

distribution with the technology-dependent coefficient of A_{VT}. By the definition of A_{VT}, the standard deviation of the V_{th} of a group of devices can be modelled as below [197]:

$$\sigma(V_{th}) = \frac{A_{VT}}{\sqrt{2W_{eff}L_g}} \tag{8.1}$$

where W_{eff} denotes the effective width of the device, which can be approximated by the following equation for nanosheet devices:

$$W_{eff} = N_{NS}(2W_{NS} + 2H_{NS}) \tag{8.2}$$

The industrial measurements of device V_{th} mismatches show that the A_{VT} coefficient is approximately 1.0 (mV $\cdot$ μm) in advanced technology [198]. By calculation, we can obtain $\sigma(V_{th})$ to be about 16mV.

The CNFET technology is based on the 2nm-equivalent technology option with the best EDP reported in [199], which is of the Gate-All-Around (GAA) structure with 1.0nm CNT diameter, 14nm gate length, 500 CNTs/μm CNT density, and 2 stacked layers. The nominal supply voltage is 0.45V. In this setup, the device size and the leakage current of CNFETs are comparable to those of the reference 2nm silicon CMOS technology. The CNFET cell library is based on the silicon CMOS cell library with the same cell layout set, except

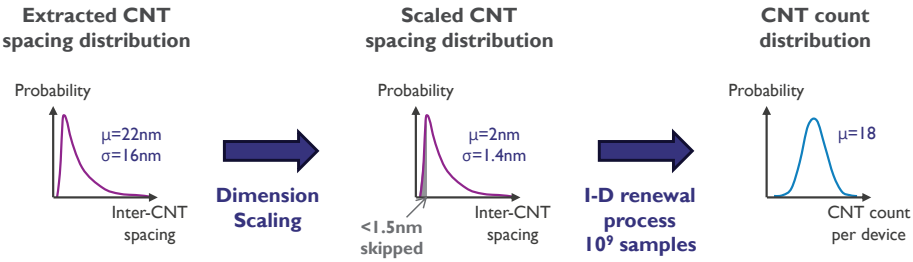


Figure 8.5: Generation process of the distribution of CNT count per device

that the FEOL layers are replaced by the CNFET counterparts. To minimize the influence of the interconnects, the MEOL and BEOL assumptions are the same as the silicon CMOS technology.

For the time-zero variability of CNFETs, we accounted for the sources of CNT count variations (CNT density variations, m-CNT removal) as the modeled variability source in this research. As shown in Section 8.2, the CNT count variations are the dominant source of variability, so the loss of accuracy is limited by neglecting other sources. The CNT count variation model is based on the probability framework of [22], which aims to derive the probability distribution of the number of CNTs in one CNFET device. In this framework, both CNT density variations and m-CNT removal are treated as the result of the 1-D renewal process of the spacings of CNT placement and removal.

The generation process of the distribution of CNT count per device is depicted in Figure 8.5. The distribution of inter-CNT spacings is extracted experimentally from the field data of fabricated CNTs of larger dimensions, in which the spacings are 22nm on average and 16nm in standard deviation [22]. To be on par with the 2nm technology node, the inter-CNT spacing distribution is scaled under the assumption of *index of dispersion for count* (IDC) being 0.5, resulting in 2nm on average and 1.4nm in standard deviation. Then, to obtain the detailed distribution of the CNT count, Monte-Carlo simulation of the 1-D renewal process was performed to mimic the process of random CNT placement on an active area of 40nm in width. One billion random device samples are generated in the simulation in order to derive the CNT count distribution up to $\pm 6\sigma$ quantiles. In the renewal process, the CNT spacings lower than 1.5nm are skipped since with a CNT diameter of approximately 1nm, the CNTs are not possible to be so close to each other. The resulting CNT count distribution has 18 CNTs per device on average and 10 and 25 CNTs per device on the $\pm 6\sigma$ quantiles, as summarized in Table 8.2.

Corner	CNFET		Silicon CMOS
	CNT count	Effective CNT pitch (nm)*	V_{th} shift (mV)
-6σ	10	4.0	100
-3σ	14	2.9	50
Nominal	18	2.2	0
$+3\sigma$	21	1.8	-50
$+6\sigma$	25	1.5	-100

* Effective CNT pitch = Width of active area / CNT count

Table 8.2: Device parameters of each design corner of CNFET and silicon CMOS technologies

8.3.2 Experimental Setup

Figure 8.6 shows the overall simulation of this experiment. In the general experimental setup, we have adopted the corner-based design flow, which is well-established for state-of-the-art digital circuit and system designs. The processor-level RTL circuit used in the analysis is a commercial 64-bit ARM processor core microarchitecture suitable for smartphones, embedded devices, and energy-efficient cores in heterogeneous multi-core architectures of high-performance processors [200, 201]. Here, we use a different design from the open-source RISC-V Ariane core used in the experiments of previous chapters, since not only the 64-bit ARM processor core is larger, more sophisticated, and thus more representative, but also this experiment can be performed without knowing or changing the architecture inside the core.

The device-level variability is extracted into different design corners, with which the processor-level integrated circuits can be designed and analyzed at different time-zero variability levels. Five corners were created for each technology, corresponding to the -6σ, -3σ, average (nominal), $+3\sigma$, and $+6\sigma$ quantiles of the device-level variability distribution respectively. According to the variability models in the last subsection, we project the CNT count of CNFET devices and the V_{th} shift of silicon CMOS devices for each quantile, summarized in Table 8.2.

Subsequently, per-corner compact models are generated by varying from the pre-existing nominal compact models (CNFET [199], silicon CMOS [195]), according to their respective device parameters (CNT count of CNFET, V_{th} shift of silicon CMOS). With these compact models, per-corner cell libraries are characterized from the cell library layouts to be used in logic synthesis and place & route (P&R).

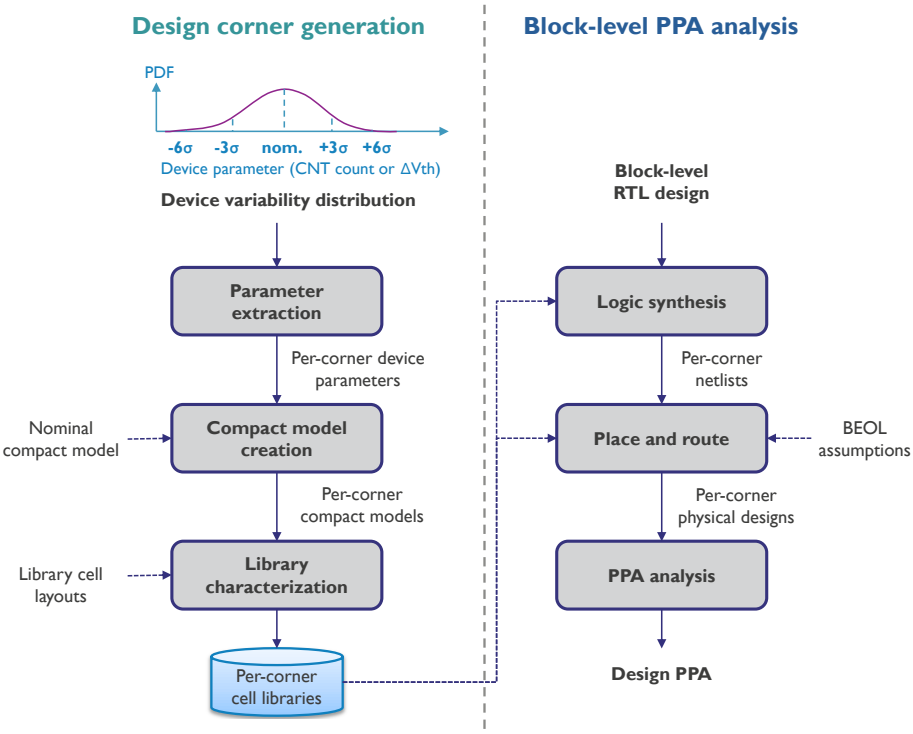


Figure 8.6: Simulation flow of processor-level variability evaluation of CNFET and silicon CMOS technology

For each design corner, we performed logic synthesis and place & route (P&R) on the same processor-level RTL circuit design using the corresponding cell library. The P&R was performed under 70% target utilization. Multiple runs were performed for each corner with different frequency constraints to find the maximum achievable frequency for each corner, generating physical designs with different speed, area, and energy. After P&R, the criteria of a valid physical design are to have fewer than 2000 timing violating paths and fewer than 10 design rule violations (DRC), since in practice this amount of violations can be manually solved by experienced engineers. Finally, all designs are analyzed to obtain their power-performance-area (PPA) differences, and only the Pareto-optimal designs of each corner in terms of frequency and energy are considered valid.

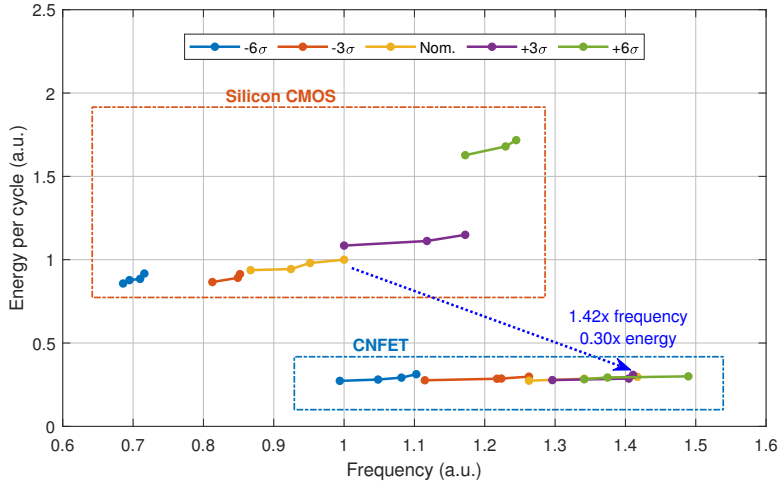
8.3.3 Results and Discussion

In the following, simulation results of the performance variability of both CNFET designs and silicon CMOS designs are shown. Though not all of the CNFET variability sources are considered, the major CNFET variability sources (CNT count variations) are incorporated into the results, in which 88% of the I_{on} variability are included, as indicated in Figure 8.4. Since the delays of logic gates are approximately inversely proportional to I_{on} [42], it is reasonable to presume that the errors in the performance variability of the circuits are within a limited range (<30%). Hence, the absolute numbers can still be used for proper comparison with the silicon CMOS technology.

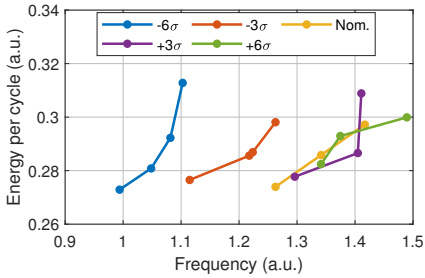
Figure 8.7a shows the achieved frequency and energy per cycle of the physical designs of each corner and each technology, normalized to the same arbitrary units (a.u.). The achieved frequency is calculated as the reciprocal of the critical path delay, and the energy per cycle is calculated as the total power under 0.1 switching activity, divided by the frequency. In general, the CNFET designs reach much higher frequency and lower energy than the silicon CMOS designs. Comparing the nominal-corner max-frequency designs of both technologies shows that the CNFET design reaches 1.42× frequency and 0.30× energy to the corresponding silicon CMOS design. The maximum achieved frequency of each corner is summarized in Figure 8.8a. Even the CNFET design of the worst -6σ corner achieves a higher frequency than the nominal silicon CMOS design, which shows the superiority in the performance of CNFET designs.

As far as performance variability is concerned, in Figure 8.7a both technologies generally show wide dispersion in performance among different variability conditions. Figure 8.7b & 8.7c zoom in on the plots of performance and energy variability of each technology. A particular trend of CNFET shown in Figure 8.7b is the little performance dispersion among fast corners (nominal, +3σ, +6σ), which can be attributed to the degradation of driving current in each CNT due to high gate coupling in small inter-CNT spacing [199]. As a result, the aggregated CNT currents of one CNFET device saturate, and increased CNT count no longer leads to better performance.

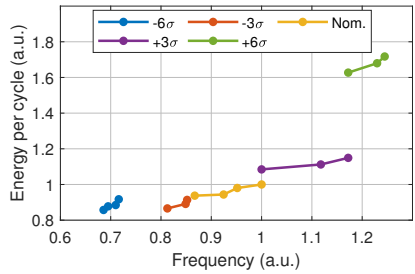
Figure 8.8b summarizes the performance variability of both technologies in terms of the per-corner maximum achieved frequency normalized to respective nominal corners. In general, the CNFET designs show smaller performance variability than the silicon CMOS designs as indicated by the frequency spread of 27.2% among CNFET corners versus 52.7% among silicon CMOS corners. In the worst -6σ corners, the performance degradation of CNFET (-22.3%) is also smaller than the one of silicon CMOS (-28.4%). The small performance dispersion among corners can be supportive of the feasibility of CNFETs in



(a)



(b)



(c)

Figure 8.7: Energy v.s. achieved frequency of the Pareto-optimal physical designs of each corner, normalized to the same arbitrary units. (a) All curves (b) CNFET technology (c) 2nm silicon CMOS technology

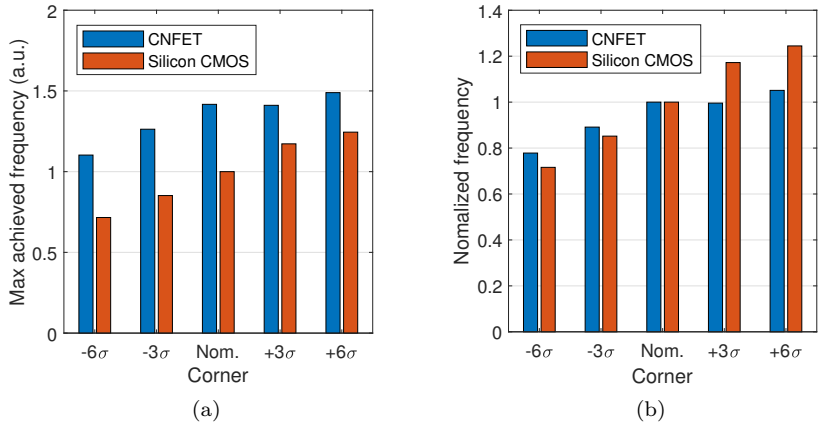


Figure 8.8: Summary of speed and performance variability of CNFET and silicon CMOS (a) maximum achieved frequency (b) maximum achieved frequency normalized to respective nominal corners

composing large-scale integrated circuits.

8.4 Mitigation Cost Estimation

The variability analysis in the last section implies a scenario that DVFS is enabled and the clock frequency of each chip or processor core is calibrated to its own maximum achieved frequency at the same voltage level. This setup of DVFS operating points is known as the Single-Voltage/Multiple-Frequency (SVMF) scheme [202]. However, the non-uniform clock frequencies cause performance variability. To mitigate performance variability, the DVFS points can opt for the Multiple-Voltage/Single-Frequency (MVSF) scheme instead, in which the supply voltage of each chip or processor core is adjusted to support the same frequency mode [202]. The mitigation costs are indicated by the voltage, frequency, and energy differences between the two schemes of DVFS points. The reposition of DVFS points from SVMF to MVSF is depicted in Figure 8.9.

To estimate the mitigation cost, first we need to know the amount of voltage and frequency scaling in the shift of DVFS points. The voltage-frequency relations of CNFET designs are projected by the DVFS points of the 3nm Ariane core fast ALU design we used in the previous experiments of mitigation approaches,

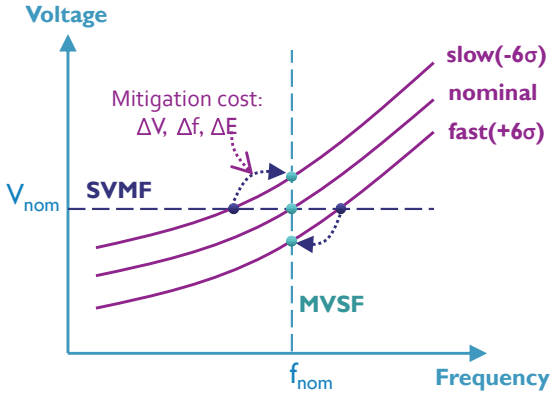


Figure 8.9: Reposition of DVFS operating points from the SVMF scheme to the MVSF scheme. The mitigation costs are indicated by the voltage, frequency, and energy differences between the two sets of DVFS points.

specified in Table 5.1. To account for the technology difference, the voltage levels are scaled proportionally with respect to the nominal supply voltages of the technologies, and the frequency levels are scaled proportionally with respect to the frequencies under the nominal supply voltage conditions. The scaling of energy consumption is proportional to V^2, same as the method described in Section 5.1. To minimize the energy cost, among the physical designs in Figure 8.7b, the one with the smaller energy consumption after scaling is selected for each corner. After mitigation, the frequencies of all designs are aligned to the maximum achieved frequency in the nominal corner.

Figure 8.10 shows the projected DVFS curves and the DVFS operating points before and after mitigation. As shown in Figure 8.10a, to transfer from the SVMF scheme to the MVSF scheme, the supply voltage of the -6σ corner design is significantly raised to 0.60V. On the contrary, the supply voltage of the +6σ corner design is only slightly reduced to 0.44V, resulting a 0.16V voltage spread over all corners. As for the energy consumption shown in Figure 8.10b, the energy of the -6σ corner design is increased by 80%, while in the +6σ corner the energy is only decreased by 6%. The significant energy cost results in a large spread in energy consumption among all corners, in which the -6σ corner design is 1.9 times as energy consuming as the +6σ corner design. Despite the energy overhead, the -6σ CNFET design still consumes smaller energy than the nominal silicon CMOS design by a ratio of 2.1×. Therefore, the energy consumptions of CNFET designs are still substantially smaller than silicon CMOS designs despite the variability.

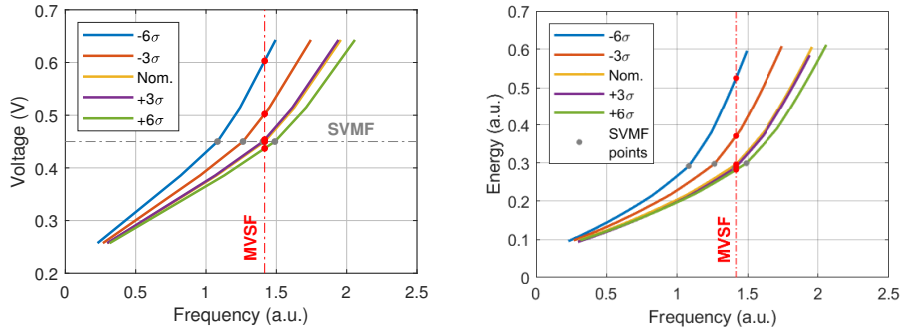


Figure 8.10: The DVFS relations of each corner of CNFET designs, including the operating points of both SVMF and MVSF schemes (a) supply voltage v.s. clock frequency (b) energy per cycle v.s. clock frequency

8.5 Discussion

In Section 8.3.3, we have shown promising numbers in which the performance variability of CNFET designs (27.2%) is smaller than the one of silicon CMOS designs (52.7%). Since the major variability sources (CNT density variations, m-CNT removal) are included in the simulation with limited error margins, we expect that the total CNFET performance variability is likely to be comparable to or smaller than the silicon CMOS performance variability.

Then, in Section 8.3.3, we have further shown the voltage and energy spread of CNFET designs after performance variability mitigation, in which the maximum energy consumption of CNFET designs still shows significantly smaller energy by 2.1× compared to the nominal silicon CMOS design. These numbers indicate that the CNFET performance variability can be properly handled by the mitigation approach, and the post-mitigation CNFET designs are still superior to silicon CMOS in performance and energy consumption.

While past research on CNFET variability in both measurements and simulations has led to diverse numbers, the acquired magnitudes of variability are highly dependent on the fabrication process or technology assumptions [189]. Nevertheless, our work is based on the most probable projections of technology to the best of our knowledge, so the results would be close to the ones of potential future CNFET technology. Still, there is ongoing research on sophisticated and complete CNFET variability models, which are required to improve the

coverage of variability sources in the simulation and eliminate the errors as much as possible. We look forward to the research on the CNFET variability, which will eventually provide a complete theoretical basis for the arguments on the feasibility of CNFETs.

8.6 Summary

In this chapter, the impact of CNFET variability on processor-level designs is evaluated. First, a comprehensive introduction of the CNFET is given, including its superiority of electrical properties over silicon CMOS, as well as the main device-level variability sources such as CNT density, m-CNTs, and CNT diameter. Then, with the corner-based design flow, we have generated the physical designs of a 64-bit ARM processor core microarchitecture under the circumstances of the different quantiles (average, $\pm 3\sigma$, and $\pm 6\sigma$) of the device-level variability distribution. The results show that in the worst -6σ corners, the performance degradation of CNFET is smaller than the one of silicon CMOS (22.3% v.s. 28.4%). In addition, the frequency spread of CNFET designs among all corners is smaller than the silicon CMOS design (27.2% v.s. 52.7%). To mitigate such performance variability, the DVFS operating points can transfer from the SVMF scheme to the MVSF scheme, resulting in a higher supply voltage of 0.60V and an energy overhead of 80% for the CNFET design of the slowest -6σ corner.

Chapter 9

Conclusion and Future Work

9.1 Main Messages

The main focus of this thesis has been on the performance variability mitigation approaches to ensuring timing guarantees. On this topic, we conclude the general messages of this dissertation as the following:

- In Chapter 2, we discussed a wide range of variability sources from the device/circuit, architecture, and software levels. With the development of technology and architecture, variability becomes larger and more complex. Therefore, the guardbanding approach is no longer sufficient, and run-time mitigation approaches are applied to effectively tackle the variability.
- In Chapter 3, we discussed the principles and the components of performance variability mitigation approaches. The state-of-the-art approaches use the WCET-based algorithm and the DVFS knob. However, they are no longer sufficient for highly dynamic workloads, since WCET models are too simplified for modern complicated architectures and DVFS has long switching times.
- The *workload-dependent multi-timescale mitigation approach* proposed in Chapter 4 can ensure timing guarantees with two features. The HDP knob pushes the time granularity to the sub-millisecond level, and the dynamic-scenario-based algorithm enables run-time calculation of the tight upper bound of execution time. Then in Chapter 5, we showed that this approach can ensure full timing guarantees with 32% energy saving

compared to the conventional WCET-based method with a single DVFS knob

- The *cross-layer variability mitigation framework* proposed in Chapter 6 can be a joint effort of individual mitigation approaches for different objectives in different layers, ensuring multiple reliability objectives in a cost-efficient way. In our framework, we use three mitigation approaches (circuit-level variability mitigation, register-level checkpointing and rollback-recovery, and processor-level performance variability mitigation) to simultaneously ensure functional correctness and timing guarantees.
- In Chapter 7 we showed that in a multi-objective reliability solution, interdependencies exist among different reliability objectives. In our cross-layer mitigation framework, the phenomenon of the *error rate wall* defines the maximum error rate which can be tolerated while ensuring timing guarantees. Furthermore, by optimizing system parameters (e.g. checkpointing granularity), it is possible to improve the error rate wall.
- In Chapter 8, the mitigation framework enables comparison of variability impact between CNFET and silicon CMOS technologies in the processor level. The results show that in terms of the performance variability induced by process variations, CNFET designs have smaller variability than silicon CMOS designs. Although eliminating the performance variability comes with a great energy overhead, the CNFET designs still consume much smaller energy than silicon CMOS designs.

9.2 Contributions

In the following, we conclude the main contributions of our research:

- In Chapter 4 & 5, we have proposed a novel workload-dependent multi-timescale mitigation approach, which can mitigate performance variability with guarantees as required in all the three application classes discussed earlier: time-critical, time-sensitive, and general with latency bounds. It tackles architecture-level and software-level performance variability of different timescales while minimizing energy consumption. This approach has two salient features. First, a novel fast-switching system knob, HDP, is implemented which can push the time granularity of mitigation to the sub-millisecond level. Second, the scheduling algorithm is designed on the principle of dynamic scenarios, which can derive the tight upper bound of execution time to prevent deadline misses. The presented work has been published in the authored work of [136, 137].

- In Chapter 5, we have developed a cycle-accurate simulation flow to evaluate the mitigation approach. Unlike the simplified processor models used in normal real-time scheduling research, this flow can capture the performance variability down to cycle-level granularity. Important processor metrics (execution time, energy consumption, etc.) can also be reported by the flow to evaluate the reliability and costs of different performance variability mitigation approaches. The presented work has been published in the authored work of [136, 137].

- In Chapter 6 & 7, a cross-layer reliability and variability mitigation framework is proposed to ensure both function correctness and timing guarantees at the same time. The framework consists of three mitigation approaches, including a circuit-level variability mitigation approach, register-level checkpointing and rollback-recovery, and processor-level performance variability mitigation. Part of the presented work has been published in the co-authored work of [147], and the complete work is planned for publication.

- In Chapter 7, co-simulation of the layers in the aforementioned framework shows the holistic system impact and the system reliability limit in ensuring the dual reliability objectives. Also, by this cross-layer analysis, the interdependencies of different reliability objectives are revealed. Specifically, the system demonstrates the phenomenon of the "error rate wall," the error rate limit beyond which the functional correctness and timing guarantees cannot be ensured at the same time. We also show that changing system parameters improve the error rate wall. Therefore, the overall system reliability can be enhanced by optimizing system parameters. Part of the presented work has been published in the co-authored work of [147], and the complete work is planned for publication.

- In Chapter 8, we further extend the use of the mitigation framework to compare the variability impacts of two different technologies, the emerging CNFET and the silicon CMOS. As a result, the CNFET processor designed with the 6σ slow design corner achieves 22.3% lower clock frequency than the one with the nominal design corner. This performance variability is smaller than the counterpart of silicon CMOS. To mitigate the performance variability to run at the same clock frequency as the nominal-corner CNFET processor, the slow-corner CNFET processor needs to raise the supply voltage, which comes with an 80% energy cost. The presented work is planned for publication in the future.

9.3 Perspective

In general, our research serves as the preliminary steps beyond the conventional WCET-based DVFS scheduling approach for ensuring timing guarantees. There is plenty of room left for extending our work to an all-around solution. Therefore, we identified the important continuation and extension of this work as follows:

- The dynamic scenario principles described in Chapter 4 can be potentially instantiated on more complicated architectures, such as multi-core processors and GPU, to develop architectural-specific mitigation approaches for performance variability. Since these architectures are inherently equipped with multiple functional units of various types, the area cost and design spaces of the knobs could even be smaller than single-core processors. In reality, there is also a need to ensure timing guarantees for these architectures that run complex workloads, such as car auto-driving systems. Hence, further research is required to develop mitigation approaches for these systems.
- The HDP knob proposed in Chapter 4 is an instantiation of how the inherent performance-energy trade-off in microarchitecture can be exploited to realize a fast-switching system knob. We suppose there is more potential for designing other microarchitectural knobs on this principle in processor cores, memory, and interconnect networks. In addition, supports of EDA tools are also crucial for optimizing the energy efficiency of these knobs. Therefore, we look forward to the extension of the concept of the microarchitectural knobs.
- We have demonstrated the dynamic-scenario-based algorithm for timing guarantees of a job sequence in Chapter 4. There is plenty of room for generalizing the algorithms for various purposes. For example, adapting the algorithm to the constraints of maximum deadline misses or maximum latency could make it suitable for time-sensitive and general applications. Also, by integrating the algorithm with real-time scheduling algorithms, a timing-guaranteed solution for multiple tasks in multi-core processors can be realized.
- The cycle-accurate simulation flow in Chapter 5 can be extended to simulate the performance variability and mitigation of more sophisticated architectures, such as including multi-level caches, multi-core processors, system-on-chip, etc. However, huge computation resource is required for performing this kind of simulation, which is a challenge to overcome.
- In Chapter 6, the proposed cross-layer reliability and variability mitigation framework covers the levels from the circuit to the processor. Beyond

our framework, the higher levels (e.g. macroarchitecture, software) are involved with very complex systems with a large amount of virtualisation and dynamism. Therefore, they are also relevant to explore in order to reduce the complexity to ensure timing guarantees.

- The cross-layer analysis in Chapter 7 reveals how the requirements for functional correctness and timing guarantees can affect each other. We speculate that similar relations exist among other reliability objectives. Therefore, we recommend both theoretical and empirical studies to understand the interdependencies of all objectives (functional/parametric correctness, timing, thermal, lifetime), which will be a crucial step in the realization of multi-objective reliability solutions.
- The CNFET variability evaluation in Chapter 8 can be further extended in multiple ways. First, the inclusion of other CNFET variability sources (e.g. diameter, chirality) can improve the accuracy of the evaluation results. Furthermore, this variability evaluation methodology can be generalized, playing an important role in evaluating the practicality of new technologies in mass production.

In conclusion, while the abovementioned gaps are required to be filled, we look forward to the development of a general performance variability mitigation approach, which can ensure timing guarantees for highly reliable products.

Bibliography

[1] M. Bohr, "The evolution of scaling from the homogeneous era to the heterogeneous era," in *2011 International Electron Devices Meeting*, pp. 1.1.1–1.1.6, 2011.

[2] M. Noltsis, F. Catthoor, and D. Soudris, *Performance Variation in Digital Systems : Workload Dependent Modeling and Mitigation.* PhD thesis, National Technical University of Athens and KU Leuven, 2021.

[3] K. J. Kuhn, M. D. Giles, D. Becher, P. Kolar, A. Kornfeld, R. Kotlyar, S. T. Ma, A. Maheshwari, and S. Mudanai, "Process technology variation," *IEEE Transactions on Electron Devices*, vol. 58, no. 8, pp. 2197–2208, 2011.

[4] A. Asenov, A. Brown, J. Davies, S. Kaya, and G. Slavcheva, "Simulation of intrinsic parameter fluctuations in decananometer and nanometer-scale MOSFETs," *IEEE Transactions on Electron Devices*, vol. 50, no. 9, pp. 1837–1852, 2003.

[5] K. Ohmori, T. Matsuki, D. Ishikawa, T. Morooka, T. Aminaka, Y. Sugita, T. Chikyow, K. Shiraishi, Y. Nara, and K. Yamada, "Impact of additional factors in threshold voltage variability of metal/high-k gate stacks and its reduction by controlling crystalline structure and grain size in the metal gates," in *2008 IEEE International Electron Devices Meeting*, pp. 1–4, 2008.

[6] H. Nam and C. Shin, "Study of high-k/metal-gate work-function variation using rayleigh distribution," *IEEE Electron Device Letters*, vol. 34, no. 4, pp. 532–534, 2013.

[7] A. Muhtaroglu, G. Taylor, and T. Rahal-Arabi, "On-die droop detector for analog sensing of power supply noise," *IEEE Journal of Solid-State Circuits*, vol. 39, no. 4, pp. 651–660, 2004.

[8] B. Kaczer, T. Grasser, P. J. Roussel, J. Franco, R. Degraeve, L.-A. Ragnarsson, E. Simoen, G. Groeseneken, and H. Reisinger, "Origin of NBTI variability in deeply scaled pFETs," in *2010 IEEE International Reliability Physics Symposium*, pp. 26–32, 2010.

[9] T. Grasser, B. Kaczer, W. Goes, H. Reisinger, T. Aichinger, P. Hehenberger, P.-J. Wagner, F. Schanovsky, J. Franco, P. Roussel, and M. Nelhiebel, "Recent advances in understanding the bias temperature instability," in *2010 International Electron Devices Meeting*, pp. 4.4.1–4.4.4, 2010.

[10] D. Suggs, M. Subramony, and D. Bouvier, "The amd "Zen 2" processor," *IEEE Micro*, vol. 40, no. 2, pp. 45–52, 2020.

[11] H. Falk, S. Altmeyer, P. Hellinckx, B. Lisper, W. Puffitsch, C. Rochange, M. Schoeberl, R. B. Sørensen, P. Wägemann, and S. Wegener, "TACLeBench: A benchmark collection to support worst-case execution time research," in *16th International Workshop on Worst-Case Execution Time Analysis (WCET 2016)* (M. Schoeberl, ed.), vol. 55 of *OpenAccess Series in Informatics (OASIcs)*, (Dagstuhl, Germany), pp. 2:1–2:10, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2016.

[12] B. Haskell, P. Howard, Y. LeCun, A. Puri, J. Ostermann, M. Civanlar, L. Rabiner, L. Bottou, and P. Haffner, "Image and video coding-emerging standards and beyond," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 8, no. 7, pp. 814–837, 1998.

[13] RISC-V International, "Spike RISC-V ISA Simulator." online. https://github.com/riscv-software-src/riscv-isa-sim.

[14] K. V. Aadithya, A. Demir, S. Venugopalan, and J. Roychowdhury, "SAMURAI: An accurate method for modelling and simulating non-stationary Random Telegraph Noise in SRAMs," in *2011 Design, Automation & Test in Europe*, pp. 1–6, 2011.

[15] F. Catthoor and G. Groeseneken, "Will chips of the future learn how to feel pain and cure themselves?," *IEEE Design & Test*, vol. 34, no. 5, pp. 80–87, 2017.

[16] C. Bienia and K. Li, "PARSEC 2.0: A new benchmark suite for chip-multiprocessors," in *Proceedings of the 5th Annual Workshop on Modeling, Benchmarking and Simulation*, June 2009.

[17] B. K. Reddy, A. K. Singh, D. Biswas, G. V. Merrett, and B. M. Al-Hashimi, "Inter-cluster thread-to-core mapping and DVFS on heterogeneous multi-cores," *IEEE Transactions on Multi-Scale Computing Systems*, vol. 4, no. 3, pp. 369–382, 2018.

[18] F. Zaruba and L. Benini, "The cost of application-class processing: Energy and performance analysis of a linux-ready 1.7-GHz 64-bit RISC-V core in 22-nm FDSOI technology," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 11, pp. 2629–2640, 2019.

[19] "Recommendation ITU-T G.722: 7 kHz audio-coding within 64 kbit/s," tech. rep., ITU-T Test Signals for Telecommunication Systems, 2012.

[20] M. S. Floyd, A. J. Drake, R. W. Berry, H. Chase, R. Willaman, and J. Pena, "Voltage droop reduction using throttling controlled by timing margin feedback," in *2012 Symposium on VLSI Circuits (VLSIC)*, pp. 96–97, 2012.

[21] M. M. Sabry, D. Atienza, and F. Catthoor, "OCEAN: An optimized HW/SW reliability mitigation approach for scratchpad memories in real-time SoCs," *ACM Transactions on Embedded Computing Systems*, vol. 13, apr 2014.

[22] J. Zhang, N. P. Patil, A. Hazeghi, H.-S. P. Wong, and S. Mitra, "Characterization and design of logic circuits in the presence of carbon nanotube density variations," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 8, pp. 1103–1113, 2011.

[23] G. Hills, M. G. Bardon, G. Doornbos, D. Yakimets, P. Schuddinck, R. Baert, D. Jang, L. Mattii, S. M. Y. Sherazi, D. Rodopoulos, R. Ritzenthaler, C.-S. Lee, A. V.-Y. Thean, I. Radu, A. Spessot, P. Debacker, F. Catthoor, P. Raghavan, M. M. Shulaker, H.-S. P. Wong, and S. Mitra, "Understanding energy efficiency benefits of carbon nanotube field-effect transistors for digital VLSI," *IEEE Transactions on Nanotechnology*, vol. 17, no. 6, pp. 1259–1269, 2018.

[24] J. Zhang, N. Patil, H.-S. Philip Wong, and S. Mitra, "Overcoming carbon nanotube variations through co-optimized technology and circuit design," in *2011 International Electron Devices Meeting*, pp. 4.6.1–4.6.4, 2011.

[25] S. Park, J. Park, D. Shin, Y. Wang, Q. Xie, M. Pedram, and N. Chang, "Accurate modeling of the delay and energy overhead of dynamic voltage and frequency scaling in modern microprocessors," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 5, pp. 695–708, 2013.

[26] G. E. Moore, "Cramming more components onto integrated circuits," *Electronics Magazine*, vol. 38, no. 8, pp. 114–117, 1965.

[27] J. L. Hennessy and D. A. Patterson, *Computer Architecture, Sixth Edition: A Quantitative Approach*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 6th ed., 2017.

[28] G. D. Hutcheson, "Moore's law, lithography, and how optics drive the semiconductor industry," in *Extreme Ultraviolet (EUV) Lithography IX* (K. A. Goldberg and N. M. Felix, eds.), vol. 10583 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, p. 1058303, Mar. 2018.

[29] B. Kaczer, S. Mahato, V. Valduga de Almeida Camargo, M. Toledano-Luque, P. J. Roussel, T. Grasser, F. Catthoor, P. Dobrovolny, P. Zuber, G. Wirth, and G. Groeseneken, "Atomistic approach to variability of bias-temperature instability in circuit simulations," in *2011 International Reliability Physics Symposium*, pp. XT.3.1–XT.3.5, 2011.

[30] S. Eyerman, J. Smith, and L. Eeckhout, "Characterizing the branch misprediction penalty," in *2006 IEEE International Symposium on Performance Analysis of Systems and Software*, pp. 48–58, 2006.

[31] J. Abella, C. Hernandez, E. Quiñones, F. J. Cazorla, P. R. Conmy, M. Azkarate-askasua, J. Perez, E. Mezzetti, and T. Vardanega, "WCET analysis methods: Pitfalls and challenges on their trustworthiness," in *10th IEEE International Symposium on Industrial Embedded Systems (SIES)*, pp. 1–10, 2015.

[32] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström, "The worst-case execution-time problem—overview of methods and survey of tools," *ACM Transactions on Embedded Computing Systems*, vol. 7, may 2008.

[33] M. Bambagini, M. Marinoni, H. Aydin, and G. Buttazzo, "Energy-aware scheduling for real-time systems: A survey," *ACM Transactions on Embedded Computing Systems*, vol. 15, jan 2016.

[34] B. Ranjbar, A. Hosseinghorban, M. Salehi, A. Ejlali, and A. Kumar, "Toward the design of fault-tolerance-aware and peak-power-aware multicore mixed-criticality systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 5, pp. 1509–1522, 2022.

[35] M. Nourani and A. Radhakrishnan, "Testing on-die process variation in nanometer VLSI," *IEEE Design & Test of Computers*, vol. 23, no. 6, pp. 438–451, 2006.

[36] C. Shin, *Variation-Aware Advanced CMOS Devices and SRAM*. Dordrecht: Springer Netherlands, 2016.

[37] K. J. Kuhn, "Reducing variation in advanced logic technologies: Approaches to process and design for manufacturability of nanoscale CMOS," in *2007 IEEE International Electron Devices Meeting*, pp. 471–474, 2007.

[38] H. Dadgour, K. Endo, V. De, and K. Banerjee, "Modeling and analysis of grain-orientation effects in emerging metal-gate devices and implications for SRAM reliability," in *2008 IEEE International Electron Devices Meeting*, pp. 1–4, 2008.

[39] X. Wang, A. R. Brown, N. Idris, S. Markov, G. Roy, and A. Asenov, "Statistical threshold-voltage variability in scaled decananometer bulk HKMG MOSFETs: A full-scale 3-D simulation scaling study," *IEEE Transactions on Electron Devices*, vol. 58, no. 8, pp. 2293–2301, 2011.

[40] A. R. Brown, N. M. Idris, J. R. Watling, and A. Asenov, "Impact of metal gate granularity on threshold voltage variability: A full-scale three-dimensional statistical simulation study," *IEEE Electron Device Letters*, vol. 31, no. 11, pp. 1199–1201, 2010.

[41] T. Sakurai and A. Newton, "Alpha-power law MOSFET model and its applications to CMOS inverter delay and other formulas," *IEEE Journal of Solid-State Circuits*, vol. 25, no. 2, pp. 584–594, 1990.

[42] C. Hu, "Low-voltage CMOS device scaling," in *Proceedings of IEEE International Solid-State Circuits Conference - ISSCC '94*, pp. 86–87, 1994.

[43] C. Tirumurti, S. Kundu, S. Sur-Kolay, and Y.-S. Chang, "A modeling approach for addressing power supply switching noise related failures of integrated circuits," in *Proceedings Design, Automation and Test in Europe Conference and Exhibition*, vol. 2, pp. 1078–1083 Vol.2, 2004.

[44] S. Pant, D. Blaauw, V. Zolotov, S. Sundareswaran, and R. Panda, "Vectorless analysis of supply noise induced delay variation," in *ICCAD-2003. International Conference on Computer Aided Design (IEEE Cat. No.03CH37486)*, pp. 184–191, 2003.

[45] V. Venkatachalam and M. Franz, "Power reduction techniques for microprocessor systems," *ACM Computing Surveys*, vol. 37, p. 195–237, sep 2005.

[46] K. A. Bowman, C. Tokunaga, T. Karnik, V. K. De, and J. W. Tschanz, "A 22 nm all-digital dynamically adaptive clock distribution for supply voltage droop tolerance," *IEEE Journal of Solid-State Circuits*, vol. 48, no. 4, pp. 907–916, 2013.

[47] K. Bowman, J. Tschanz, C. Wilkerson, S.-L. Lu, T. Karnik, V. De, and S. Borkar, "Circuit techniques for dynamic variation tolerance," in *2009 46th ACM/IEEE Design Automation Conference*, pp. 4–7, 2009.

[48] D. Brooks and M. Martonosi, "Dynamic thermal management for high-performance microprocessors," in *Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture*, pp. 171–182, 2001.

[49] A. Iranfar, F. Terraneo, G. Csordas, M. Zapater, W. Fornaciari, and D. Atienza, "Dynamic thermal management with proactive fan speed control through reinforcement learning," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 418–423, 2020.

[50] M. J. Dousti and M. Pedram, "Power-aware deployment and control of forced-convection and thermoelectric coolers," in *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2014.

[51] P. Weckx, B. Kaczer, C. Chen, P. Raghavan, D. Linten, and A. Mocuta, "Relaxation of time-dependent NBTI variability and separation from RTN," in *2017 IEEE International Reliability Physics Symposium (IRPS)*, pp. XT–9.1–XT–9.5, 2017.

[52] T. Matsumoto, K. Kobayashi, and H. Onodera, "Impact of random telegraph noise on CMOS logic circuit reliability," in *Proceedings of the IEEE 2014 Custom Integrated Circuits Conference*, pp. 1–8, 2014.

[53] M. Cho, J.-D. Lee, M. Aoulaiche, B. Kaczer, P. Roussel, T. Kauerauf, R. Degraeve, J. Franco, L.-Å. Ragnarsson, and G. Groeseneken, "Insight into N/PBTI mechanisms in sub-1-nm-EOT devices," *IEEE Transactions on Electron Devices*, vol. 59, no. 8, pp. 2042–2048, 2012.

[54] K.-L. Chen, S. Saller, I. Groves, and D. Scott, "Reliability effects on MOS transistors due to hot-carrier injection," *IEEE Journal of Solid-State Circuits*, vol. 20, no. 1, pp. 306–313, 1985.

[55] C. Hu, S. C. Tam, F.-C. Hsu, P.-K. Ko, T.-Y. Chan, and K. Terrill, "Hot-electron-induced mosfet degradation - model, monitor, and improvement," *IEEE Journal of Solid-State Circuits*, vol. 20, no. 1, pp. 295–305, 1985.

[56] P. Heremans, R. Bellens, G. Groeseneken, and H. Maes, "Consistent model for the hot-carrier degradation in n-channel and p-channel MOSFETs," *IEEE Transactions on Electron Devices*, vol. 35, no. 12, pp. 2194–2209, 1988.

[57] S. Mukherjee, *Architecture Design for Soft Errors*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2008.

[58] J. H. Stathis, "Reliability limits for the gate insulator in cmos technology," *IBM Journal of Research and Development*, vol. 46, no. 2.3, pp. 265–286, 2002.

[59] D. Bekiaris, A. Papanikolaou, C. Papameletis, D. Soudris, G. Economakos, and K. Pekmestzi, "A temperature-aware time-dependent dielectric breakdown analysis framework," in *Integrated Circuit and System Design. Power and Timing Modeling, Optimization, and Simulation* (R. van Leuken and G. Sicard, eds.), (Berlin, Heidelberg), pp. 73–83, Springer Berlin Heidelberg, 2011.

[60] P. Roussel, R. Degraeve, G. Van den Bosch, B. Kaczer, and G. Groeseneken, "Accurate and robust noise-based trigger algorithm for soft breakdown detection in ultrathin gate dielectrics," *IEEE Transactions on Device and Materials Reliability*, vol. 1, no. 2, pp. 120–127, 2001.

[61] C. H. Yang, S. C. Chen, Y. S. Tsai, R. Lu, and Y.-H. Lee, "The physical explanation of TDDB power law lifetime model through oxygen vacancy trap investigations in HKMG NMOS FinFET devices," in *2017 IEEE International Reliability Physics Symposium (IRPS)*, pp. 3C–4.1–3C–4.6, 2017.

[62] K. Torii, H. Kitajima, T. Arikado, K. Shiraishi, S. Miyazaki, K. Yamabe, M. Boero, T. Chikyow, and K. Yamada, "Physical model of BTI, TDDB and SILC in HfO_2-based high-k gate dielectrics," in *IEDM Technical Digest. IEEE International Electron Devices Meeting, 2004.*, pp. 129–132, 2004.

[63] C.-K. Hu, K. P. Rodbell, T. D. Sullivan, K. Y. Lee, and D. P. Bouldin, "Electromigration and stress-induced voiding in fine Al and Al-alloy thin-film lines," *IBM Journal of Research and Development*, vol. 39, no. 4, pp. 465–497, 1995.

[64] J. Tao, N. Cheung, and C. Hu, "Electromigration characteristics of copper interconnects," *IEEE Electron Device Letters*, vol. 14, no. 5, pp. 249–251, 1993.

[65] V. Sukharev and F. N. Najm, "Electromigration check: Where the design and reliability methodologies meet," *IEEE Transactions on Device and Materials Reliability*, vol. 18, no. 4, pp. 498–507, 2018.

[66] S. Furber, "Microprocessors: The engines of the digital age," *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Science*, vol. 473, p. 20160893, Mar. 2017.

[67] M. B. Taylor, "Is dark silicon useful? harnessing the four horsemen of the coming dark silicon apocalypse," in *DAC Design Automation Conference 2012*, pp. 1131–1136, 2012.

[68] D. Patterson, "50 years of computer architecture: From the mainframe CPU to the domain-specific tpu and the open RISC-V instruction set," in *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*, pp. 27–31, 2018.

[69] D. Skinner and W. Kramer, "Understanding the causes of performance variability in HPC workloads," in *IEEE International. 2005 Proceedings of the IEEE Workload Characterization Symposium, 2005.*, pp. 137–149, 2005.

[70] S. McFarling, "Combining branch predictors," tech. rep., Digital Western Research Laboratory, 1993.

[71] N. Weste and D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*. USA: Addison-Wesley Publishing Company, 4th ed., 2010.

[72] S. K. Saha, "Modeling process variability in scaled CMOS technology," *IEEE Design & Test of Computers*, vol. 27, no. 2, pp. 8–16, 2010.

[73] K. Kang, S. Gangwal, S. P. Park, and R. Kaushik, "NBTI induced performance degradation in logic and memory circuits: how effectively can we approach a reliability solution?," in *2008 Asia and South Pacific Design Automation Conference*, pp. 726–731, 2008.

[74] C. R. Lefurgy, A. J. Drake, M. S. Floyd, M. S. Allen-Ware, B. Brock, J. A. Tierno, J. B. Carter, and R. W. Berry, "Active guardband management in Power7+ to save energy and maintain reliability," *IEEE Micro*, vol. 33, no. 4, pp. 35–45, 2013.

[75] K. A. Bowman, J. W. Tschanz, S.-L. L. Lu, P. A. Aseron, M. M. Khellah, A. Raychowdhury, B. M. Geuskens, C. Tokunaga, C. B. Wilkerson, T. Karnik, and V. K. De, "A 45 nm resilient microprocessor core for dynamic variation tolerance," *IEEE Journal of Solid-State Circuits*, vol. 46, no. 1, pp. 194–208, 2011.

[76] A. Deb, B. Vermeulen, and L. van Dijk, "Overview of health monitoring techniques for reliability," in *Proceedings of the Workshop on Early Reliability Modeling for Aging and Variability in Silicon Systems (ERMAVSS 2016)*, pp. 30–33, 2016.

[77] C. Gonzalez, M. Floyd, E. Fluhr, P. Restle, D. Dreps, M. Sperling, R. Rao, D. Hogenmiller, C. Vezyrtis, P. Chuang, D. Lewis, R. Escobar, V. Ramadurai, R. Kruse, J. Pille, R. Nett, P. Owczarczyk, J. Friedrich, J. Paredes, T. Diemoz, S. Islam, D. Plass, and P. Muench, "The 24-core POWER9 processor with adaptive clocking, 25-Gb/s accelerator links, and 16-Gb/s PCIe Gen4," *IEEE Journal of Solid-State Circuits*, vol. 53, no. 1, pp. 91–101, 2018.

[78] J. Tschanz, N. S. Kim, S. Dighe, J. Howard, G. Ruhl, S. Vangal, S. Narendra, Y. Hoskote, H. Wilson, C. Lam, M. Shuman, C. Tokunaga, D. Somasekhar, S. Tang, D. Finan, T. Karnik, N. Borkar, N. Kurd, and V. De, "Adaptive frequency and biasing techniques for tolerance to dynamic temperature-voltage variations and aging," in *2007 IEEE International Solid-State Circuits Conference. Digest of Technical Papers*, pp. 292–604, 2007.

[79] A. Grenat, S. Pant, R. Rachala, and S. Naffziger, "Adaptive clocking system for improved power efficiency in a 28nm x86-64 microprocessor," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pp. 106–107, 2014.

[80] K. Wilcox, R. Cole, H. R. Fair III, K. Gillespie, A. Grenat, C. Henrion, R. Jotwani, S. Kosonocky, B. Munger, S. Naffziger, R. S. Orefice, S. Pant, D. A. Priore, R. Rachala, and J. White, "Steamroller module and adaptive clocking system in 28 nm CMOS," *IEEE Journal of Solid-State Circuits*, vol. 50, no. 1, pp. 24–34, 2015.

[81] K. A. Bowman, S. Raina, J. T. Bridges, D. J. Yingling, H. H. Nguyen, B. R. Appel, Y. N. Kolla, J. Jeong, F. I. Atallah, and D. W. Hansquine, "A 16 nm all-digital auto-calibrating adaptive clock distribution for supply voltage droop tolerance across a wide operating range," *IEEE Journal of Solid-State Circuits*, vol. 51, no. 1, pp. 8–17, 2016.

[82] C. Takahashi, S. Shibahara, K. Fukuoka, J. Matsushima, Y. Kitaji, Y. Shimazaki, H. Hara, and T. Irita, "A 16nm FinFET heterogeneous nona-core SoC complying with ISO26262 ASIL-B: Achieving 10^{-7} random hardware failures per hour reliability," in *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, pp. 80–81, 2016.

[83] M. S. Floyd, P. J. Restle, M. A. Sperling, P. Owczarczyk, E. J. Fluhr, J. Friedrich, P. Muench, T. Diemoz, P. Chuang, and C. Vezyrtzis, "Adaptive clocking in the POWER9™ processor for voltage droop protection," in *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, pp. 444–445, 2017.

[84] S. Shibahara, C. Takahashi, K. Fukuoka, Y. Kitaji, T. Irita, H. Hara, Y. Shimazaki, and J. Matsushima, "A 16 nm FinFET heterogeneous nona-core soc supporting ISO26262 ASIL B standard," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 77–88, 2017.

[85] M. M. Sabry and D. Atienza, "Temperature-aware design and management for 3D multi-core architectures," *Foundations and Trends in Electronic Design Automation*, vol. 8, pp. 117–197, Jan 2014.

[86] T. Chantem, R. P. Dick, and X. S. Hu, "Temperature-aware scheduling and assignment for hard real-time applications on MPSoCs," in *2008 Design, Automation and Test in Europe*, pp. 288–293, 2008.

[87] D. Brooks and M. Martonosi, "Dynamic thermal management for high-performance microprocessors," in *Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture*, pp. 171–182, 2001.

[88] W. Lee, K. Patel, and M. Pedram, "Dynamic thermal management for MPEG-2 decoding," in *ISLPED'06 Proceedings of the 2006 International Symposium on Low Power Electronics and Design*, pp. 316–321, 2006.

[89] I. Yeo, C. C. Liu, and E. J. Kim, "Predictive dynamic thermal management for multicore systems," in *2008 45th ACM/IEEE Design Automation Conference*, pp. 734–739, 2008.

[90] Z. Liu, T. Xu, S. X.-D. Tan, and H. Wang, "Dynamic thermal management for multi-core microprocessors considering transient thermal effects," in *2013 18th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 473–478, 2013.

[91] E. Karl, D. Blaauw, D. Sylvester, and T. Mudge, "Multi-mechanism reliability modeling and management in dynamic systems," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 4, pp. 476–487, 2008.

[92] J. Srinivasan, S. Adve, P. Bose, and J. Rivers, "The case for lifetime reliability-aware microprocessors," in *Proceedings. 31st Annual International Symposium on Computer Architecture, 2004.*, pp. 276–287, 2004.

[93] C. Bolchini, M. Carminati, A. Miele, A. Das, A. Kumar, and B. Veeravalli, "Run-time mapping for reliable many-cores based on energy/performance trade-offs," in *2013 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS)*, pp. 58–64, 2013.

[94] M.-H. Haghbayan, A. Miele, A. M. Rahmani, P. Liljeberg, and H. Tenhunen, "Performance/reliability-aware resource management for many-cores in dark silicon era," *IEEE Transactions on Computers*, vol. 66, no. 9, pp. 1599–1612, 2017.

[95] A. S. Hartman and D. E. Thomas, "Lifetime improvement through runtime wear-based task mapping," in *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, CODES+ISSS '12, (New York, NY, USA), p. 13–22, Association for Computing Machinery, 2012.

[96] S. T. Kim, Y.-C. Shih, K. Mazumdar, R. Jain, J. F. Ryan, C. Tokunaga, C. Augustine, J. P. Kulkarni, K. Ravichandran, J. W. Tschanz, M. M. Khellah, and V. De, "Enabling wide autonomous DVFS in a 22 nm graphics execution core using a digitally controlled fully integrated voltage regulator," *IEEE Journal of Solid-State Circuits*, vol. 51, no. 1, pp. 18–30, 2016.

[97] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, AFIPS '67 (Spring), (New York, NY, USA), p. 483–485, Association for Computing Machinery, 1967.

[98] ARM, "big.LITTLE technology: The future of mobile," tech. rep., ARM white paper, 2013.

[99] P. Greenhalgh, "big.LITTLE processing with ARM Cortex-A15 & Cortex-A7," tech. rep., ARM white paper, 2011.

[100] A. K. Singh, M. Shafique, A. Kumar, and J. Henkel, "Mapping on multi/many-core systems: Survey of current and emerging trends," in *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–10, 2013.

[101] V. Nollet, T. Marescaux, P. Avasare, D. Verkest, and J.-Y. Mignolet, "Centralized run-time resource management in a network-on-chip containing reconfigurable hardware tiles," in *Design, Automation and Test in Europe*, pp. 234–239 Vol. 1, 2005.

[102] E. W. Briao, D. Barcelos, and F. R. Wagner, "Dynamic task allocation strategies in MPSoC for soft real-time applications," in *2008 Design, Automation and Test in Europe*, pp. 1386–1389, 2008.

[103] A. Raghavan, Y. Luo, A. Chandawalla, M. Papaefthymiou, K. P. Pipe, T. F. Wenisch, and M. M. K. Martin, "Computational sprinting," in *IEEE International Symposium on High-Performance Comp Architecture*, pp. 1–12, 2012.

[104] E. Rotem, A. Naveh, A. Ananthakrishnan, E. Weissmann, and D. Rajwan, "Power-management architecture of the Intel microarchitecture code-named Sandy Bridge," *IEEE Micro*, vol. 32, no. 2, pp. 20–27, 2012.

[105] E. Rotem, A. Naveh, D. Rajwan, A. Ananthakrishnan, and E. Weissmann, "Power management architecture of the 2nd generation intel® core microarchitecture, formerly codenamed sandy bridge," in *2011 IEEE Hot Chips 23 Symposium (HCS)*, pp. 1–33, 2011.

[106] D. Rodopoulos, F. Catthoor, and D. Soudris, "Tackling performance variability due to RAS mechanisms with PID-controlled DVFS," *IEEE Computer Architecture Letters*, vol. 14, no. 2, pp. 156–159, 2015.

[107] M. Noltsis, D. Rodopoulos, N. Zompakis, F. Catthoor, and D. Soudris, "Runtime slack creation for processor performance variability using system scenarios," *ACM Transactions on Design Automation of Electronic Systems*, vol. 23, dec 2017.

[108] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced cpu energy," in *Proceedings of the 1st USENIX Conference on Operating Systems Design and Implementation*, OSDI '94, (USA), USENIX Association, 1994.

[109] K. Govil, E. Chan, and H. Wasserman, "Comparing algorithm for dynamic speed-setting of a low-power cpu," in *Proceedings of the 1st Annual International Conference on Mobile Computing and Networking*, MobiCom '95, (New York, NY, USA), p. 13–25, Association for Computing Machinery, 1995.

[110] T. Pering, T. Burd, and R. Brodersen, "Voltage scheduling in the IpARM microprocessor system," in *ISLPED'00: Proceedings of the 2000 International Symposium on Low Power Electronics and Design*, pp. 96–101, 2000.

[111] T. Pering, T. Burd, and R. Brodersen, "The simulation and evaluation of dynamic voltage scaling algorithms," in *Proceedings. 1998 International Symposium on Low Power Electronics and Design*, pp. 76–81, 1998.

[112] P. Kumar and M. Srivastava, "Predictive strategies for low-power RTOS scheduling," in *Proceedings 2000 International Conference on Computer Design*, pp. 343–348, 2000.

[113] W. Yuan and K. Nahrstedt, "Energy-efficient soft real-time CPU scheduling for mobile multimedia systems," in *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, SOSP '03, (New York, NY, USA), p. 149–163, Association for Computing Machinery, 2003.

[114] H. Jung and M. Pedram, "Supervised learning based power management for multicore processors," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 9, pp. 1395–1408, 2010.

[115] S. J. Tarsa, A. P. Kumar, and H. T. Kung, "Workload prediction for adaptive power scaling using deep learning," in *2014 IEEE International Conference on IC Design & Technology*, pp. 1–5, 2014.

[116] S. Lee and T. Sakurai, "Run-time voltage hopping for low-power real-time systems," in *Proceedings 37th Design Automation Conference*, pp. 806–809, 2000.

[117] F. Gruian, "Hard real-time scheduling for low-energy using stochastic data and DVS processors," in *ISLPED'01: Proceedings of the 2001 International Symposium on Low Power Electronics and Designd*, pp. 46–51, 2001.

[118] W. Yuan and K. Nahrstedt, "Energy-efficient soft real-time CPU scheduling for mobile multimedia systems," in *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, SOSP '03, (New York, NY, USA), p. 149–163, Association for Computing Machinery, 2003.

[119] C. Scordino, E. Bini, *et al.*, "Optimal speed assignment for probabilistic execution times," in *2nd Workshop on Power-Aware Real-Time Computing (PARC'05)*, Citeseer, 2005.

[120] P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," in *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles*, SOSP '01, (New York, NY, USA), p. 89–102, Association for Computing Machinery, 2001.

[121] Y. Shin and K. Choi, "Power conscious fixed priority scheduling for hard real-time systems," in *Proceedings 1999 Design Automation Conference*, pp. 134–139, 1999.

[122] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez, "Dynamic and aggressive scheduling techniques for power-aware real-time systems," in *Proceedings 22nd IEEE Real-Time Systems Symposium (RTSS 2001)*, pp. 95–105, 2001.

[123] Y. Zhu and F. Mueller, "Feedback edf scheduling exploiting dynamic voltage scaling," in *Proceedings. RTAS 2004. 10th IEEE Real-Time and Embedded Technology and Applications Symposium, 2004.*, pp. 84–93, 2004.

[124] C. Im, H. Kim, and S. Ha, "Dynamic voltage scheduling technique for low-power multimedia applications using buffers," in *ISLPED'01: Proceedings of the 2001 International Symposium on Low Power Electronics and Design*, pp. 34–39, 2001.

[125] D. Shin and J. Kim, "Dynamic voltage scaling of periodic and aperiodic tasks in priority-driven systems," in *ASP-DAC 2004: Asia and South Pacific Design Automation Conference 2004 (IEEE Cat. No.04EX753)*, pp. 653–658, 2004.

[126] C. Scordino and G. Lipari, "A resource reservation algorithm for power-aware scheduling of periodic and aperiodic real-time tasks," *IEEE Transactions on Computers*, vol. 55, no. 12, pp. 1509–1522, 2006.

[127] C. Scordino, L. Abeni, and J. Lelli, "Energy-aware real-time scheduling in the Linux kernel," in *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, SAC '18, (New York, NY, USA), p. 601–608, Association for Computing Machinery, 2018.

[128] F. M. M. u. Islam and M. Lin, "Hybrid DVFS scheduling for real-time systems based on reinforcement learning," *IEEE Systems Journal*, vol. 11, no. 2, pp. 931–940, 2017.

[129] Q. Zhang, M. Lin, L. T. Yang, Z. Chen, and P. Li, "Energy-efficient scheduling for real-time systems based on deep Q-learning model," *IEEE Transactions on Sustainable Computing*, vol. 4, no. 1, pp. 132–141, 2019.

[130] G. Dhiman and T. S. Rosing, "System-level power management using online learning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 5, pp. 676–689, 2009.

[131] M. Bhatti, C. Belleudy, and M. Auguin, "Hybrid power management in real time embedded systems: An interplay of DVFS and DPM techniques," *Real-Time Systems*, vol. 47, pp. 143–162, Mar. 2011.

[132] R. Hapka, A. Christmann, and R. Ernst, "Controlling high-performance platform uncertainties with timing diversity," in *2022 IEEE 28th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pp. 212–219, 2022.

[133] S. Munaga and F. Catthoor, "Systematic design principles for cost-effective hard constraint management in dynamic nonlinear systems," *International Journal of Adaptive, Resilient and Autonomic Systems*, vol. 2, p. 18–45, Jan 2011.

[134] S. V. Gheorghita, M. Palkovic, J. Hamers, A. Vandecappelle, S. Mamagkakis, T. Basten, L. Eeckhout, H. Corporaal, F. Catthoor, F. Vandeputte, and K. D. Bosschere, "System-scenario-based design of dynamic embedded systems," *ACM Transactions on Design Automation of Electronic Systems*, vol. 14, jan 2009.

[135] F. Catthoor, T. Basten, N. Zompakis, M. Geilen, and P. G. Kjeldsberg, *System-Scenario-based Design Principles and Applications*. Dordrecht: Springer International Publishing, 2020.

[136] J.-Y. Lin, P. Weckx, S. Mishra, A. Spessot, and F. Catthoor, "Proactive run-time mitigation for time-critical applications using dynamic scenario methodology," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 616–621, 2022.

[137] J.-Y. Lin, P. Weckx, S. Mishra, A. Spessot, and F. Catthoor, "Multitimescale mitigation for performance variability improvement in time-critical systems," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 30, no. 11, pp. 1757–1769, 2022.

[138] T. Sato and I. Arita, "Execution latency reduction via variable latency pipeline and instruction reuse," in *Euro-Par 2001 Parallel Processing* (R. Sakellariou, J. Gurd, L. Freeman, and J. Keane, eds.), (Berlin, Heidelberg), pp. 428–438, Springer Berlin Heidelberg, 2001.

[139] L. Gwennap, "MIPS R10000 uses decoupled architecture," *Microprocessor Report*, vol. 8, no. 14, pp. 18–22, 1994.

[140] J. Balkind, T.-J. Chang, P. J. Jackson, G. Tziantzioulis, A. Li, F. Gao, A. Lavrov, G. Chirkov, J. Tu, M. Shahrad, and D. Wentzlaff, "Openpiton at 5: A nexus for open and agile hardware design," *IEEE Micro*, vol. 40, no. 4, pp. 22–31, 2020.

[141] Z. Hu, A. Buyuktosunoglu, V. Srinivasan, V. Zyuban, H. Jacobson, and P. Bose, "Microarchitectural techniques for power gating of execution units," in *Proceedings of the 2004 International Symposium on Low Power Electronics and Design (IEEE Cat. No.04TH8758)*, pp. 32–37, 2004.

[142] A. Waterman, K. Asanović, and J. Hauser, eds., *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture Document Version 20211203*. RISC-V International, 2021.

[143] D. Yakimets, M. G. Bardon, D. Jang, P. Schuddinck, Y. Sherazi, P. Weckx, K. Miyaguchi, B. Parvais, P. Raghavan, A. Spessot, D. Verkest, and A. Mocuta, "Power aware finfet and lateral nanosheet fet targeting for 3nm cmos technology," in *2017 IEEE International Electron Devices Meeting (IEDM)*, pp. 20.4.1–20.4.4, 2017.

[144] E. Hammari, F. Catthoor, P. G. Kjeldsberg, J. Huisken, K. Tsakalis, and L. Iasemidis, "Identifying data-dependent system scenarios in a dynamic embedded system," in *The 2012 International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA'12)*, pp. 70–76, 2012.

[145] Z. Ma, P. Marchal, D. P. Scarpazza, P. Yang, C. Wong, J. I. Gómez, S. Himpe, C. Ykman-Couvreur, and F. Catthoor, eds., *Systematic Methodology for Real-Time Cost-Effective Mapping of Dynamic Concurrent Task-Based Systems on Heterogenous Platforms*. Dordrecht: Springer Netherlands, 2007.

[146] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization*, pp. 3–14, 2001.

[147] B. Ranjbar, F. Klemme, P. R. Genssler, H. Amrouch, J. Jung, S. Dave, H. So, K. Lee, A. Shrivastava, J.-Y. Lin, P. Weckx, S. Mishra, F. Catthoor, D. Biswas, and A. Kumar, "Learning-oriented reliability improvement of computing systems from transistor to application level," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, April 2023.

[148] S. Das, D. Bull, and P. Whatmough, "Error-resilient design techniques for reliable and dependable computing," in *2014 IEEE International Reliability Physics Symposium*, pp. 3D.2.1–3D.2.6, 2014.

[149] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge, "Razor: a low-power pipeline based on circuit-level timing speculation," in *Proceedings. 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36.*, pp. 7–18, 2003.

[150] S. Das, C. Tokunaga, S. Pant, W.-H. Ma, S. Kalaiselvan, K. Lai, D. M. Bull, and D. T. Blaauw, "RazorII: In situ error detection and correction for PVT and SER tolerance," *IEEE Journal of Solid-State Circuits*, vol. 44, no. 1, pp. 32–48, 2009.

[151] D. Bull, S. Das, K. Shivashankar, G. S. Dasika, K. Flautner, and D. Blaauw, "A power-efficient 32 bit ARM processor using timing-error

detection and correction for transient-error tolerance and adaptation to PVT variation," *IEEE Journal of Solid-State Circuits*, vol. 46, no. 1, pp. 18–31, 2011.

[152] M. S. Gupta, K. K. Rangan, M. D. Smith, G.-Y. Wei, and D. Brooks, "DeCoR: A delayed commit and rollback mechanism for handling inductive noise in processors," in *2008 IEEE 14th International Symposium on High Performance Computer Architecture*, pp. 381–392, 2008.

[153] H. D. Dixit, S. Pendharkar, M. Beadon, C. Mason, T. Chakravarthy, B. Muthiah, and S. Sankar, "Silent data corruptions at scale," *CoRR*, vol. abs/2102.11245, 2021.

[154] P. H. Hochschild, P. Turner, J. C. Mogul, R. Govindaraju, P. Ranganathan, D. E. Culler, and A. Vahdat, "Cores that don't count," in *Proceedings of the Workshop on Hot Topics in Operating Systems*, HotOS '21, (New York, NY, USA), p. 9–16, Association for Computing Machinery, 2021.

[155] V. Huard, F. Cacho, F. Giner, M. Saliva, A. Benhassain, D. Patel, N. Torres, S. Naudet, A. Jain, and C. Parthasarathy, "Adaptive wearout management with in-situ aging monitors," in *2014 IEEE International Reliability Physics Symposium*, pp. 6B.4.1–6B.4.11, 2014.

[156] A. Benhassain, F. Cacho, V. Huard, S. Mhira, L. Anghel, C. Parthasarathy, A. Jain, and A. Sivadasan, "Robustness of timing in-situ monitors for AVS management," in *2016 IEEE International Reliability Physics Symposium (IRPS)*, pp. CR–4–1–CR–4–6, 2016.

[157] A. Benhassain, S. Mhira, F. Cacho, V. Huard, and L. Anghel, "In-situ slack monitors: taking up the challenge of on-die monitoring of variability and reliability," in *2016 1st IEEE International Verification and Security Workshop (IVSW)*, pp. 1–5, 2016.

[158] H. Amrouch, S. B. Ehsani, A. Gerstlauer, and J. Henkel, "On the efficiency of voltage overscaling under temperature and aging effects," *IEEE Transactions on Computers*, vol. 68, no. 11, pp. 1647–1662, 2019.

[159] AMD, "AMD eighth-generation processor architecture," tech. rep., AMD white paper, 2001.

[160] J. Mitchell, D. Henderson, and G. Ahrens, "IBM POWER 5 processor-based servers: A highly available design for buisness-critical applications," tech. rep., IBM white paper, 2005.

[161] P. Kongetira, K. Aingaran, and K. Olukotun, "Niagara: a 32-way multithreaded sparc processor," *IEEE Micro*, vol. 25, no. 2, pp. 21–29, 2005.

[162] M. May, M. Alles, and N. Wehn, "A case study in reliability-aware design: A resilient LDPC code decoder," in *2008 Design, Automation and Test in Europe*, pp. 456–461, 2008.

[163] S. Gupta, S. Feng, A. Ansari, J. Blome, and S. Mahlke, "StageNetSlice: A reconfigurable microarchitecture building block for resilient CMP systems," in *Proceedings of the 2008 International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, CASES '08, (New York, NY, USA), p. 1–10, Association for Computing Machinery, 2008.

[164] A. Ejlali, B. M. Al-Hashimi, and P. Eles, "A standby-sparing technique with low energy-overhead for fault-tolerant hard real-time systems," in *Proceedings of the 7th IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis*, CODES+ISSS '09, (New York, NY, USA), p. 193–202, Association for Computing Machinery, 2009.

[165] S.-S. Pyo, C.-H. Lee, G.-H. Kim, K.-M. Choi, Y.-H. Jun, and B.-S. Kong, "45nm low-power embedded pseudo-SRAM with ECC-based auto-adjusted self-refresh scheme," in *2009 IEEE International Symposium on Circuits and Systems*, pp. 2517–2520, 2009.

[166] P. Bannon, G. Venkataramanan, D. D. Sarma, and E. Talpes, "Computer and redundancy solution for the full self-driving computer," in *2019 IEEE Hot Chips 31 Symposium (HCS)*, pp. 1–22, 2019.

[167] P. Pop, V. Izosimov, P. Eles, and Z. Peng, "Design optimization of time- and cost-constrained fault-tolerant embedded systems with checkpointing and replication," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, p. 389–402, Mar 2009.

[168] F. Abate, L. Sterpone, and M. Violante, "A new mitigation approach for soft errors in embedded processors," in *2007 9th European Conference on Radiation and Its Effects on Components and Systems*, pp. 1–6, 2007.

[169] P. Eles, V. Izosimov, P. Pop, and Z. Peng, "Synthesis of fault-tolerant embedded systems," in *2008 Design, Automation and Test in Europe*, pp. 1117–1122, 2008.

[170] M. Vayrynen, V. Singh, and E. Larsson, "Fault-tolerant average execution time optimization for general-purpose multi-processor system-on-chips," in *2009 Design, Automation & Test in Europe Conference & Exhibition*, pp. 484–489, 2009.

[171] R. Hyman, K. Bhattacharya, and N. Ranganathan, "A strategy for soft error reduction in multi core designs," in *2009 IEEE International Symposium on Circuits and Systems*, pp. 2217–2220, 2009.

[172] J. Henkel, L. Bauer, J. Becker, O. Bringmann, U. Brinkschulte, S. Chakraborty, M. Engel, R. Ernst, H. Härtig, L. Hedrich, A. Herkersdorf, R. Kapitza, D. Lohmann, P. Marwedel, M. Platzner, W. Rosenstiel, U. Schlichtmann, O. Spinczyk, M. Tahoori, J. Teich, N. When, and H.-J. Wunderlich, "Design and architectures for dependable embedded systems," in *2011 Proceedings of the Ninth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pp. 69–78, 2011.

[173] J. Henkel, L. Bauer, N. Dutt, P. Gupta, S. Nassif, M. Shafique, M. Tahoori, and N. Wehn, "Reliable on-chip systems in the nano-era: Lessons learnt and future trends," in *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–10, 2013.

[174] M. Vayrynen, V. Singh, and E. Larsson, "Fault-tolerant average execution time optimization for general-purpose multi-processor system-on-chips," in *2009 Design, Automation & Test in Europe Conference & Exhibition*, pp. 484–489, 2009.

[175] J. Henkel, L. Bauer, H. Zhang, S. Rehman, and M. Shafique, "Multi-layer dependability: From microarchitecture to application level," in *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2014.

[176] K. Lee, A. Shrivastava, M. Kim, N. Dutt, and N. Venkatasubramanian, "Mitigating the impact of hardware defects on multimedia applications: A cross-layer approach," in *Proceedings of the 16th ACM International Conference on Multimedia*, MM '08, (New York, NY, USA), p. 319–328, Association for Computing Machinery, 2008.

[177] H. Lee, H. Shin, and S.-L. Min, "Worst case timing requirement of real-time tasks with time redundancy," in *Proceedings Sixth International Conference on Real-Time Computing Systems and Applications. RTCSA'99 (Cat. No.PR00306)*, pp. 410–414, 1999.

[178] R. Melhem, D. Mosse, and E. Elnozahy, "The interplay of power management and fault recovery in real-time systems," *IEEE Transactions on Computers*, vol. 53, no. 2, pp. 217–231, 2004.

[179] D. Zhu and H. Aydin, "Reliability-aware energy management for periodic real-time tasks," *IEEE Transactions on Computers*, vol. 58, no. 10, pp. 1382–1397, 2009.

[180] P. Eles, V. Izosimov, P. Pop, and Z. Peng, "Synthesis of fault-tolerant embedded systems," in *2008 Design, Automation and Test in Europe*, pp. 1117–1122, 2008.

[181] M. Salehi, M. Khavari Tavana, S. Rehman, M. Shafique, A. Ejlali, and J. Henkel, "Two-state checkpointing for energy-efficient fault tolerance in hard real-time systems," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 7, pp. 2426–2437, 2016.

[182] M. Ansari, S. Safari, H. Khdr, P. Gohari-Nazari, J. Henkel, A. Ejlali, and S. Hessabi, "Power-aware checkpointing for multicore embedded systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 12, pp. 4410–4424, 2022.

[183] L. Huang, F. Yuan, and Q. Xu, "Lifetime reliability-aware task allocation and scheduling for MPSoC platforms," in *2009 Design, Automation & Test in Europe Conference & Exhibition*, pp. 51–56, 2009.

[184] S. Mishra, P. Weckx, J. Y. Lin, B. Kaczer, D. Linten, A. Spessot, and F. Catthoor, "Fast & accurate methodology for aging incorporation in circuits using adaptive waveform splitting (AWS)," in *2020 IEEE International Reliability Physics Symposium (IRPS)*, pp. 1–5, 2020.

[185] S. Walla, "Primetime® advanced OCV technology," tech. rep., Synopsys white paper, 2009.

[186] A. Mutlu, J. Le, R. Molina, and M. Celik, "A parametric approach for handling local variation effects in timing analysis," in *2009 46th ACM/IEEE Design Automation Conference*, pp. 126–129, 2009.

[187] V. M. van Santen, H. Amrouch, and J. Henkel, "New worst-case timing for standard cells under aging effects," *IEEE Transactions on Device and Materials Reliability*, vol. 19, no. 1, pp. 149–158, 2019.

[188] Q. Cao, "Carbon nanotube transistor technology for More-Moore scaling," *Nano Research*, vol. 14, pp. 3051–3069, Sep 2021.

[189] J. Zhang, A. Lin, N. Patil, H. Wei, L. Wei, H.-S. P. Wong, and S. Mitra, "Carbon nanotube robust digital VLSI," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 4, pp. 453–471, 2012.

[190] Q. Zhang, J.-Q. Huang, M.-Q. Zhao, W.-Z. Qian, and F. Wei, "Carbon nanotube mass production: Principles and processes," *ChemSusChem*, vol. 4, no. 7, pp. 864–889, 2011.

[191] S. J. Tans, A. R. M. Verschueren, and C. Dekker, "Room-temperature transistor based on a single carbon nanotube," *Nature*, vol. 393, pp. 49–52, May 1998.

[192] G. Hills, C. Lau, A. Wright, S. Fuller, M. D. Bishop, T. Srimani, P. Kanhaiya, R. Ho, A. Amer, Y. Stein, D. Murphy, Arvind, A. Chandrakasan, and M. M. Shulaker, "Modern microprocessor built from complementary carbon nanotube transistors," *Nature*, vol. 572, pp. 595–602, Aug 2019.

[193] S. G. Noyce, J. L. Doherty, Z. Cheng, H. Han, S. Bowen, and A. D. Franklin, "Electronic stability of carbon nanotube transistors under long-term bias stress," *Nano Letters*, vol. 19, no. 3, pp. 1460–1466, 2019. PMID: 30720283.

[194] R. Saito, G. F. Dresselhaus, and M. S. Dresselhaus, *Physical Properties of Carbon Nanotubes*. Imperial College Press, 1998.

[195] P. Weckx, J. Ryckaert, E. D. Litta, D. Yakimets, P. Matagne, P. Schuddinck, D. Jang, B. Chehab, R. Baert, M. Gupta, Y. Oniki, L.-A. Ragnarsson, N. Horiguchi, A. Spessot, and D. Verkest, "Novel forksheet device architecture as ultimate logic scaling device towards 2nm," in *2019 IEEE International Electron Devices Meeting (IEDM)*, pp. 36.5.1–36.5.4, 2019.

[196] B. Chehab, O. Zografos, E. D. Litta, Z. Ahmed, P. Schuddinck, D. Jang, G. Hellings, A. Spessot, P. Weckx, and J. Ryckaert, "Two-level MOL and VHV routing style to enable extreme height scaling beyond 2nm technology node," in *2021 IEEE International Interconnect Technology Conference (IITC)*, pp. 1–3, 2021.

[197] M. Pelgrom, A. Duinmaijer, and A. Welbers, "Matching properties of MOS transistors," *IEEE Journal of Solid-State Circuits*, vol. 24, no. 5, pp. 1433–1439, 1989.

[198] S.-Y. Wu, C. Chang, M. Chiang, C. Lin, J. Liaw, J. Cheng, J. Yeh, H. Chen, S. Chang, K. Lai, M. Liang, K. Pan, J. Chen, V. Chang, T. Luo, X. Wang, Y. Mor, C. Lin, S. Wang, M. Hsieh, C. Chen, B. Wu, C. Lin, C. Liang, C. Tsao, C. Li, C. Chen, C. Hsieh, H. Liu, P. Chen, C. Chen, R. Chen, Y. Yeo, C. Chui, W. Chang, T. Lee, K. Huang, H. Lin, K. Chen, M. Tsai, K. Chen, X. Chen, Y. Cheng, C. Wang, W. Shue, Y. Ku, S. M. Jang, M. Cao, L. Lu, and T. Chang, "A 3nm CMOS FinFlex™ platform technology with enhanced power efficiency and performance for mobile soc and high performance computing applications," in *2022 International Electron Devices Meeting (IEDM)*, pp. 27.5.1–27.5.4, 2022.

[199] C. Gilardi, B. Chehab, G. Sisto, P. Schuddinck, Z. Ahmed, O. Zografos, Q. Lin, G. Hellings, J. Ryckaert, H.-S. Wong, and S. Mitra, "Extended scale length theory targeting low-dimensional FETs for carbon nanotube

FET digital logic design-technology co-optimization," in *2021 IEEE International Electron Devices Meeting (IEDM)*, pp. 27.3.1–27.3.4, 2021.

[200] H. T. Mair, G. Gammie, A. Wang, R. Lagerquist, C. Chung, S. Gururajarao, P. Kao, A. Rajagopalan, A. Saha, A. Jain, E. Wang, S. Ouyang, H. Wen, A. Thippana, H. Chen, S. Rahman, M. Chau, A. Varma, B. Flachs, M. Peng, A. Tsai, V. Lin, U. Fu, W. Kuo, L.-K. Yong, C. Peng, L. Shieh, J. Wu, and U. Ko, "A 20nm 2.5GHz ultra-low-power tri-cluster CPU subsystem with adaptive power allocation for optimal mobile SoC performance," in *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, pp. 76–77, 2016.

[201] J. Pyo, Y. Shin, H.-J. Lee, S.-i. Bae, M.-s. Kim, K. Kim, K. Shin, Y. Kwon, H. Oh, J. Lim, D.-w. Lee, J. Lee, I. Hong, K. Chae, H.-H. Lee, S.-W. Lee, S. Song, C.-H. Kim, J.-S. Park, H. Kim, S. Yun, U.-R. Cho, J. C. Son, and S. Park, "20nm high-K metal-gate heterogeneous 64b quad-core CPUs and hexa-core GPU for high-performance and energy-efficient mobile application processor," in *2015 IEEE International Solid-State Circuits Conference - (ISSCC) Digest of Technical Papers*, pp. 1–3, 2015.

[202] S. Dighe, S. R. Vangal, P. Aseron, S. Kumar, T. Jacob, K. A. Bowman, J. Howard, J. Tschanz, V. Erraguntla, N. Borkar, V. K. De, and S. Borkar, "Within-die variation-aware dynamic-voltage-frequency-scaling with optimal core allocation and thread hopping for the 80-core teraFLOPS processor," *IEEE Journal of Solid-State Circuits*, vol. 46, no. 1, pp. 184–193, 2011.

List of Publications

International Journal Papers

- **J.-Y. Lin**, P. Weckx, S. Mishra, A. Spessot, and F. Catthoor, "Multitimescale mitigation for performance variability improvement in time-critical systems," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 30, no. 11, pp. 1757–1769, 2022.

International Conference Proceedings

- B. Ranjbar, F. Klemme, P. R. Genssler, H. Amrouch, J. Jung, S. Dave, H. So K. Lee, A. Shrivastava, **J.-Y. Lin**, P. Weckx, S. Mishra, F. Catthoor, D. Biswas, and A. Kumar, "Learning-Oriented Reliability Improvement of Computing Systems From Transistor to Application Level," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, April 2023.
- **J.-Y. Lin**, P. Weckx, S. Mishra, A. Spessot, and F. Catthoor, "Proactive run-time mitigation for time-critical applications using dynamic scenario methodology," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 616–621, 2022.
- S. Mishra, P. Weckx, O. Zografos, **J. Y. Lin**, A. Spessot, and F. Catthoor, "Overhead reduction with optimal margining using a reliability aware design paradigm," in 2021 IEEE International Reliability Physics Symposium (IRPS), pp. 1–7, 2021.
- S. Mishra, P. Weckx, **J. Y. Lin**, B. Kaczer, D. Linten, A. Spessot, and F. Catthoor, "Fast & accurate methodology for aging incorporation in circuits using adaptive waveform splitting (AWS)," in *2020 IEEE International Reliability Physics Symposium (IRPS)*, pp. 1–5, 2020.

FACULTY OF ENGINEERING SCIENCE
DEPARTMENT OF ELECTRICAL ENGINEERING
Celestijnenlaan 200A box 2402
B-3001 Leuven