Yahya H. Yassin

Techniques for Scenario Prediction and Switching in System Scenario Based Designs

Dual PhD agreement between NTNU and KU Leuven

Thesis for the Degree of
Philosophiae Doctor (NTNU)
Doctor of Engineering Science (KU Leuven)

Trondheim, April 2018

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Electronic Systems

KU Leuven
Faculty of Engineering Science
Department of Electrical Engineering (ESAT)

NTNU
Norwegian University of Science and Technology

KU LEUVEN

NTNU and KU Leuven

Thesis for the Degree of
Philosophiae Doctor (NTNU)
Doctor of Engineering Science (KU Leuven)

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Electronic Systems

KU Leuven
Faculty of Engineering Science
Department of Electrical Engineering (ESAT)

© Yahya H. Yassin

ISBN 978-82-326-2970-1 (printed ver.)
ISBN 978-82-326-2971-8 (electronic ver.)
ISSN 1503-8181

Doctoral theses at NTNU, 2018:90

Doctoral theses at KU Leuven

Printed by NTNU Grafisk senter

Abstract

Many modern applications exhibit dynamic behavior, which can be exploited for reduced energy consumption. In this dissertation we contribute to the development of a two-phase combined design-time/run-time methodology known as the system scenario design methodology. This methodology identifies different run-time situations and clusters similar behaviors into system scenarios. It is implemented in our framework for system scenario based designs, which dynamically tunes the hardware to match the application behavior. We develop techniques for scenario prediction and switching for data variable dependent applications, and achieve significant energy reductions when applying these techniques on an extracted control structure of a video codec widely used in hand-held devices today.

Two different microcontroller boards are used to validate our techniques with the use of dynamic voltage and frequency scaling (DVFS). In addition, we developed a novel DVFS controller extension for the GEM5 computer architecture (CA) simulator in order to better evaluate the tuning of platform resources to the dynamic behavior of modern applications. The GEM5 extension is used together with our novel energy model, which is calibrated with measurements from real hardware.

Finally, we develop a novel data dependent neuro-bio application that significantly reduces the amount of data to be transmitted from high-density electrode arrays. These arrays are used to study the behavior of neural cells and networks through extracellular monitoring of the action potentials in animals' brains in vivo. This application also serves as a complex, realistic example of a data dependent application, which is applicable for our system scenario techniques.

Preface

This doctoral thesis was submitted to the Norwegian University of Science and Technology (NTNU) in partial fulfillment of the requirements for the degree Philosophiae Doctor (PhD). The thesis is a part of a dual PhD program between NTNU and the Catholic University of Leuven (KULeuven), in cooperation with Interuniversity Microelectronics Center (IMEC) in Leuven, Belgium. The work herein was performed under the supervision of Professor Per Gunnar Kjeldsberg and Professor Francky Catthoor, at the Department of Electronic Systems, NTNU and the Department of Electrical Engineering, KULeuven.

Part of the research work is done locally in Leuven, Belgium, as required by the dual PhD agreement. The research stay at IMEC consisted of a number of visits which lasted for approximately 6 months in total. As one major result from these research visits, our data reduction application presented in Appendix F was developed. This application turned out to be more promising and interesting than initially expected, so to allow appropriate effort and time to be spent, this topic was significantly extended in the research plan, and this has resulted in a strong research contribution on its own. KULeuven, NTNU and IMEC later filed a patent application for the results of this work, which is currently patent pending. Due to the lack of time in the remainder of the contract period, the scenario implementation in this application was not researched and is subject to future work. At the end of Chapter 3.5 we discuss and provide ideas on how we can apply our techniques for scenario prediction and switching to this neural science application.

Acknowledgements

First of all I would like to thank my main supervisor, Prof. Per Gunnar Kjeldsberg, for giving me the opportunity to work with interesting research topics, which further extended my knowledge and skills in digital systems design. His advice and support was invaluable throughout the PhD work. I also want to thank my co-supervisor Prof. Francky Catthoor for his support and advise through the long and challenging journey that eventually led to the results presented in this thesis.

I also extend my gratitude to my co-supervisor, associative Prof. Magnus Jahre, for his insight in the GEM5 simulator and all the guidance he provided me with during the development of the GEM5 simulator. I thank Prof. Snorre Aunet for his help with the power consumption formulas and for his time spent in double checking the properties of my simplified static power formulas. A special thanks goes to Prof. Andrew Perkis, who helped me with understanding the conditions to maintain good perceptual video quality and frame rate in video codes. I am also grateful for Prof. Lasse Natvig's help and review of our prefetching report which led to the ideas developed for the proposed framework for system scenarios.

I want to express my thanks to the Body Area Networks (BAN), Biomedical Circuits (MEDIC) and Neuroelectronics Research Flanders (NERF, Kloosterman Lab and Giugliano Lab) groups, at IMEC, Belgium, for their support and expertise in their fields, which helped me achieve the results that led to an important contribution in the neural science domain. In particular I want to thank the team leader at the Kloosterman Lab, Dr. Fabian Kloosterman, for giving me access to his team through Dr. Jyh-Jang Sun, and Dr. João Couto. His team provided me with all the knowledge and databases I needed to fully understand the challenges in their domain.

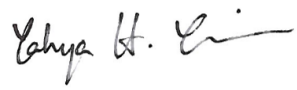
A special thanks goes to the team leader at the MEDIC group, Dr. Nick Van Helleputte, who gave me full insight in what was possible with neural probes in the future. Dr. Srinjoy Mitra and Dr. Firat Yazicioglu provided me with their knowledge in previous work related to data-reduction approaches and helped me understand the limitations of today's neural probes. I want to thank Dr. Karthick Parashar from the Circuits and Systems for ICT, Mobile group at IMEC for aiding me with a power efficient software square root algorithm for use in our neural science application. Management Assistant Marion Hegemann, made sure my research stay at IMEC was a pleasure and I thank her for the administrative support she provided

me with.

In general I want to thank all the members of the MEDIC group at IMEC for their social discussions at lunch time, and Dr. Nicola Marrocco and PhD student Bogdan C. Raducanu for their interesting coffee breaks involving suggestions for future business opportunities.

I want to thank all my research colleagues at the computing, signal processing, and the circuit and radio systems groups at the Norwegian University of Science and Technology (NTNU) for making my time at NTNU pleasant. I thank Dr. Iason Filippopoulos and PhD student Elena Hammari for helping me understand the system scenario concept and sharing their work with me. I also want to thank Dr. Milica Orlandic, Dr. Hourieh Atarzadeh, and PhD students Bahram Najafiuchevler and Ali A. Vatanjou for their motivational support in frustrating times, and our interesting discussions about all social and work related things during my research stay at NTNU.

A special thanks goes to my parents, Hussain and Hadeel, for their love and support. They always encourage me to work hard and never give up. My parents grew up dreaming about graduating with a PhD degree in medicine, but they were stripped of this opportunity because of unfortunate circumstances in their lives, such as war, many years ago. I therefore dedicate this thesis to them. Last, but not least, I want to thank my wife, Deema, for coming into my life during my PhD journey, and for her tremendous love and support during difficult times. She continues to give me her unconditional love and support when I need it the most, and she always makes me believe that I can do great things.



Yahya H. Yassin
September 2017

CONTENTS

List of Tables **xvi**

List of Figures **xix**

List of Abbreviations **xxi**

1 Introduction **1**
- 1.1 Embedded systems, energy consumption and dynamic application behavior . . . 1
- 1.2 Dynamic data variable dependencies in applications . . . 4
- 1.3 Problem statement . . . 5
- 1.4 Related work for energy efficient design . . . 6
- 1.5 Thesis contributions . . . 7
- 1.6 Thesis outline . . . 8

2 Background **9**
- 2.1 System scenario design methodology . . . 9
  - 2.1.1 Use-case vs. system scenarios . . . 9
  - 2.1.2 Methodology overview . . . 11
- 2.2 The prediction and switching step of the system scenario design methodology . . . 13

2.2.1 Prediction . . . 13
2.2.2 Switching . . . 16
2.3 Related work on identification and prediction of RTS parameters in system scenario applications . . . 16
2.4 Switching mechanism used in a run-time resource management framework . . . 18
2.5 Dynamic voltage and frequency scaling (DVFS) . . . 19
2.5.1 DVFS at different implementation levels . . . 19
2.5.2 Race-To-Halt (RTH) vs DVFS . . . 20
2.5.3 RTH in combination with DVFS . . . 21
2.6 Design and evaluation platforms . . . 21
2.6.1 EFM32 evaluation board . . . 21
2.6.2 SAM4L evaluation board . . . 22
2.6.3 GEM5 CA simulator . . . 23
2.7 Representative example of an application with dynamic behavior . . . 25
2.8 Example of a data dependency problem: Extracellular monitoring of the action potentials in animal's brains . . . 26
2.8.1 The study of neural cells and networks . . . 26
2.8.2 General goal of data reduction approaches . . . 28

3 Proposed framework, platform implementations and techniques 31
3.1 Motivating the need for data variable system scenarios . . . 32
3.2 Framework for system scenarios (FSS) . . . 33
3.3 System scenario framework implementation . . . 34
3.3.1 EFM32 . . . 35
3.3.2 SAM4L . . . 35
3.3.3 GEM5 . . . 38
3.4 Data variable based system scenario methodology . . . 40
3.4.1 Methodology Overview . . . 40
3.4.2 Design-time profiling based on data variables . . . 43
3.4.3 Design-time system scenario identification based on data variables . . . 44
3.4.4 Run-time system scenario detection and switching based on data variables . . . 46
3.4.5 Improving the energy overhead and exploiting PMM . . . 48
3.5 Data dependent application development for system scenarios 48
3.5.1 Automatic data reduction of wireless read-out in high-density electrode arrays . . . 49

3.5.2 RTS parameters applicable for our system scenario techniques 50

4 Research Results and Contributions **53**

4.1 Paper 1: Adaptive Run-Time Prefetch Manager using Sequential On-Access and DCPT Prefetching 53
4.1.1 Abstract 54
4.1.2 Main Contributions 54
4.1.3 Roles of the Authors 54

4.2 Paper 2: System Scenario Framework evaluation on EFM32 using the H264/AVC encoder control structure 55
4.2.1 Abstract 55
4.2.2 Main Contributions 55
4.2.3 Roles of the Authors 56

4.3 Paper 3: Dynamic hardware management of the H264/AVC encoder control structure using a framework for system scenarios 56
4.3.1 Abstract 56
4.3.2 Main Contributions 57
4.3.3 Roles of the Authors 57

4.4 Paper 4: Techniques for dynamic hardware management of streaming media applications using a framework for system scenarios 58
4.4.1 Abstract 58
4.4.2 Main Contributions 59
4.4.3 Roles of the Authors 59

4.5 Paper 5: Fast and accurate DVFS implementation in GEM5 using system call emulation mode 59
4.5.1 Abstract 60
4.5.2 Main Contributions 60
4.5.3 Roles of the Authors 61

4.6 Paper 6: Algorithm/architecture co-optimisation technique for automatic data reduction of wireless read-out in high-density electrode arrays 61
4.6.1 Abstract 61
4.6.2 Main Contributions 62
4.6.3 Roles of the Authors 62

5 Conclusions and Future Work **63**
5.1 Conclusions . . . 63
5.2 Future Work . . . 64

Bibliography **67**

A Adaptive run-time prefetch manager using sequential on-access and DCPT prefetching **83**
Abstract . . . 85
A.1 Introduction . . . 87
A.2 Related work . . . 90
A.3 M5 simulation environment, platform architecture and benchmark programs . . . 91
A.4 Prefetcher concepts and APM extension . . . 95
A.4.1 Simple APM based on cache hit rate . . . 96
A.4.2 Memory access pattern matching based APM (MAP-APM) . . . 98
A.4.3 Dynamic MAP-APM (DMAP-APM) . . . 100
A.5 Performance results . . . 102
A.6 Power and energy consumption estimation . . . 106
A.6.1 Dynamic power and energy evaluation . . . 106
A.6.2 Static power and energy evaluation . . . 108
A.7 Conclusions . . . 109

B System scenario framework evaluation on EFM32 using the H264/AVC encoder control structure **113**
Abstract . . . 115
B.1 Introduction . . . 117
B.2 FSS evaluation using partial H264/AVC . . . 119
B.3 Results . . . 123
B.4 Conclusion . . . 124

C Dynamic hardware management of the H264/AVC encoder control structure using a framework for system scenarios **127**
Abstract . . . 129
C.1 Introduction . . . 131
C.2 Definitions . . . 133
C.2.1 The system scenario based design methodology . . . 133
C.2.2 Control variables vs data variables . . . 133
C.3 Related work . . . 134

C.4 Motivational example . . . 136
C.5 Experimental setup . . . 137
C.6 Scenario identification methodology . . . 140
C.7 Prediction and switching mechanism . . . 142
C.8 Experimental results . . . 143
C.9 Conclusion . . . 146

D Techniques for dynamic hardware management of streaming media applications using a framework for system scenarios 149
Abstract . . . 151
D.1 Introduction . . . 153
D.2 Terminology . . . 155
D.2.1 The system scenario based design methodology . . . 155
D.2.2 Control variables vs data variables . . . 156
D.3 Related work . . . 157
D.4 Motivational example . . . 159
D.5 Experimental setup . . . 161
D.6 Scenario identification methodology . . . 164
D.7 Prediction and switching mechanism . . . 166
D.7.1 Input data variable dependency prediction . . . 168
D.7.2 Improving the scenario energy overhead . . . 169
D.7.3 Introducing sleep mode management with system scenarios . . . 171
D.8 Experimental results . . . 172
D.8.1 Energy reductions compared to a RTH approach without system scenarios . . . 173
D.8.2 Exploiting available sleep modes based on remaining available slack time . . . 175
D.8.3 Introducing more DVFS states into our scenario mechanisms . . . 177
D.9 Conclusion . . . 178

E Fast and accurate DVFS implementation in GEM5 using system call emulation mode 181
Abstract . . . 183
E.1 Introduction . . . 185
E.2 Related work . . . 187
E.3 Contribution I: GEM5 DVFS in SE mode . . . 188
E.3.1 GEM5 kernel level modifications . . . 191
E.3.2 GEM5 user level modifications . . . 193

E.3.3 Usage of DVFS in the application . . . 196
E.3.4 Limitations of our DVFS mechanisms . . . 196
E.4 Contribution II: Energy model . . . 197
E.4.1 Estimation of the power consumption . . . 197
E.4.2 Combining GEM5 statistics with SAM4L power measurements . . . 199
E.4.3 GEM5 energy modeling script . . . 200
E.4.4 Offset error corrections . . . 201
E.5 Experimental setup . . . 202
E.6 Results . . . 203
E.6.1 Correction of systematic offset errors . . . 203
E.6.2 Comparing energy reduction techniques in GEM5 . . . 204
E.7 Discussion of results . . . 205
E.8 Conclusion . . . 207

F Algorithm/architecture co-optimisation technique for automatic data reduction of wireless read-out in high-density electrode arrays 209
Abstract . . . 211
F.1 Introduction . . . 213
F.2 Related work . . . 216
F.3 General goal of data reduction approaches . . . 219
F.4 Wired and wireless data transfer limitations . . . 220
F.5 Temporal and spatial spike data extraction . . . 222
F.5.1 Phase 1: Temporal spike detection . . . 222
F.5.2 Phase 2: Spatial spike data analysis . . . 225
F.5.3 Phase 3: Spike data extraction . . . 226
F.6 Estimated HW requirements . . . 226
F.7 Experimental setup . . . 229
F.8 Quality metrics . . . 230
F.9 Data reduction and quality results . . . 231
F.9.1 Reduction of transmitted samples . . . 231
F.9.2 Quality results . . . 232
F.9.3 Discussion . . . 236
F.10 Conclusion . . . 238

LIST OF TABLES

3.1 Scenario assignment based on run time 44

A.1 M5 system parameters. 92
A.2 Interface functions. 93
A.3 Our selection of the programs from the SPEC CPU2000 benchmark suite, and their category description. 93
A.4 Each element in the structure, representing one row in the DCPT table. 95
A.5 The APM threshold parameters, and the access interval between measurements that resulted in the best speedup. 97
A.6 Variations of the sequential prefetching scheme. 98
A.7 MAP-APM lookup table entry. 98

B.1 Scenario assignment based on run time 123

C.1 DVFS switching times between PS0 and PS1. 139
C.2 Scenario assignment based on run time 141

D.1 Run/Sleep modes in our SAM4L-board from Atmel 162
D.2 DVFS switching times between PS0 and PS1. 164
D.3 Scenario assignment based on run time 166

E.1 GEM5 modifications 190
E.2 Comparison to state-of-the-art DVFS implementations in GEM5 191
E.3 Variables related to the total power based on measurements . 199
E.4 GEM5 energy measurement deviation from SAM4L measurements 203
E.5 GEM5 offset correction parameters 204
E.6 Energy consumption of our H264 application 205

F.1 Supported number of electrodes using 40 nm cell-based memories 228

LIST OF FIGURES

1.1 TCM scheduling 3
1.2 Control vs data variable dependency 5

2.1 A use-case vs system scenario design trajectory for embedded systems. Source: [39] (modified). 10
2.2 System Scenario methodology 11
2.3 Three types of scenario predictors 15
2.4 EFM32 board from Silicon Labs 22
2.5 SAM4L board from Atmel 23
2.6 myDAQ board from National Instruments 24
2.7 The GEM5 architecture used as a starting point for the research in this thesis 25
2.8 Probe with multiple electrodes 27
2.9 Spike sorting 29

3.1 Overview of how the papers in this dissertation are connected together 32
3.2 Proposed FSS framework 34
3.3 SAM4L setup with NI myDAQ 36
3.4 SAM4L measurements with our own developed Energy Monitor 37

3.5 Flow chart for identifying dynamic data variable dependencies and classifying system scenarios 41
3.6 Energy consumption relative to no system scenarios 47
3.7 3-phase spike data extraction 49
3.8 Potential spatial selection improvements when using chFactor as an algorithm knob 52

A.1 The adaptive prefetch manager solution in a customized low power general purpose system. 88
A.2 An entry in the DCPT table. 91
A.3 The simple prefetch manager state machine. 96
A.4 Prefetch distance, degree and stride. 98
A.5 Memory access pattern matching based APM flow. 99
A.6 DMAP-APM flow. 101
A.7 Speedup per benchmark program relative to a no prefetching solution. 103
A.8 Coverage per benchmark program. 103
A.9 IPC per benchmark program. 104
A.10 Accuracy per benchmark program. 104
A.11 Harmonic mean of identified vs issued prefetches. 105
A.12 Dynamic run-time power consumption relative to no prefetching. 106
A.13 Dynamic energy consumption relative to no prefetching. . . . 107
A.14 Runtime prefetching scheme selection for our simple APM. . 108
A.15 Runtime prefetching scheme selection for our MAP-APM. . . 109
A.16 Runtime prefetching scheme selection for our DMAP-APM. . 109
A.17 Static energy consumption consumed by the DCPT memory. 110

B.1 Framework single-core and multi-core platforms 118
B.2 Realization of the FSS framework on EFM32 121
B.3 Power consumption of the system at available frequencies and the FSS AMU and PAM power overhead. 124

C.1 Control vs data variable dependency 134
C.2 Available bandwidth measurement and moving average over WIFI, LTE and WCDMA 136
C.3 Energy consumption relative to no system scenarios 144

D.1 System Scenario methodology 155
D.2 Control vs data variable dependency 156

D.3 Available bandwidth measurement and moving average over WIFI, LTE and WCDMA . . . 160
D.4 Contents of rts_data for different input frame sizes . . . 169
D.5 Changes to the code after simple loop transforms . . . 170
D.6 Energy consumption relative to no system scenarios . . . 174
D.7 Potential improvements in energy consumption with slower wakeup times. . . . 176

E.1 GEM5 DVFS simulator flow . . . 189
E.3 Energy reduction with 6 DVFS modes . . . 205

F.1 Probe with multiple electrodes . . . 214
F.2 Summary of current spike detection solutions . . . 216
F.3 Spike sorting . . . 219
F.4 Block scheme of our envisioned platform . . . 222
F.5 3-phase spike data extraction . . . 223
F.6 Storage estimate . . . 227
F.7 Definitions for quality measurement . . . 230
F.8 Data reduction of 0.1 second data averaged over various recording lengths . . . 232
F.9 Total data reduction (%) of a 55 minute recording . . . 232
F.10 Sample-based quality results . . . 233
F.11 Spike-based quality results . . . 234
F.12 Sample-based quality results with different ThFactor values . 235
F.13 Spike-based quality results with different ThFactor values . . 236

LIST OF ABBREVIATIONS

AMU Application Monitoring Unit

APM Adaptive Prefetch Manager

ASIC Application Specific Integrated Circuit

ASM Assembler

CA Computer Architecture

CPU Central Processing Unit

DFS Dynamic Frequency Scaling

DSP Digital Signal Processor

DVFS Dynamic Voltage and Frequency Scaling

FPGA Field-Programmable-Gate-Array

FSS Framework for System Scenarios

GALS Globally-Asynchronous Locally-Synchronous

GPU Graphics Processing Unit

HW Hardware

IC Integrated Circuit

IEEE Institute of Electrical and Electronics Engineers

IMEC Interuniversity Microelectronics Centre

KULeuven Catholic University of Leuven

LUT Lookup Table

MCD Multiple-Clock Domain

MDD Multi-valued Decision Diagram

MPSoC Multi-Processor System on Chip

NI National Instruments

NOP No OPeration instruction

NRE Non-Recurring Engineering cost

NTNU Norwegian University of Science and Technology

OS Operating System

PAM Platform Adaption Manager

PCA Principal Component Analysis

PMM Power Mode Management

QoE Quality-of-Experience

RRM Run-time Resource Management

RTH Race-To-Halt

RTI Race-To-Idle

RTS Run Time Situation

SE System call Emulation

SRAM Static Random Access Memory

SW Software

TCM Task Concurrency Management

UML Unified Modeling Language

VLSI Very Large Scale Integration

WCET Worst-Case Execution Time

XML eXtensible Markup Language

CHAPTER

1

INTRODUCTION

1.1 Embedded systems, energy consumption and dynamic application behavior

In recent years, energy efficiency and power consumption have become major design constraints. Hand held devices are becoming ever more popular, and the applications embedded in these devices require more advanced functionality. This often results in increased power consumption.

According to Ma et al. [65], until a decade ago programmers wrote highly efficient embedded software using low-level programming languages such as C or assembly when the performance constraints are tight. Performance and memory requirements force the use of application-dependent, proprietary operating systems, or no operating systems at all. This approach was feasible when embedded software was relatively simple. With today's increased complexity of embedded-system applications, this primitive approach has become a bottleneck of the design productivity.

An effective methodology to partition and re-organize complex applications with concurrent tasks, and map them onto an embedded multi-core system platform is needed. Conventional embedded system design techniques are not able to meet the requirements of today's demanding soft-

ware functionality. They rely on manual partitioning and scheduling of software components. In addition, there is an increasing gap between the semiconductor processing capability and integrated circuit (IC) design capability, due to the faster growth of processing capability (58% per year) against the growth of design capability (21% per year) [31]. The result is increased non-recurring engineering (NRE) and IC design cost. Gheorghita et al. [39] explain that state-of-the-art design methodologies try to cope with this challenge by identifying several most used cases and dealing with them separately, in order to reduce the increased complexity of the system.

The inherent dynamism, i.e., the system property that enables the system to adapt at run time under the influence of environments, is the new challenge in embedded systems [65]. For example, if the application varies significantly at run-time, and only requires high performance for a fraction of its lifetime, then this observation can be exploited to tune the underlying platform accordingly to save energy. Conventional hardware/software co-design approaches focus mainly on static analysis where no dynamism is present. When dynamism is introduced, then only the worst case behavior is used in many platform implementations to perform the mapping. Or in the best case, several individual "use case scenarios" are identified. These scenarios allow the designer to optimize the worst-case behavior of these use cases separately. But even then, the worst-case analysis at design-time leads to a significant waste in resources and energy. A gap exists between the high-level specification and the detailed implementation of embedded software design.

Ma et al. [65] presents Marchal et al.'s [67], [66] proposal of a systematic design methodology for high-performance embedded systems, known as the Task Concurrency Management (TCM) methodology. The objective of this methodology is to be a systematic approach to conduct the design space exploration as well as to synthesize the final implementation automatically. As explained by Ma et al. [65], the basic idea of the TCM methodology is a well balanced combined design-time/run-time two-phase approach. This is illustrated in Figure 1.1.

At design-time, the input applications are scheduled on the target multiprocessor platform, where the TCM scheduler performs a multi-objective scheduling on objectives such as throughput, latency, and energy per task. The design-time scheduling result is represented by a set of Pareto-optimal schedules.

The run-time scheduler then exploits the Pareto-optimal schedules for the particular application that is running. With little overhead, the run-

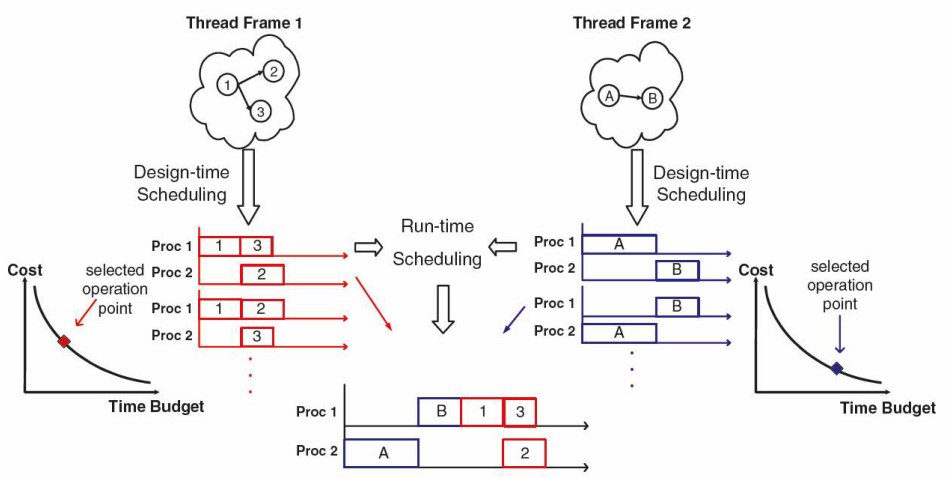


Figure 1.1: A two-phased scheduling method. Source: [65]

time scheduler can choose the schedule for each instance of applications based on their run-time situation. The chosen schedule from each running application instance are then interleaved to generate a final schedule. In order to tune each task mapping and to enable the Pareto trade-offs, platform and application knobs have to be available. One of the most well-used and powerful knob set is the so called dynamic voltage and frequency scaling (DVFS) technique.

This methodology, which focuses on task scheduling, is fully complementary to the use of the system scenario approach. System scenarios group similar system behaviors from a multidimensional cost perspective, such that the system can be configured to exploit these cost similarities.

The concept of system scenarios differ from use-case scenarios, which focus on the application's functional and timing behavior, as well as the application's interaction with the users and the environment. System scenarios on the other hand, are derived from a combination of the application's behavior and the application mapping on the system platform [39]. In other words, a system scenario is defined as the application behavior based on a combination of run-time situations not visible to the user. For example, a loop can have it's total number of iterations assigned at run-time by a data variable, and this variable could vary dynamically based on other run-time situations occurring at run-time. Therefore, this loop may have a dramatic impact on the application's execution time. These scenarios are used to reduce the system cost by exploiting the knowledge of the application's

run-time behavior to make better design decisions at design-time, and to exploit the dynamic behavior at run-time.

The complementarity between the TCM and system scenario approaches is demonstrated in [3] for the control variable system scenario approach. The system scenario design methodology can also be used to efficiently tune system resources according to the applications needs, e.g., by utilizing DVFS, dynamically turning on and off idle memory banks, or by remapping tasks and data into partitions implemented as hardware accelerators.

Gheorghita et al. [39] show through different case studies that this methodology obtains substantial energy reductions. They define five steps for the design-time and run-time phases. These steps include the identification, prediction, exploitation, switching and calibration of system scenarios. This methodology exploits the dynamic variation in an application, where system knobs, such as frequency, voltage, and low power modes, are tuned dynamically at run-time in order to gain significant reductions in energy consumption. Gheorghita et al. focuses on the so-called control variable variant of the system scenario approach. Another variant focusing on data variables has been introduced more recently, where Hammari et al. [45] extends the system scenario approach to be able to handle data variables. It enables the handling of applications with dynamic parameters that exhibit a large value range (e.g. in a nested while loop). However, only the identification step of the data variable system scenarios approach has been researched [45]. We describe the system scenario design methodology in more detail in Chapter 2.

The work in this dissertation focuses on further improving and extending the system scenario prediction and switching mechanisms in the data variable system scenario design methodology. The goal is to provide techniques for scenario prediction and switching when the application's control structure is data variable dependent.

1.2 Dynamic data variable dependencies in applications

As motivated in Section 1.1, real-time embedded systems have become more complex in the past decade, which have resulted in applications with a more dynamic behavior. This section will highlight the challenge of monitoring application parameters which are data variable dependent.

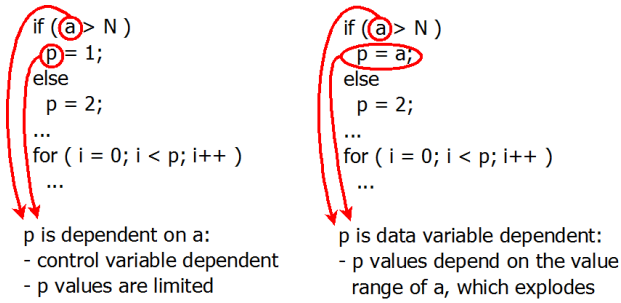


Figure 1.2: Control vs data variable dependency

In order to better understand the challenge of data variable dependencies, Figure 1.2 illustrates simplified code snippets illustrating the difference between a control variable and data variable dependency.

The code snippet on the left in Figure 1.2 illustrates a control variable dependency. The variable p in this case has a limited value range, and it will be relatively easy to monitor the application behavior if p is later used, for example, as a loop parameter.

The code snippet on the right shows a data variable dependent situation, where the value range of p equals the value range of a. If a is an input parameter, and p is used as a loop parameter in succeeding computations, then the loop size varies with the input parameter value range. Assuming the input parameter is a 16-bit integer value, then the value range of a will be equal to 2^{16}. This value range would simply produce too much run-time overhead for the scenario prediction and switching mechanisms if these data variable dependent parameters are used directly to classify system scenarios with the initial approach proposed in [39].

1.3 Problem statement

The aim of this research is to extend the existing system scenario based design methodology and implement a working prototype in which scenario based designs can be demonstrated. Several platforms will be used to demonstrate our approach, and a publicly available application with a dynamic data variable dependent behavior in the multi-media domain, i.e., the H264 encoder from the Mediabench II website [70], will be used as a representative and realistic base for our approach on handling the prediction

of nested data dependencies. Another dynamic data variable dependent application will be developed in the neuro-bio domain for significant reduction in the transmit energy of future wireless read-out of high density electrode arrays. High density electrode arrays are used by neural scientists to study the behavior of neural cells and networks through extracellular monitoring of the action potentials in animal's brains *in vivo*. The purpose of this application is to serve as an example of how complex dynamic data variable dependent behavior in an application can be managed by our approach. No Application Specific Integrated Circuit (ASIC) is developed, but instead the focus lies on instruction-set programmable target platforms. Most of the implementations are based on existing evaluation kits, MATLAB and the GEM5 computer architecture (CA) simulator.

The focus of this doctoral thesis will be on how to predict system scenarios (the prediction step) and how to switch from one system scenario to another (the switching step) in the scenario based design methodology presented in Chapter 2.

1.4 Related work for energy efficient design

Commonly used techniques exist for controlling system configurations, such as Dynamic Voltage and Frequency Scaling (DVFS) and Dynamic Frequency Scaling (DFS). The DFS and DVFS techniques are proven to be a highly effective method of achieving low power consumption while meeting the performance requirements [22], [49].

Another energy saving approach focus on using low power modes, called Power Mode Management (PMM) or dynamic power mode management [72], [33], [68], [96]. This approach is widely used in embedded systems today. PMM techniques restricts the application by forcing the processor to halt in low power modes, which leads to Race-to-Halt (RTH) solutions [10], i.e., by speeding up the execution of the application workload before powering down to a lower power consuming state.

Unconventional cores [72], such as Digital Signal Processors (DSPs) [104], Graphics Processing Units (GPUs) [74], and Field-Programmable Gate-Arrays (FPGAs) [32] are used when general-purpose processors lack the performance requirements to efficiently run advanced applications on embedded platforms.

Other techniques focus on the microarchitecture, i.e., by leveraging the application properties or variation in workload to dynamically reconfigure

a component of the system to save energy [72], [114], [102], [109].

According to Spiliopoulos et. al [99], the use of DVFS techniques and PMM is one of the most important techniques for managing the dynamic power consumption of a system. Therefore, CA simulators need configurable voltage scaling functionality in order to further develop energy saving techniques.

The techniques described above are usually applied between application runs or when powering down the system to preserve energy. The system scenario based design methodology enables the designer to apply these techniques while the application is running, by tuning the system knobs according to the predicted scenario. Limited research has been done for most steps in the data variable variant of the system scenario design methodology, and most of the research that is done has mainly focused on how to identify system scenarios (the identification step).

More detailed description of previous work can be found in Chapter 2 and in the individual papers presented in the appendixes of this thesis.

1.5 Thesis contributions

In this thesis, we focus on further developing the system scenario design methodology for data variable dependent applications. A generalized and flexible framework for system scenarios is proposed that can be used by researchers in scenario based designs, both for the control and data variable variants. This framework takes into account the limitations observed from the research results in related work. It is an extension of the RRM framework presented in [127], making it capable of predicting and switching scenarios within an application instead of switching between applications.

Techniques are then developed for data variable dependent prediction and switching in system scenario based designs, which are validated on a H264 encoder control structure application on multiple HW development kits. These techniques can be used regardless of the type of identification method chosen for the identification step. It is based on the constraint observations pointed out by previous work which is required for effective and lightweight prediction and switching mechanisms. Our techniques are applied with different configurations together with the DFS, DVFS and RTH techniques.

We further contribute with the development of a novel, fast and accurate DVFS extension for the GEM5 CA simulator. We also develop a novel

architecture independent energy model which is calibrated based on power measurements from real HW. This model is used with our GEM5 DVFS extension.

Finally, we develop an advanced neural science application that reduces the data transferred from electrodes used to read out, for instance, neural activity in animals' brains *in vivo*. This application also serves as an example of a data variable dependent application, which is applicable for the system scenario design methodology. We provide indications of how to apply our techniques for scenario prediction and switching to this application in future work, in order to dynamically reduce the platform energy consumption and hence the transmit energy of wireless read-out in high density electrode arrays.

1.6 Thesis outline

The remainder of this thesis is organized as follows: Chapter 2 contains background information which gives the reader enough knowledge to understand our work. The research process and the details of our work are presented in Chapter 3. Each paper on which the thesis is based is described in Chapter 4 with a breakdown of the roles of each author. Chapter 5 concludes the thesis with a summary of contributions and directions for future work. The appendixes hold each paper in chronological order. These papers are reproduced faithfully with regard to the published text, but have been reformatted to increase readability.

CHAPTER

2

BACKGROUND

2.1 System scenario design methodology

This section describes the basic concepts behind the system scenario design methodology and its main steps.

2.1.1 Use-case vs. system scenarios

The concept of system scenarios has been presented in a wide range of applications [37], [65], [38]. It differs from another type of scenarios, called use-case scenarios, that is used in several published works. Use-case scenario approaches generate different scenarios based on a user's behavior. The use of a future system is described by these scenarios in an early phase of the development process. In case of human-computer interaction, use-case scenarios appear like narrative descriptions of envisioned usage episodes. In object oriented software engineering, use-case scenarios are used as a unified modeling language (UML) use-case diagram. From functional and timing point of view, this UML diagram enumerates all possible user actions and the system reactions required to meet a proposed system function. In the

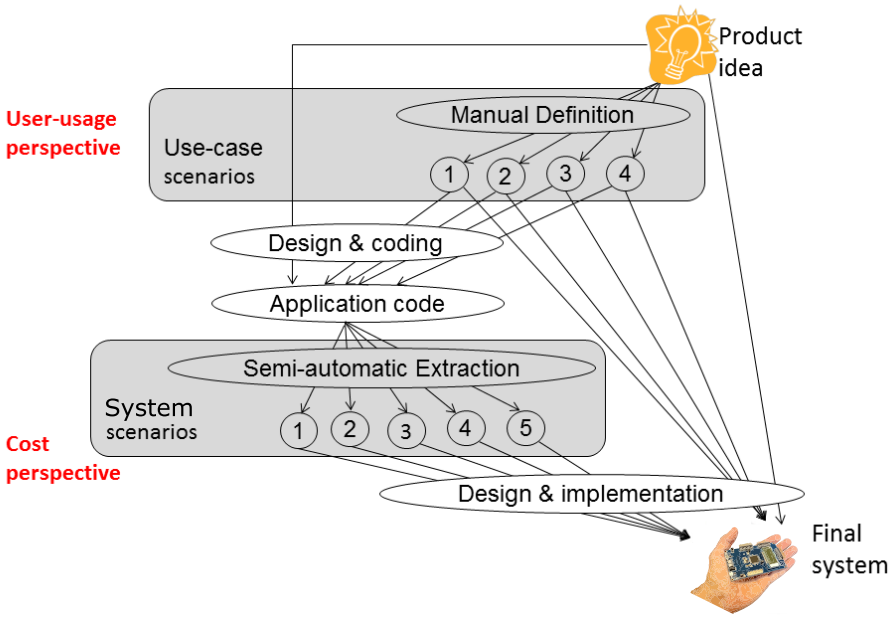


Figure 2.1: A use-case vs system scenario design trajectory for embedded systems. Source: [39] (modified).

embedded systems domain, use-case scenarios are used in both hardware [50] [83] and software design [25]. They focus on the application's functional and timing behaviors and on its interaction with the users and the environment. These scenarios are used as an input during system design for user-centered design approaches. Use-case scenarios do not focus on the resources required by a system to meet its constraints.

The work in this thesis is based on the concept of system scenarios, which are derived from the behavior of the application. In order to make better design decisions, system scenarios are used to reduce the system cost by exploiting all available information about what can happen at run-time. While use-case scenarios classify the application's behavior based on the different ways it can be used, system scenarios classify it from the resource usage perspective, based on the cost trade-off aspects during the mapping to the platform.

Figure 2.1 depicts a design trajectory using use-case and system scenarios, and is based on the same figure in [39]. First, the product idea defines

the product's functionality manually through use-case scenarios. These scenarios characterize the system from a user perspective, and are used as an input to the design of an embedded system. In order to optimize the design of the system, the detection and usage of system scenarios augments this trajectory (the bottom gray box in the figure). Once the application is coded, its scenarios related to resource utilization are extracted in an automatic way, and they are considered for the decisions made during the following phases of the system design. Hence, the run-time behavior of the application is classified into several system scenarios, where the cost of the operation modes within a system scenario is always fairly similar. For each individual system scenario, more specific and aggressive design decisions can be made.

2.1.2 Methodology overview

Gheorghita et al. [39] uses the system scenario methodology for handling the dynamic nature of modern embedded systems. In a number of case studies this methodology obtained substantial energy reductions. They define five design steps for the design-time and run-time phases as shown in Figure 2.2. These steps are briefly described in the following.

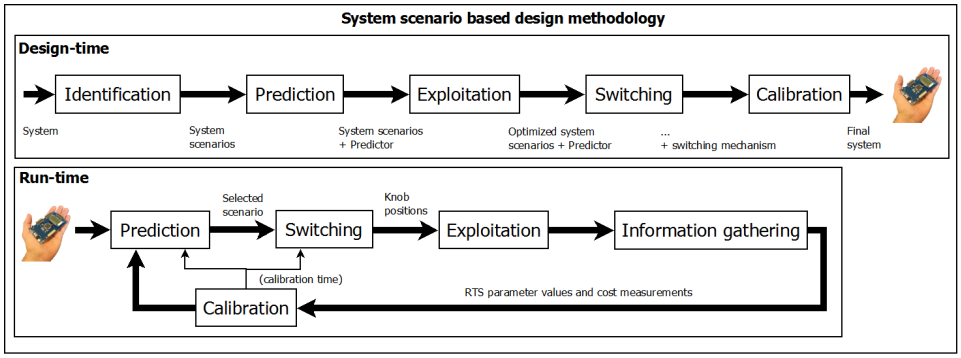


Figure 2.2: The system scenario based design methodology overview. Modified from [39].

- **Identification of the scenario set**

  In this step, the different run-time situations (RTS) are identified, usually through profiling. The run-time situations are different behaviors

of the application which occur throughout the application lifetime. Similar RTSs are grouped together and clustered into scenarios. A RTS is normally defined by a number of key parameters (i.e., RTS parameters) in the application. These parameters determine the application's behavior through the application's control structure. For example, if a parameter is used to define the size of a loop in consecutive parts of the application, then it may have an impact on the application behavior if this parameter changes throughout the application lifetime. All overhead costs introduced by the following steps should be considered in this step as well. Hammari et al. [45] extends the identification step to handle data variable dependencies with polyhedral techniques. Filippopoulos et al. [29] identifies data variable dependent system scenarios by analyzing memory access pattern variations and the allocated memory size during execution.

- **Prediction of scenarios**

  At design time, the run-time prediction algorithm is selected, based on the output from the identification step. When the application is running at run-time, the prediction mechanism determines the upcoming scenario and whether or not to switch from one system scenario to another based on the actual RTS parameter values. It is not a trivial task since the prediction incurs a certain run-time overhead. In addition, the number of parameters and the number of scenarios may be considerable, so a simple lookup in a list of scenarios may not be feasible. The scenario set may therefore be refined based on the prediction overhead.

- **Exploitation of the scenario set**

  This step is essentially based on optimizations applied without the scenario approach. A scenario approach can be put on top of existing optimizations to achieve better optimization from a global perspective at design-time, i.e., by using code compaction techniques, and reuse common parts in different scenarios. In general, each scenario is optimized individually in this step. At run-time this step is the actual execution of the scenario.

- **Switching from one scenario to another**

  When a scenario switch is predicted by the prediction mechanism, the system changes from one set of run-time platform configuration parameters (system knobs, such as DVFS) to another. This implies

some system overhead, and it might not always be a good idea to switch from one scenario to another if the system overhead is too large, e.g., when migrating a task from one processor to another. The switching mechanism to be used at run-time is selected at design-time. This mechanism is triggered by the prediction mechanisms. How to implement the switch is decided, and the scenario set is refined, taking into account the switching overhead.

- **Calibration**

  Calibration support is optionally added at a last step to collect information about the actual values of the parameters at run-time, as well as the quality of the resulting system (i.e., deadline misses). At design-time, the information gathering mechanism is designed and added to the application. Calibration is done infrequently due to the high overhead cost. The main purpose of the calibration step is to optimize the scenario prediction and exploitation steps based on observations at run-time.

2.2 The prediction and switching step of the system scenario design methodology

The focus of this dissertation is to improve the prediction and switching steps described in the previous section. In this section, a more detailed description is given for these two steps in order to understand the contributions presented in this thesis in later chapters. The prediction and switching step of the system scenario design methodology described in this section is a summary of the description by Gheorghita et al. [39].

2.2.1 Prediction

At design-time, the predictor is based on the information about the RTS parameters collected during the identification step. The resulting predictor mainly bases its decisions on the values of the RTS parameters. It has to add limited decision overhead in the final system. If the predictor fails to predict a scenario based on some unconsidered values of the RTS parameters, then it basically switches to a backup scenario. The backup scenario is based

on the worst-case situation, but projected to the actual parameter ranges which have been assigned to the backup scenario partition.

If all possible RTS parameter values can be covered through the identification step, then the prediction accuracy would be 100%. However, the number of RTSs might be too large to efficiently consider all cases. Two approaches which target such situations have been discussed in [39].

The first is based on reducing or changing the set of considered parameters. The idea in this case is to identify the set of predictable scenarios that gives the best energy reduction.

In the second approach, a probabilistic function is used. The prediction point, i.e., the place where the prediction function is used in the application, may be moved to an earlier point in time. The function is augmented with a mechanism which selects the scenario with highest probability from the possible set of scenarios predicted by the function. Using this probabilistic approach, the miss-prediction might increase in terms of over-prediction (scenario with a larger cost is selected) and under-prediction (scenario with a smaller cost is selected). Over-prediction leads to a less cost effective system, and under-prediction might lead to an increase of missed deadlines, reducing the quality of the system.

In general, [39] classifies three types of predictors, as illustrated in Figure 2.3:

- Centralized predictors
- Distributed predictors with exclusive points
- Distributed predictors with refinement points

Centralized predictors

This type of predictor is centralized in the application, and it is inserted in a common location in the application for all scenarios (i.e., all RTS parameter values in the system are known). In Figure 2.3, it is introduced in the main loop, after the read part.

Distributed predictors with exclusive points

In this distributed predictor, an identical prediction function is introduced multiple times in the application code. The function must be placed in locations where the required RTS parameter values are known. If no common locations for all scenarios exists, this approach is feasible.

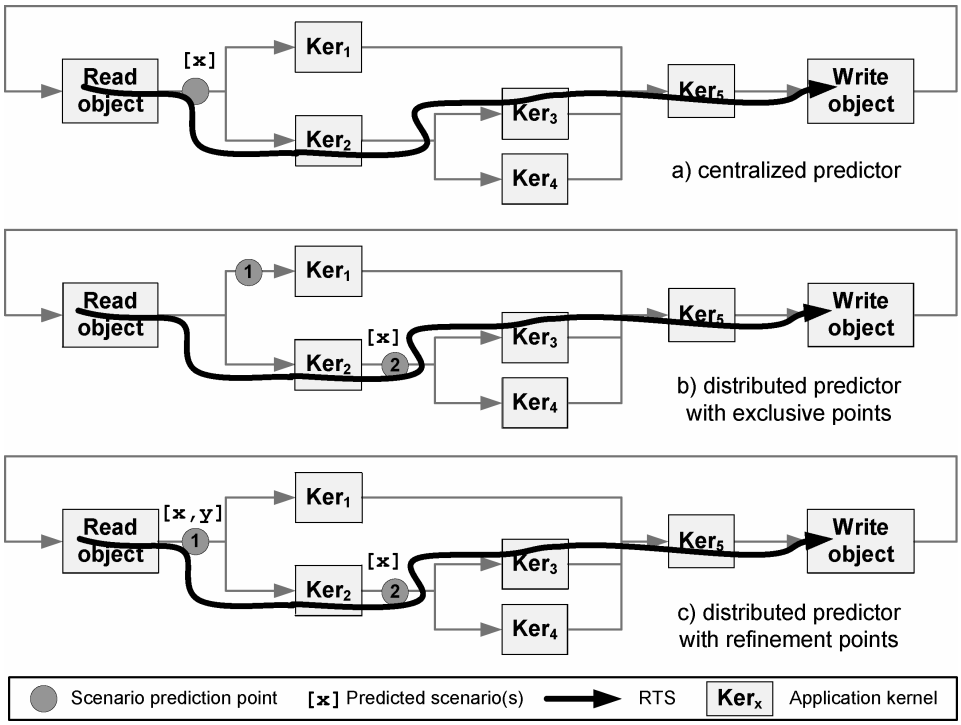


Figure 2.3: Three types of scenario predictors. Source: [39]

Distributed predictors with refinement points

Another version of the distributed predictor use refinement points to predict scenarios more accurately in a hierarchical manner. This approach might improve the efficiency of the optimizations as earlier switching might be done. However, a trade-off has to be done since too many switches increases the system overhead. In Figure 2.3, the first predictor selects a set of scenarios (x and y). This selection is refined in the second predictor to choose only scenario x.

Code size and an average run-time evaluation cost are the overheads introduced when using a prediction function. [39] suggests that the design time prediction step may cover a further clustering of scenarios (considering the prediction overhead and when the scenario may be predicted), further pruning of RTS parameters if possible, clustering of previously unassigned RTSs into scenarios, and defining and placing the prediction mechanism into the application. A trade-off between prediction accuracy versus overhead must be made with respect to the final system cost and quality.

2.2.2 Switching

The switching mechanisms are triggered by the prediction mechanisms. In parallel with deriving the switching mechanism at design-time, the set of scenarios, as well as the predictor may need to be adapted. The cost of each scenario, how often a switch appears at run-time and the overhead cost of the switch has to be taken into account in the adaption process.

The overhead cost of switching affects both the final system cost (e.g., more energy consumption) and its run-time properties (e.g., more deadline misses). It is therefore important to consider the lifetime of the new scenario. If the new scenario only lasts for a short period relative to the switching time, then we might not benefit from a scenario switch. The switching overhead is therefore exploitation dependent, i.e., it uses the scenario cost versus overhead cost, together with the information about how often a switch appears at run-time. It is important for the system to avoid spending most of the system's running time on switching between scenarios.

An example of the switching operation which is frequently employed is the DVFS technique which we describe in Section 2.5. It is important to compare the minimum time the system stays in a scenario with the time overhead of the switch. In some cases, a separate source code might be stored for each scenario. The time overhead may in this case be larger than changing some stored parameters for voltage and frequency. It will depend on the platform the application is mapped to.

2.3 Related work on identification and prediction of RTS parameters in system scenario applications

Agrawal [3] explore application partitioning and assignment of control variables in real-life multi-mode streaming applications together with platform multi-processor system on chip (MPSoC) architecture instantiation. Through case studies, Agrawal's use of the control variable variant of the system scenario methodology results in better than worst-case energy efficiency for platform instantiation to support multi-mode applications.

Gheorghita et al. [40], [38] introduce a method for automatic discovery of scenarios that incorporate correlations between different parts of applications. Only application parameters with large impact on the execution

time are considered, and by using scenarios, they improve the estimated worst case execution time (WCET). The work is later extended, where a methodology and a tool that can automatically detect the most important variables of an application is created, and used to dynamically select and predict scenarios with respect to the time budget (for soft real-time systems). This method is based on the control variables that have the highest impact on the run-time cost of the system. These variables are referred to as RTS parameters, and the methodology assumes that each parameter has a limited set of possible values.

Hammari et al. [45] concludes that the methodology suggested by Gheorghita et al. [40], [38] is infeasible for run-time prediction when the application contains many-valued parameters, because the size of their multi valued decision diagram (MDD) explodes. This is the case, i.e., when control variables in the application are data-dependent. Therefore, Hammari et al. [45] propose a method based on the projection of scenarios onto a RTS parameter space.

The projection of scenarios produces regions that characterize the system scenarios in terms of RTS parameter values. Each region can then be described as a polyhedron, and the run-time scenario prediction mechanism can check which polyhedron contains the RTS parameter values for the next RTS. The methodology presented by Hammari et al. [45] is useful when the application contains arbitrary large domains, given that the number of regions stays relatively low.

The initial algorithm produces convex scenario projections in the RTS parameter space. Hammari et al. [45] also discuss the need for concave scenario projections in some situations. There are situations where, e.g., the system scenarios overlap, and a projection of the related RTS parameters results in considerable reduction of run-time gain. A concave scenario projection will reduce the scenario overlap, and may improve the run-time gain. Algorithms processing concave polyhedra are complex. Therefore, Hammari et al. [45] propose a possible solution by splitting the concave projection into a set of convex polyhedra at design-time, and apply convex hull algorithms.

The prediction mechanism is highly dependent on how the identification step is performed, as indicated by previous work [40], [38], [45]. The authors in previous work conclude that the number of variables used in the prediction should be low. The same result can be observed by other approaches for identification, e.g., the method of using loop-trip count profiling [82]. The number of variations of each variable should also be limited in order

to guarantee an effective light weight prediction mechanism with acceptable system overhead cost during run-time.

2.4 Switching mechanism used in a run-time resource management framework

Ykman-Couvreur et al. [127] present a Run-time Resource Management (RRM) framework for embedded heterogeneous multi-core platforms. This framework allows dynamic adaption to change the application context, and transparent optimization of the platform resource usage. This RRM framework is targeting the support of various applications running concurrently, and which can start and stop at any time. Each application may have different constraints, given by the user or external environment.

In order to achieve high efficiency under changing environment conditions, this RRM must be able to dynamically adapt to a changing context. In this environment, quality-of-experience (QoE) requirements and platform resources are scaled dynamically by adjusting clock frequencies and voltages. Hence, the RRM is able to efficiently control the energy/power consumption and heat dissipation of the platform.

The framework presented by Ykman-Couvreur et al. [127] is targeted to the system scenario methodology. At design-time, a set of Pareto optimal application configurations under given QoE requirements and optimization goals are derived by an automated design-space exploration. Switching between these predefined configurations at run-time are handled by the RRM dynamically.

The limitation of this framework is that the total energy available has to be fixed and known, as well as the total size of input sequences. This information is used to calculate the maximum user value in terms of maximum allowed energy per input sequence. If the application changes some constraints dynamically during run-time, or if the input size is not known (e.g., in applications where the constraints are based on dynamically changing data-dependent variables), some adaptions have to be made in order to control the added dynamism. In general, the ideas this framework is based on serve as a good base and starting point for a more enhanced framework development where efficient prediction mechanisms are embedded.

2.5 Dynamic voltage and frequency scaling (DVFS)

The general goal of the Dynamic Voltage and Frequency Scaling (DVFS) techniques is to exploit the idle time of the system [27]. The voltage and frequency of the system is tuned in order to exploit the idle time and consume less power with lower voltage and frequency settings. Equation 2.1 shows the formula for the dynamic energy consumption (E_{Dyn}), which is the power consumption (P_{Dyn}) over time (derived from [27]).

$$E_{\text{Dyn}} = P_{\text{Dyn}} \cdot t = \alpha C V^2 f t \tag{2.1}$$

In Equation 2.1, α is the activity factor, C is the load capacitance, V is the voltage, f is the running clock frequency, and t is the time in seconds. Kaxiras et al. [27] explains that the use of the DVFS technique could lead to a quadratic or even cubic reduction of the dynamic energy consumption based on Equation 2.1. Note that reducing the frequency f only reduces the power consumption, while one has to reduce the voltage V in order to reduce the energy consumption. There is a dependency between the voltage reduction and the frequency's upper limit. If the voltage is reduced, then the upper limit of the frequency is also reduced.

2.5.1 DVFS at different implementation levels

According to Kaxiras et al. [27], the idleness of a system can be exploited at different levels: system level, program level and hardware level.

At system level, the idle time seen by the operating system (OS) is eliminated by scheduling (oracle) algorithms or OS-controlled power management mechanisms.

Program level DVFS focus more on the instruction level of an application. Idleness is reduced by either making offline compiler analysis manually or online through supported tools. Profile assisted approaches include the use of DVFS instructions and look at loop nests, call sites, call procedures, statement sequences or the entire program. Alternatively, multiple-clock domains (MCD) and Globally-asynchronous locally-synchronous (GALS) techniques [76] are used. GALS techniques divide the processor core into synchronous islands each of which is then interconnected asynchronously but with added circuitry to avoid metastability. Independent DVFS for

each clock domain within a processor can then be done dynamically or statically.

Hardware-level DVFS focus on failsafe HW, where the voltage and frequency is lowered until timing faults occur [27].

Our DVFS scope in this dissertation is mainly on program level DVFS management with the use of the techniques for scenario prediction and switching of data variable dependent scenarios at the application level.

2.5.2 Race-To-Halt (RTH) vs DVFS

Focusing only on reducing the power consumption does not necessarily result in energy efficient designs. As an alternative, techniques such as race-to-halt (RTH) [10] can be used to speed up the execution of an application, thus increasing instantaneous power consumption, in order to finish early and achieve longer idle times. RTH is an extension of a very commonly used (worst case) technique called Race-To-Idle (RTI), where the power consumption during idle times is significantly less than the power consumed when tasks are processed. With RTH the static power consumed during idle times is reduced significantly due to the utilization of extremely low power states. In other words, RTI focus on increasing the idle time of the system, while RTH focuses on turning the system off when it is in idle. RTH techniques thus reduces the total energy consumption of the system and is considered state-of-the-art for CPU bound workloads when the arrival time of the next set of tasks to be processed is unknown. According to Awan et al. [10], DVFS techniques are on the other hand more suitable for memory bound workloads, and for real time systems with deadlines and long idle times.

Many embedded applications are data dominated in the multimedia and telecommunication domain [16]. Catthoor et al. [16] concludes, through experiments, that a very large part of the power consumption is due to memory bound workloads, i.e., data storage and data transfer. Similar conclusions are made by the Compilers Creating Custom Processors (CCCP) Research Group at the University of Michigan[1].

[1] http://cccp.eecs.umich.edu/research/poweraware.php

2.5.3 RTH in combination with DVFS

Many researchers have optimized energy efficiency in embedded systems by proposing improved dynamic scheduling algorithms, such as Leakage Control Earliest-Deadline-First (LC-EDF) and Leakage Control Dual Priority (LC-DP) for reducing leakage current in hard real-time systems [61], 3-competitive offline and constant competitive ratio online algorithms for power saving while considering shutdown in combination with DVFS [51], and a combination of DVFS and shutdown to minimize the overall energy consumption based on the latest arrival time of jobs [79]. Most of these approaches produce significant run-time analysis overhead. Awan et al. [10], propose a solution with significantly less run-time overhead. They combine an online and offline algorithm that selects the best feasible sleep mode based on the available time slack for a given workload.

2.6 Design and evaluation platforms

Our target domain is embedded systems such as mobile phone platforms where energy efficiency is the main constraint. Flora et al. [30] highlight the main hardware characteristics of such a system, containing relatively weak CPUs and limited memory. Their survey show that good architectures are even more important for mobile phones than for desktops.

In this section, we present two ARM Cortex M evaluation boards which allow changing the CPU voltage and/or frequency while the CPU is processing workloads. These evaluation boards were available for our research project, and we used them to evaluate our techniques for system scenario prediction and switching presented in Chapter 3. A general overview of the GEM5 Computer Architecture (CA) simulator is also presented in this section.

2.6.1 EFM32 evaluation board

The EFM32 (EFM32GG-STK3700) board from Silicon Labs [97] contains a microcontroller with an ARM Cortex-M3 CPU core, and the microcontroller allows DFS at run-time through clock division. A frequency switch takes one system clock cycle when the system is set up properly, and is triggered

by a write to dedicated registers. Furthermore, this kit includes an advanced energy monitoring unit which enables us to monitor the systems power and energy consumption while the application is running. This microcontroller does not allow voltage control while the CPU is active, except when entering into a low power mode where the CPU remains inactive. Figure 2.4 depicts the EFM32 board used in our research.

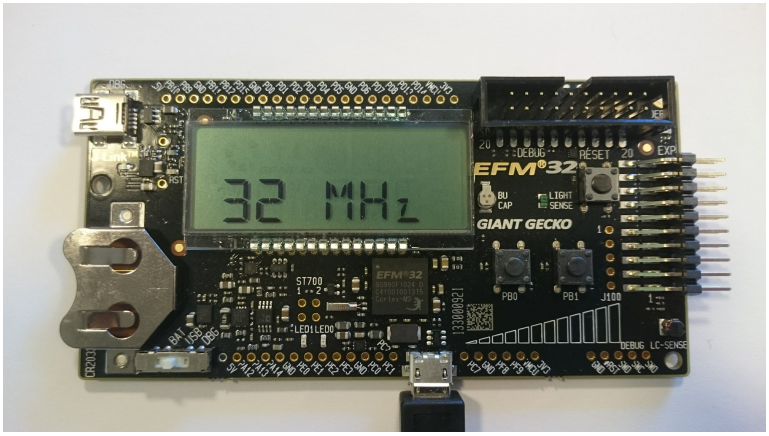


Figure 2.4: EFM32 board from Silicon Labs.

2.6.2 SAM4L evaluation board

The SAM4L board from Atmel [8] is a representative example of a mobile platform because it contains an energy friendly CPU with limited on-chip SRAM. One advantage of the SAM4L-board is the application control of active voltage states, making it possible to dynamically tune the hardware for the running application. The SAM4L board is depicted in Figure 2.5.

The SAM4L board we use for our research contains the ATSAM4LC4CA microcontroller. The microcontroller contains an ARM Cortex M4 processor, and supports two different run modes. RUN0 operates with a core voltage of 1.8V, while RUN1 operates at 1.2V. The microcontroller also supports various low power states, i.e., sleep modes, where the processor clock is turned off, and multiple oscillators with different frequency ranges.

Unlike the EFM32 board, the SAM4L board does not have an embedded energy monitor. A myDAQ board from National Instruments (NI) is therefore used to measure and log the live current consumption of the SAM4L

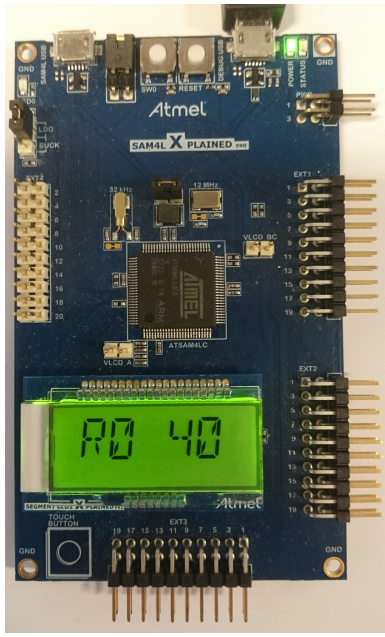


Figure 2.5: SAM4L board from Atmel.

microcontroller and the CPU voltage. The NI myDAQ board is depicted in Figure 2.6. The NI myDAQ is supplied with a software that enables efficient measurements through a PC connection. In addition the myDAQ board comes with LabView integration, which enables developers to develop custom measurement and logging software. How we setup and measure the live current consumption and voltage, and use these measurements to find the power and energy consumption of the our applications is part of our contributions described in Chapter 3.

2.6.3 GEM5 CA simulator

A computer architecture (CA) simulator incorporates detailed performance models and uses them to approximate the behavior of real hardware. They enable computer designers to experiment with a variety of different configurations early in the design stage of a system and investigate interesting tradeoffs before the prototype stage of the design process [99].

A number of CA simulators exist today in addition to the GEM5 simulator [12], such as SimpleScalar [9], Sniper [15], and Multi2Sim [110].

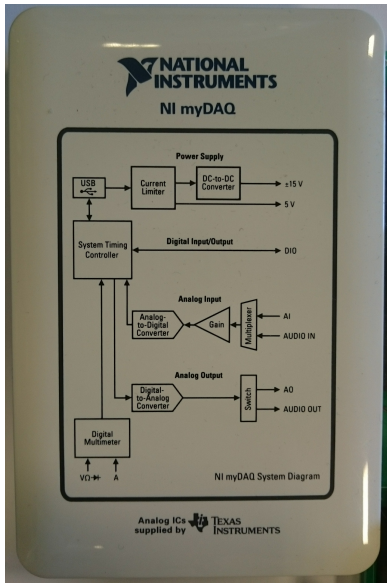


Figure 2.6: myDAQ board from National Instruments.

Multi2Sim [110] supports only out-of order execution and is mainly intended for CPU-GPU computing. SimpleScalar [9] and Sniper [15] are application-level simulators as opposed to GEM5, which is a full system simulator. The benefit of an application-level simulator is the ability to run only target applications instead of a full fledged target operating system (OS).

The GEM5 simulator is widely used in academia today, and supports various processor models, such as Alpha, ARM and Intel processors. In addition to being a full system simulator, GEM5 can emulate system calls (SE mode) and run target applications without involving the OS of the simulated kernel. This will speed up the simulation time significantly, and the user can focus more on implementing the functionality of non OS-controlled performance mechanisms used by the application. In GEM5, the processor platform is modeled as C++ kernels, and the HW interaction with the application is simulated through Python, Swig and Ruby scripts at the user level. Figure 2.7 shows an overview of the basic GEM5 architecture, which is defined without caches. We use this architecture as a starting point in our research, with GEM5's ARM processor model.

McPAT [63], which is an integrated power, area, and timing modeling framework for multi-core and many-core architectures, is often used together with the GEM5 simulator output statistics to estimate the overall

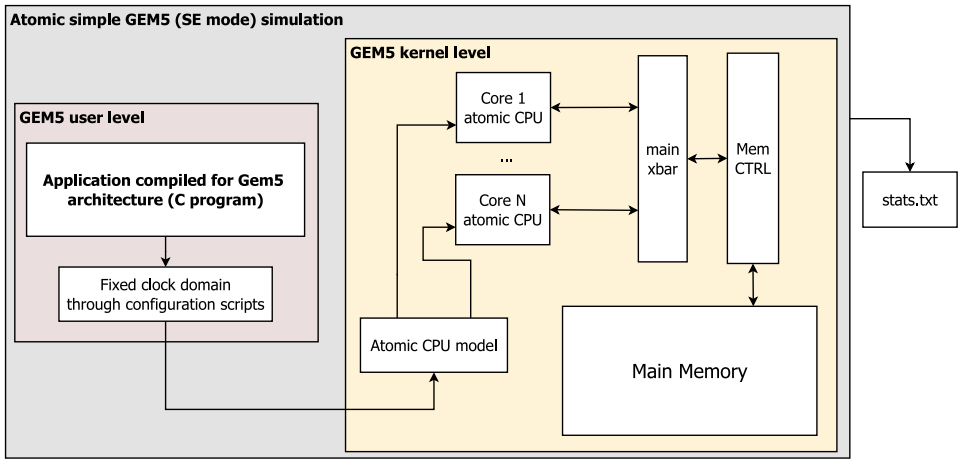


Figure 2.7: The GEM5 architecture used as a starting point for the research in this thesis.

energy consumption. This framework includes models for the fundamental components of different types of processor cores. With a flexible XML interface, McPAT can be interfaced with many performance simulators. McPAT focuses mainly on high performance processors with high complexity, and all parts of the system architecture must be modeled properly in order to extract the correct energy result from McPAT.

2.7 Representative example of an application with dynamic behavior

A typical example of an application with dynamic behavior in the multimedia domain is the H264/AVC codex (hereby referred to as H264). In this dissertation, our developed techniques for system scenarios are tested on the H264/AVC encoder control structure. This section presents a brief overview of the H264 encoder behavior.

The H264 codex is an advanced and flexible codec, which contains many multi-valued input parameters and features which directly affects the processing time of encoding and decoding [118]. The H264 encoder obtained from the Mediabench II website [60] contains multiple nested loops, where their loop sizes are dependent on the frame size through other parameters in the code. Hence, the nested loop structure within the application is data

variable dependent. Within each nested loop the code performs memory accesses to process the video to be encoded. The processing time for a large frame size is therefore significantly longer than for a small frame size, and this information could be exploited to adapt the platform according to the processing time of the encoder with DVFS. In our research, we focus on situations where the input frame size into the encoder control structure varies. This can for instance be caused by dynamic adaptation to changes in a wireless transmission channel. Although the video content by itself is dynamic in behavior, we leave all other parameters in the encoder constant for simplicity. In the next section, we will present an example of a data dependency problem in the neural science domain.

2.8 Example of a data dependency problem: Extracellular monitoring of the action potentials in animal's brains

In this dissertation we develop an application to solve a data dependent problem in neural science. This application contains data variable dependencies that are controlled by different parameters. It serves as an example in which we can apply our techniques for data variable dependent system scenarios, presented in Chapter 3, to reduce the energy consumption. This section covers the necessary background information in order to understand a realistic challenge neural scientists face when studying the behavior of neural cells and networks in animals brains.

2.8.1 The study of neural cells and networks

The behavior of neural cells and networks are studied through extracellular monitoring of the action potentials in animals' brains *in vivo*. The activity is normally recorded using very small electrodes on implantable neural probes. The digitized signals are then transmitted to a remote receiver where detailed analysis is performed using advanced signal processing algorithms.

The electrodes detect extracellular voltage deflections that are a mixture of action potentials (spike band: 300 Hz - 6 kHz) emitted by nearby neurons and slower fluctuations that reflect population activity (i.e., local field potential band: 0.1 Hz - 300 Hz). We focus on a data reduction method

applied to the spike band, based on the detection of action-potentials (also known as "spike detection") [86].

At least 20 kHz sampling rate is required to capture the spike waveform necessary for spike detection and further analysis. Figure 2.8 illustrates a typical probe and its electrode arrangement. The figure also shows the bandpass filtered signal (0.250 - 5.938 kHz) from each electrode at 25 kHz sampling rate, which are analyzed together in order to extract relevant spike data for post-processing. Due to the high electrode density, a spike detected in one electrode is also detected in neighboring electrodes. One of the main challenges of using such electrodes is to capture temporal and spatial spike data in noisy environments. Temporal spike data is captured by inspecting a single channel over time, while spatial spike data is captured by inspecting multiple channels with different spatial placement on the probe.

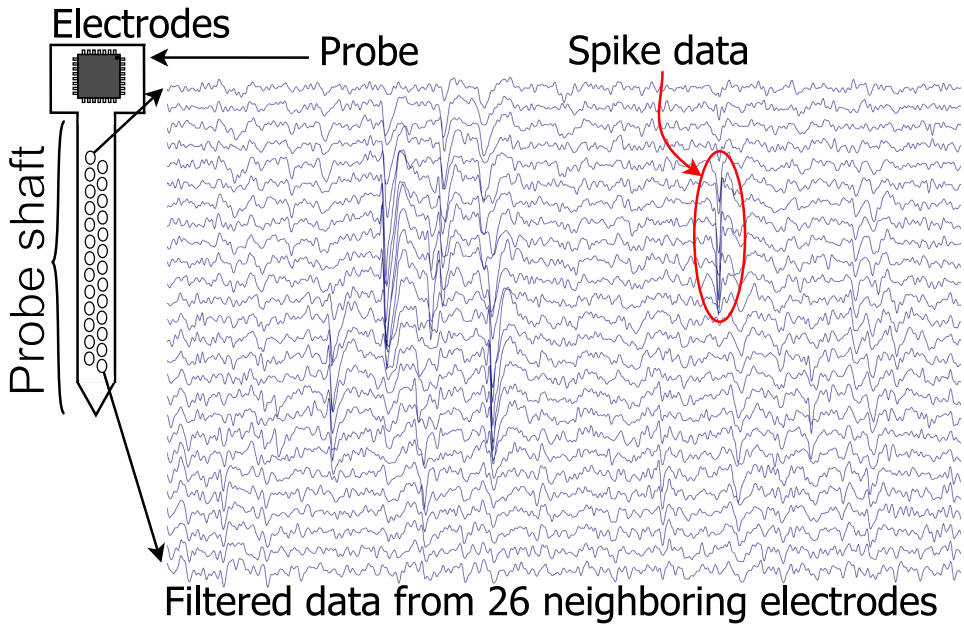


Figure 2.8: The filtered recorded signal of a probe with multiple electrodes, clearly showing the spikes and spatial signal dependency.

The number of electrode channels will soon reach the limits of how much data we can transfer within reasonable energy budgets, even while transmitting the data through electrical wiring [88]. The reason is the fast growing number of concurrent sensor nodes (electrodes) on a neural probe. The state-of-the-art neural probe produced with silicon technology at the start

of 2017 contains 1356 electrodes [88]. Raducanu et al. [88] show that the transmission circuitry contributes significantly to the power consumption. Other research projects aim towards thousands of electrodes on a single neural probe in the relatively near future [100], [20]. This will exacerbate the transmission problem even more.

Researchers also aim at reducing the number of wires connected to the probe, or eliminate them all together, in order to have more freedom of movement while recording. A wireless transmitter embedded on a battery powered probe will limit the amount of data that can be transmitted in "raw form" even further compared to wired probes. Such wireless probes introduce a long term energy constraint, i.e., the total available energy based on the battery capacity. Even if the electrodes are implemented with optimal energy efficient circuitry, it is still a challenge to meet the energy constraints related to wired and wireless data transmission for future high-density electrodes. The only way to meet the requirements of the long term energy constraint is to perform data reduction to be able to transmit less data. For all data reduction approaches it is therefore important to capture sufficiently detailed information about the spike waveforms, such that these can be assigned to distinct neurons remotely (e.g., through a process called spike sorting [86]).

2.8.2 General goal of data reduction approaches

The goal of data reduction approaches is to reduce the transmitted data while preserving all relevant information. What constitutes relevant information is application dependent. Spikes generated by individual neurons are the result of integration of input signals and transmitted to other neurons. For some applications it is sufficient to know the time of a spike regardless of the neuronal source, for example neural decoding approaches based on activities from multiple neurons (multi-unit activity) [101]. However, for most applications additional information about the wave shape is needed, as this relates to the specific neuron that emitted the spike (different neurons emit spikes with different wave shapes). Wave shape features can be used to improve neural decoding approaches [58]. More often, waveform features are used to identify groups of spikes that originate from the same neuronal source ("spike sorting"; [86]). The spike sorting process relies on differences in intrinsic properties of neurons and their location relative to the electrodes, which are reflected in the spike amplitude and wave shape. After detection

and extraction of spike waveforms from high-pass filtered (>300 Hz) raw signals, the sorting procedure is usually performed by deriving informative features from the waveforms (e.g. peak amplitude or components from principal component analysis, PCA [2]) across a set of electrodes, followed by a feature-based classification algorithm, as summarized in Figure 2.9. Step iii in Figure 2.9 shows an illustration where the x- and y-axis represents the first and second PCA components, respectively. In the final spike sorting step, similar features are clustered together with a label representing its cluster. Conventional approaches transmit the filtered data from the neural probe to a remote receiver, and process steps ii to iv remotely. The goal of our approach is to detect relevant spike waves (temporally and spatially) with minimal amount of resources on the neural probe. We then reduce the amount of data by only transmitting these waveforms to a remote receiver, where steps iii to iv are processed remotely.

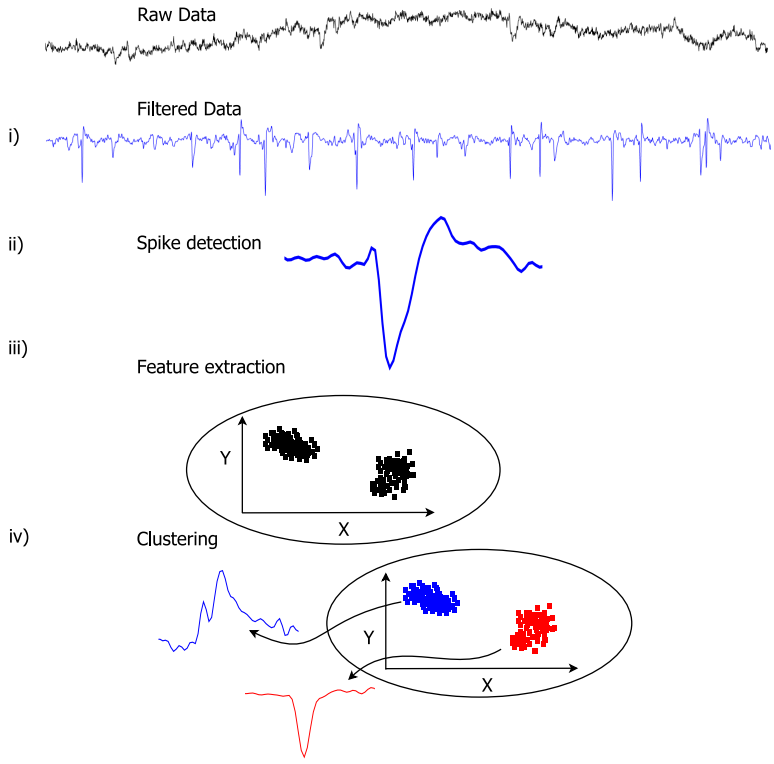


Figure 2.9: Typical spike sorting overview as illustrated in [86].

In this Chapter, the neccessary background theory is presented, in or-

der to understand the contributions presented in this dissertation. These contributions is presented in the next chapter.

CHAPTER

3

PROPOSED FRAMEWORK, PLATFORM IMPLEMENTATIONS AND TECHNIQUES

This chapter presents the contributions from the research work of this dissertation, where a framework for system scenario design is proposed and validated on different platforms. A technique for scenario prediction and switching is proposed, which takes data variable dependencies into consideration. A novel DVFS extension and energy model is developed for the GEM5 computer architecture (CA) simulator. Finally, a data variable dependent data-reduction application is developed, which serves as an application applicable for our system scenario techniques. How to apply our techniques for prediction and switching on this application is discussed at the end of this chapter. Figure 3.1 shows how the papers of this dissertation are connected together.

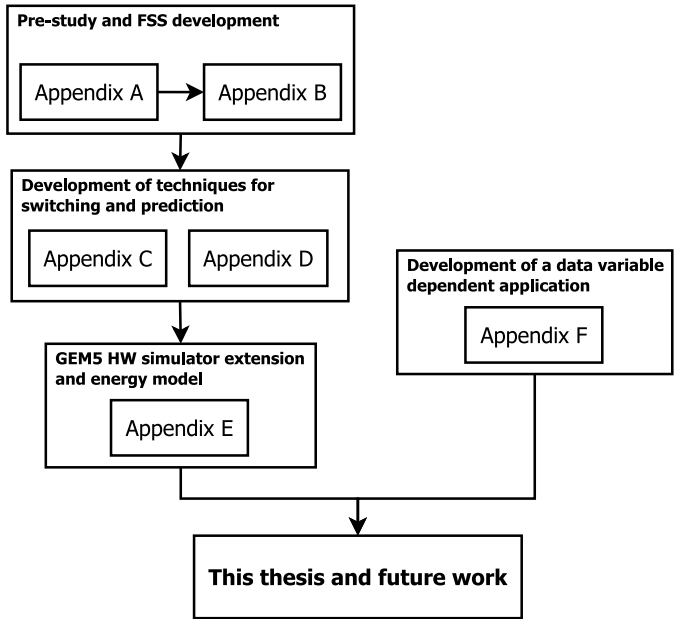


Figure 3.1: Overview of how the papers in this dissertation are connected together.

3.1 Motivating the need for data variable system scenarios

Conventional hardware/software co-design approaches focus mainly on static analysis where no dynamism is present. When dynamism is introduced, then only the worst case behavior is used to implement the system. In order to motivate the need for system scenarios, different data memory prefetching schemes were studied as part of a semester assignment from the course TDT4260 Computer Architecture at the Norwegian University of Science and Technology (NTNU). These prefetching schemes treat the running application as a black box, where only the memory access patterns are monitored by a prefetcher mechanism. This prefetching mechanism then prefetch the data from the main memory into the data cache based on the previous history of access patterns. As a result, different applications with different behavior would have different outcomes with each prefetching scheme. The study is extended beyond the scope of the TDT4260 course and resulted in the development of different adaptive prefetch managers (APMs), which

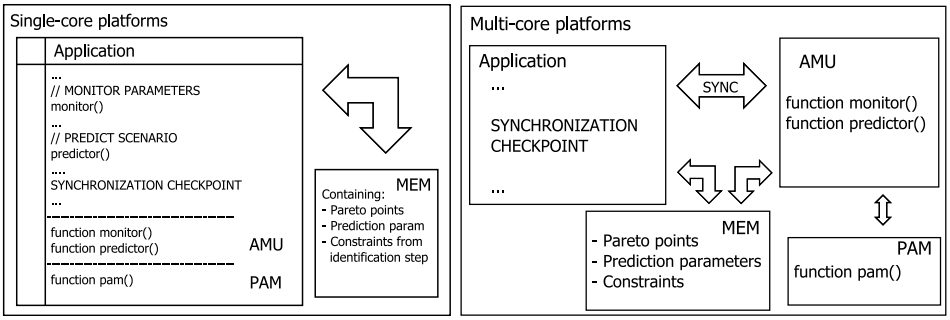
still treats the application as a black box. Only the sequential memory access patterns are monitored and these patterns are used by the APM to find the best applicable memory prefetching scheme. Because the behavior of the applications are not known, the prefetching schemes spend most of their time trying to learn the memory access patterns. When the memory access patterns change throughout the application lifetime, then the history buffers in prefetching schemes become outdated.

This work is presented as a technical report in Appendix A. The best APM solution from this work showed a relative energy reduction in static energy (i.e., energy consumed when the system is in idle) by up to 70.9%, at the cost of a 4.5% relative increase in dynamic energy consumption (energy consumed when the system is executing tasks). However, the static energy reduction only occurs when the APM detects a simple prefetching pattern and turns off the memory required for the history buffer. This work motivates the need for more insight in the application behavior and to exploit it for further energy reductions (including dynamic energy reduction), which is a major goal of the system scenario design methodology. In the next section we present our framework for system scenarios, which we use to apply system scenario methodology on applications with dynamically varying application behavior.

3.2 Framework for system scenarios (FSS)

We propose a generic framework model that enables the use of a system scenarios approach through an application monitoring unit (AMU) and a platform adaption manager (PAM) for single-core and multi-core architectures. This is illustrated in Figure 3.2(a) and 3.2(b).

In single-core architectures, the AMU and PAM are implemented in software (SW) as lightweight inline functions with minimal overhead. In homogeneous or heterogeneous multi-core architectures, the AMU and PAM could be implemented in hardware (HW) running in parallel with the application, or as co-processors where parts of the AMU and PAM are implemented in software. This makes our framework also applicable for customized single-core architectures with AMU and PAM implemented as customized units. The synchronization checkpoint in Figure 3.2 are locations in the application code where the RTS parameters monitored by the prediction mechanisms are evaluated. Based on the outcome of the evaluation, the switching mechanism triggers a platform re-configuration. Where the synchronization



(a) Single-core FSS framework. **(b)** Multi-core FSS framework.

Figure 3.2: Proposed FSS framework.

checkpoint is placed in the code depends on the behavior of the application, and a trade-off must be made based on how often we wish to re-configure the platform versus the overhead cost of the monitoring and re-configuration mechanisms.

The FSS framework is an extension to the RRM framework, where the platform and configurations adapt according to different system scenarios occurring dynamically within a running application. Custom platform related commands, such as a write to a clock division register, are implemented using inline function calls. These functions belong to our PAM module, which consists of platform specific functionality. Our AMU mechanism consists of lightweight inline functions that triggers the PAM functions dynamically at run-time. Our AMU mechanism also monitors loop variables of nested and consecutive loops in the running application, and predicts at run-time which system scenario is about to take place before the loops execute. Lookup tables (LUTs) are used to store the AMU's prediction parameters. The FSS framework introduces more freedom to apply power and energy reduction techniques, and provides a generalized framework for the system scenario design methodology.

3.3 System scenario framework implementation

In this section, we describe how our FSS framework is implemented and validated on our evaluation boards and GEM5.

3.3.1 EFM32

Our single-core FSS framework is prototyped on an EFM32GG-STK3700 board from Silicon Labs [97]. The open source H264/AVC code obtained from the Mediabench II website [60] is analyzed and profiled using video samples with different input sizes. The whole encoder part is extracted, but we focus on the control and data dependent control structures, which is determining the system scenario behavior. Everything not related to the input frame size is replaced with delay statements (no operation ASM instructions, NOP) delaying the code with the same number of clock cycles as the original code. Delay statements are inserted under the assumption that memory load and store instructions only need one clock cycle to finish its operations. Different control variables are tracked throughout the profiling, and system scenarios are classified based on the length of the nested loops controlled by the RTS parameters. DFS is the only available system knob that can be tuned without disabling the CPU at run-time on the EFM32 board. DFS is enabled through clock division, where a frequency switch takes one system clock cycle when the system is set up properly. The DFS is triggered by a write to dedicated registers, and the energy consumption is measured through the built-in energy monitor on the EFM32 board. For more information, see Appendix B.

The run-time application controlled re-configuration options on EFM32 board are limited with frequency scaling (DFS). Only the power consumption is reduced when lowering the frequency. If the EFM32 board allowed run-time reconfiguration of the core voltage, then our framework could utilize dynamic voltage and frequency scaling (DVFS), and hence be able to reduce the energy consumption of the platform. In the next section we will present our framework implementation on a SAM4L microcontroller board from Atmel, which allows application controlled DVFS at run-time.

3.3.2 SAM4L

Unlike the EFM32 board, which only allows the control of the processor's clock frequency while the application is running, the SAM4L board allows dynamic control of both voltage and frequency. By controlling both voltage and frequency, it is possible to obtain a quadratic or cubic reduction in the dynamic energy consumption by reducing the frequency and voltage. This can be observed from the formula for dynamic energy consumption in

Equation 2.1 presented in Chapter 2.5.

Preliminary board setup

During our experiments we measured the live current consumption and core voltage on the SAM4L board using a NI myDAQ measurement device [77] as shown in Figure 3.3.

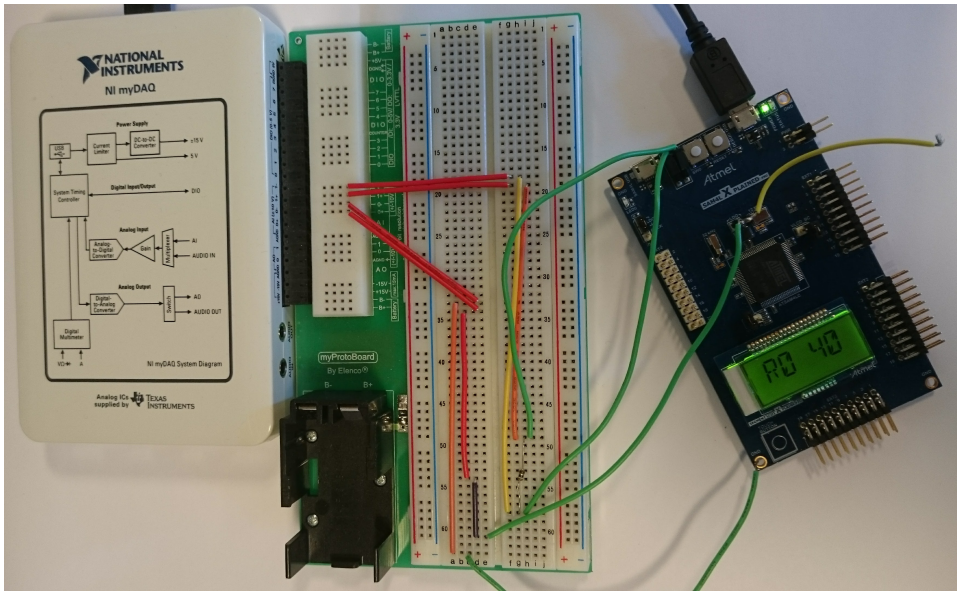


Figure 3.3: SAM4L setup with NI myDAQ.

The NI myDAQ device was connected to a computer and the time stamped data was logged with a locally developed LabVIEW program, and used together with our voltage and current measurements, to find the energy consumption of the application. The single-core FSS framework is used to implement the H264/AVC encoder control structure with system scenarios. Figure 3.4 shows a screen dump from our locally developed graphical LabVIEW program with the different power modes of the board.

For the scope of our research, we have selected the internal oscillators RCFAST (12 MHz) and RC80M (80 MHz) as the main clock sources for our DVFS settings. Since the ARM Cortex M4 processor only supports frequencies up to 48 MHz in SAM4L, the RC80M oscillator is clock divided to 40 MHz when it is selected as the main clock source. Hence, our DVFS settings are PS0 (1.8 V and 40 MHz) and PS1 (1.2 V and 12 MHz). More details around the experimental setup, including our board modifications to



Figure 3.4: SAM4L measurements with our own developed Energy Monitor.

reduce the switching time, is explained in detail in Appendix C.

Framework implementation

This work further extends our FSS framework from only using DFS with clock division on the EFM32 board, to handling DVFS in combination with RTH with the SAM4L board. The NOPs inserted into the nested loops of our H264 application are exchanged with single cycle memory access to SRAM in order to make the model more realistic.

The frequency and voltage switch setup configurations are implemented in our PAM mechanism, where they are combined together to one inline SAM4L_DVFS function. The monitor functionality of our AMU mechanism are part of the pre-calculation equations used to estimate the execution time of the application. They combine data variable dependent RTS parameters together into one value, which is stored in a global LUT. Our predictor functionality is implemented as a lightweight inline function. It compares the global RTS lookup-table values with global scenario lookup table values in order to determine which scenario the platform will adopt to. This approach is a generalization that takes into account multiple scenario parameters and RTS parameters. In our particular case it would have been

enough to compare the RTS parameter with the threshold between Scenario 0 and 1 directly. The values in the global scenario lookup table is found from our identification step. If the value of the global RTS lookup table is greater than the value in the global scenario lookup table, then the platform changes its configurations to PS0 if it is in PS1, and vice versa. The platform adaption manager (PAM) function is then called by the AMU predictor function and performs the platform re-configuration using the DVFS system knob.

In addition to our H264 scenario implementation, we used the scenario methodology to utilize RTH more energy efficiently by exploiting available sleep modes. Such exploitation was used by Awan et al. [10] to utilize RTH more energy efficiently. However, they determined the best feasible sleep mode by computing the available slack online. Our system scenario approach enables us to simplify the solution by Awan et al. significantly by exploiting our pre-calculated loop execution times from our AMU mechanism.

How our global scenario lookup table parameters are derived and how they are used for scenario prediction are described together with our proposed methodology in Section 3.4.

The SAM4L board is limited with only two voltage states, which restricts our framework with only two optimal DVFS settings that could be selected at run-time. In order to dynamically adapt the platform to the behavior of the application more accurately, we need more voltage states available. In our SAM4L work we show how more voltage states could lead to improved energy consumption. We therefore continue our research with the implementation of six DVFS states in the GEM5 Computer Architecture (CA) simulator. Our GEM5 work is presented in the next section.

3.3.3 GEM5

In order to further improve the DVFS and RTH energy saving techniques, custom circuitry are often required, such as platform re-configuration circuitry. Only a few of the available low energy general purpose embedded processors or microcontrollers are able to dynamically tune their power configurations at run-time, such as our SAM4L microcontroller board [8]. However, the SAM4L microcontroller is limited with two available voltage states while the processor is active. The increasing dynamic behavior in advanced applications used in mobile platforms enables more opportunities to

implement energy saving approaches with finer granularity. Finer granularity leads to the need of more run-time re-configuration settings, and many computer architects are using programmable HW simulators to model and analyze new processor designs and energy saving techniques [4].

In this dissertation we extend the widely used GEM5 simulator in order to add more system knobs to our system scenario framework. We propose 1) a novel non-intrusive application controlled DVFS management implementation in GEM5 SE mode, with custom DVFS pseudo-instruction in the ARM instruction-set architecture, and 2) a novel architecture independent energy model based on engineering friendly classification of different workload classes. Power measurements from real HW are used together with the GEM5 output statistics to estimate the total energy consumption of the simulated system.

Our application controlled DVFS implementation in GEM5 focuses on simulating low power embedded systems with less complex architecture than high performance processors. Our DVFS mechanism is application controlled through custom DVFS pseudo-instructions, and our energy model is calibrated with actual performance measurements from the SAM4L microcontroller board.

The FSS framework from our SAM4L board is ported to the GEM5 simulation environment with minimal modifications to the PAM functions. The PAM functions were exchanged with our custom DVFS pseudo-instruction implemented in the GEM5 simulatior kernel. The SAM4L measurement results are used as a reference point when validating the GEM5 simulator results. After systematic offset error correction the GEM5 simulator results deviates at most 4% from the SAM4L measurement results, which we find acceptable.

After the validation we extended the available number of DVFS configurations in the GEM5 simulator from two to six DVFS settings. The AMU functions and the scenario LUTs were also extended to support six DVFS states with new scenario parameters. These parameters are derived from the scenario identification step considering six DVFS states are available. Our finer granularity of available DVFS modes resulted in upto 36.3% energy improvement compared to only having two DVFS states available. However, this energy improvement depends on the exploitable dynamism present in the application, and an improvement is only present if the application requires high performance occasionally. On the other hand, if the application is less dynamic, i.e., high performance is required for the majority of the application lifetime, then the benefit of having finer DVFS modes

available is reduced. In the worst case, the overhead might even become larger than the gain.

The AMU and PAM mechanisms described so far requires a methodology in order to be optimal for a particular application. In the next section we will present our methodology and explain how we apply our system scenario techniques when data variable dependencies are present.

3.4 Data variable based system scenario methodology

In data variable dependent applications, such as streaming media applications, the behavior is determined by data variables. One example of such applications is the H264 encoder control structure we use as the main driver in our research. Static analysis is not enough because the available parameter space explodes when considering all possible values for the different RTS parameters in the application. In the following sections we will present our methodology for applying the system scenario techniques on applications with a dynamic data variable dependent behavior, before we give examples on how we solve such dependencies with system scenarios in the H264 encoder control structure.

3.4.1 Methodology Overview

Before applying our techniques for prediction and switching on data variable dependent applications, the application code must be analyzed to see whether or not there is enough dynamism present in the application. Our methodology targets media applications with similar behavior as the H264 encoder control structure. Figure 3.5 shows a flow chart which is used to identify data variable dependencies in the application code, and to identify whether or not the application contains enough dynamic behavior to benefit from the system scenario methodology. We use this methodology as part of the system scenario identification step, and our methodology focuses on identifying scenarios which impacts the application's execution time, such as variations in the loop conditions that changes the length of nested loops. A similar approach can be used to identify scenarios related to memory usage or other types of scenarios, but this extension is out of the scope of this

dissertation.

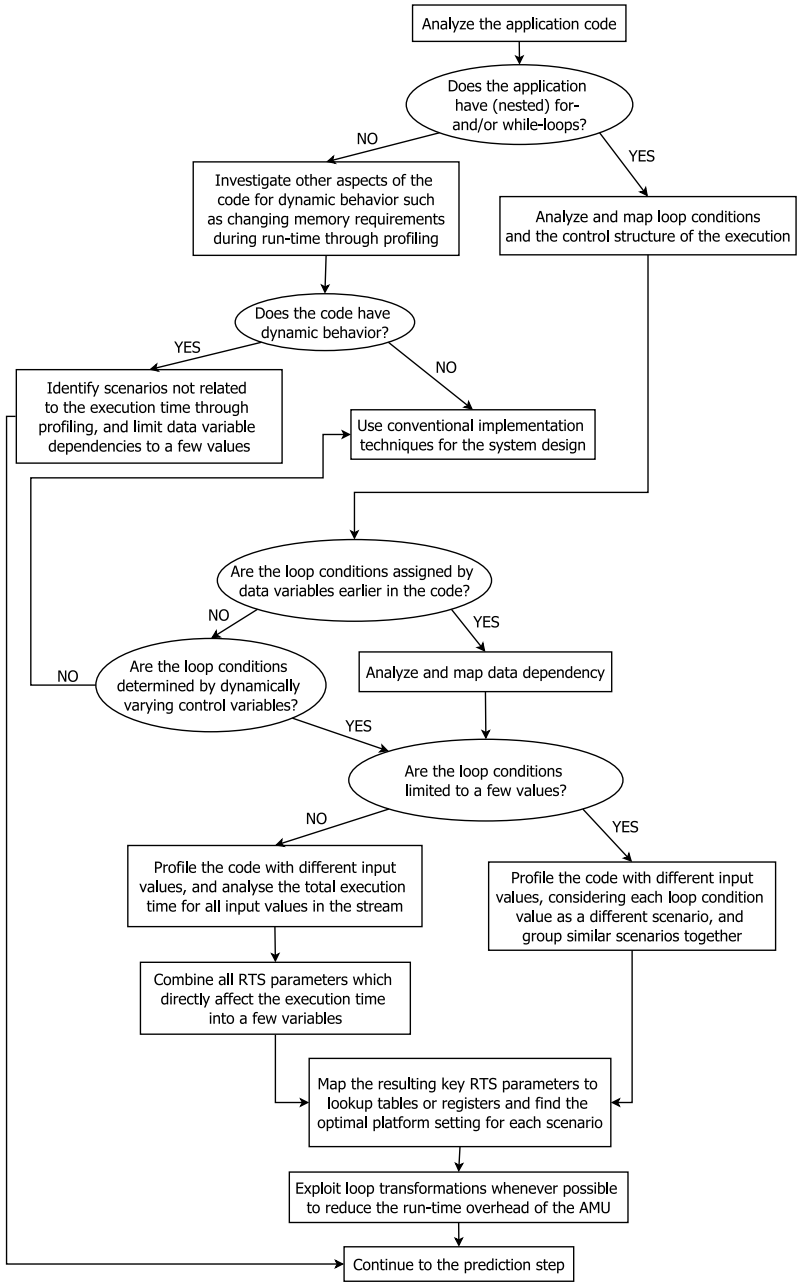


Figure 3.5: Flow chart for identifying dynamic data variable dependencies and classifying system scenarios.

When analyzing the application's code, we first look for conditions that may lead to dynamic variations, such as nested loops. If there are loops or nested loops that define their conditions from variables assigned earlier in the application code, then these variables controlling the loops are profiled in order to identify the dynamic variation in the application behavior in terms of execution time. These variables are the RTS parameters of the application, and typical control variable techniques for system scenarios may be used in cases where these variables are limited to a few values. When these parameter values are not limited to a few values, or if they are assigned by data variables which may be any available value within their type declaration, then the number of possible values explodes as motivated in Chapter 1.2. In addition, we only have dynamism in the application's behavior if the behavior varies significantly during the application run-time. When the behavior is almost constant during the application run-time, then no dynamism is present.

In order to identify dynamic variations in the application behavior, the application is profiled with different input values, or a typical stream of input data. The identified RTS parameters are monitored and analyzed. If the execution time of the application varies for different input values during the stream of input data, then dynamism is detected in the workload. The application is profiled with different workloads, before combining the data dependent RTS parameters together, in order to classify value ranges for inputs with relatively close execution times. These ranges are the system scenarios, which are classified based on the profiling information, and the available platform configurations for DVFS at run-time. For example, more available DVFS settings allow us to define more input value ranges for our system scenario selection.

The key for handling exploding data variable dependent RTS parameter values is to reduce the overhead of the run-time predictor. We reduce the run-time overhead by combining different RTS parameters together, such that the prediction mechanism only have to analyze a few variables at run-time. Instead of only focusing on individual values we define value ranges for our combined RTS variables that identifies each scenario. The prediction overhead is then reduced to a simple comparison at run-time to see whether or not the combined RTS parameters are within a scenario range. For example, if the processing time of two different input values in the application does not differ significantly, then these two input values could be processed with the same platform configuration.

When designing the prediction and switching mechanisms, it is impor-

tant to reduce their execution time and energy overhead as much as possible. We focus on using inline functions for our prediction mechanisms, which do not exceed the complexity of an if-else statement or a simple switch-case statement comparing values from profiled scenario parameters stored in LUTs with the run-time values. In order to reduce the overhead cost of the system, it is important to exploit loop transofrmations whenever possible. E.g., if the AMU pre-calculates the length of each loop in the code, then the loops in the code could be re-written to re-use these pre-calculated values, and hence avoid redundant calculations.

The run-time switching mechanism is platform dependent, and triggered by the predictor mechanisms. In order to benefit form a platform re-configuration, the switching time should not be significant, otherwise the energy overhead may be too high. The overhead of the switching mechanisms becomes negligible if the switching time is significantly shorter than the application's execution time. In addition, the number of switches at run-time should also be taken into account because increased switching activity contributes to higher energy overhead.

In the next sections we provide an example on how we applied our methodology to our H264 encoder control structure application. We refer to Appendix C and D for more details.

3.4.2 Design-time profiling based on data variables

The H264 encoder control structure application was profiled at design-time in order to explore the dynamic range of loop iteration count variations for various input frame sizes according to our methodology in Figure 3.5. All RTS parameters defining the loop conditions within the application were combined together into one variable, representing the execution time of the encoder control structure.

In addition, different voltage and frequency settings were profiled in order to find optimal DVFS configurations in combination with RTH. Our profiled results show that the lowest energy consumption is achieved if the maximum available frequency setting for each voltage level is selected. This results in two scenario settings for the SAM4L board, PS0 (1.8V and 40 MHz) and PS1 (1.2V and 12 MHz). A traditional DVFS strategy exploits idle times by changing the frequency and voltage. Using a RTH strategy for each voltage level in addition gives the best energy consumption because all idle times are exploited with the use of low-power states (i.e., by turning

the system off).

3.4.3 Design-time system scenario identification based on data variables

Based on our profiled results, we selected an evenly distributed set of true 16:9 resolutions, which our SAM4L board can process within one second time frame using our extracted model. This selection is shown on the left-most column in Table 3.1.

Table 3.1: Scenario assignment based on run time

Input frame size	Run time (million cycles)	Scenario	Scenario*
816 x 459	34.70	0	0
752 x 423	29.54	0	0
688 x 387	24.79	0	1
624 x 351	20.46	0	2
560 x 315	16.54	0	2
496 x 279	13.04	0	3
432 x 243	9.96	1	4
368 x 207	7.29	1	4
288 x 162	4.54	1	5
224 x 126	2.80	1	5

Table 3.1 also shows the corresponding scenario selections and calculated run times, based on our available power modes in the SAM4L board. The Scenario* column represents the scenario distribution for a platform with six available voltage settings, evenly spaced between 1.05 V and 1.80 V. The scenario parameters in the Scenario* column are used in our six DVFS simulation (based on intra- and extrapolation of voltage and frequency values), and later in our GEM5 simulation.

The largest frame size the SAM4L board can handle from our selection is measured to be 816 x 459. The required run time (in clock cycles) for each frame size is estimated by counting all loop iterations and multiplying with all instructions within each loop based on the program dis-assembly of the extracted model. These numbers are compared to the measured run time for each frame size. Note that with a more powerful processor, the

frame sizes that can be handled will increase. The methodology will still be the same, however. More details regarding the run-time estimation can be found in Appendix C.

In the application code, the number of all loop iterations is calculated as shown in Listing 3.1. The total number of iterations are combined together with other parameters for all the nested loops in the application, resulting in the loopcnt variable.

The largest frame processing time is approximated to 34.7 million clock cycles in an un-optimized compiler setting (-O0 compiler optimization) where no pipelines are utilized on the SAM4L board. We use this frame size processing time as our worst-case scenario, and we exploit it in combination with DVFS and RTH for lower frame sizes. We benefit from DVFS if the processing time of a smaller frame size with a lower frequency setting is less than the processing time of our worst-case frame size. Therefore, we have to take our available system knobs into consideration when we classify our system scenarios.

Our scenario selection is dependent on the platform's running frequency. In PS1 the platform is processing a frame 70% slower than in PS0. Therefore, frame sizes that need more than 30% of the run time of our worst-case frame size are processed with PS0 (Scenario 0), and smaller frame sizes are processed with the PS1 configuration (Scenario 1). This is generalized in Appendix C. Our scenario classifications when having six DVFS states available were derived in a similar way, resulting in the Scenario* column in Table 3.1.

```
1  // Functions for monitoring RTS parameters
2  long loopcnt1 = (size_y + img_pad_size + img_pad_size) * (2 + (3*(
       size_x + img_pad_size + img_pad_size)));
3  long loopcnt2 = ((size_x + 2 * img_pad_size) * 2) * (5 + ((3*(maxy
       - 2 - 2)) + (3*(ypadded_size - maxy + 2))));
4  long loopcnt3 = ((je2 + 2)/2) * (2 + (((ie2 - 2)/2) + 2));
5  long loopcnt4 = (ie2 + 2) * (((je2 - 2)/2) + 2);
6  long loopcnt5 = (size_y/4) * (1 + (6*(size_x/4)));
7  long loopcnt = loopcnt1 + loopcnt2 + loopcnt3 + loopcnt4 +
       loopcnt5;
8
9  // Update combined RTS parameter
10 (g_rts_data)[0] = loopcnt;
11
12 // Scenario detection/prediction
13 AMU_1(scenario_old, scenario, g_rts_data[], g_pareto_lc[]);
```

Listing 3.1: Functions combining data-dependent RTS parameters

3.4.4 Run-time system scenario detection and switching based on data variables

The input data variable dependency of our H264 encoder control structure is shown in Listing 3.1, where inline application dependent equations are pre-calculated (lines 2-7), before a global lookup-table is updated with the combined RTS parameter (line 10) as part of the overall AMU. This RTS parameter is then read by the AMU_1 function, which detects the next scenario (line 13). The application dependent calculations in lines 2-6 correspond to succeeding (nested) for- and/or while-loops in the encoder control structure, which are combined together in line 7 to estimate the required frame processing time in terms of loop iterations. In our particular case it would have been enough to compare the loopcnt variable with the threshold between Scenario 0 and 1 directly, and use the result to determine the next scenario.

```
1  static inline void AMU_1(volatile int *scenario)
2  {
3    if( ((g_rts_data)[0] > (g_pareto_lc)[0]) ){
4      *scenario = 0;
5
6      DVFS_SAM4L(scenario); // PAM function
7    }
8    else {
9      *scenario = 1;
10
11     DVFS_SAM4L(scenario); // PAM function
12   }
13 }
```

Listing 3.2: Scenario selection in AMU, calling the PAM DVFS_SAM4L function

However, we have generalized this comparison by using the AMU_1 function shown in Listing 3.2. This lightweight inline function compares the global RTS LUT values with global scenario LUT values in order to determine which scenario the platform will adapt to. This generalization takes into account multiple scenario parameters and RTS parameters. The use of lookup tables would simplify an extension to multiple independent RTS parameters and/or scenarios.

In our scenario implementation we execute the AMU_1 function before the loop structures are executed as shown in Listing 3.3. After each frame is processed in Listing 3.3 the application is halted through the sam4l_rth

function, which wakes the SAM4L microcontroller just before the next set of input data arrives at a 1 second interval.

```
main(){
  while(1){
    // New input frames arrive
    //...
    // Predict/detect the upcoming scenario
    AMU_1(*scenario);
    // Nested loops of the H264/AVC encoder control structure
    //...
    // Power down until the next input frames arrive
    sam4l_rth();
  }
}
```

Listing 3.3: Application structure with prediction and switching

Our scenario implementation is validated using a 500 second sequence of frames on three different wireless networks with dynamically varying bandwidth availability. The bandwidth availability determined the frame size to be processed by our H264 encoder control structure. Compared to not using scenarios, our system scenario approach resulted in significant reductions as shown in Figure 3.6. These results show the energy reduction with the recorded bandwidth availability for the 500 second sequence. In our particular case, the results also show the overhead in situations when not enough dynamism in the available bandwidth is present. These results are expected to change if the available bandwidth changes differently for other sequences. The average performance overhead, which initially was 8.5%, is reduced to 0.7% after implementing the improvements we will describe in Section 3.4.5. A detailed description of the validation process is described in the journal paper presented in Appendix D.

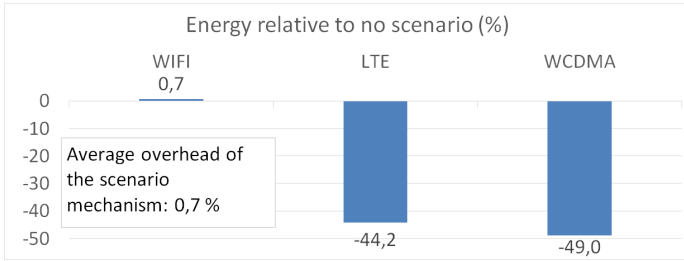


Figure 3.6: Energy consumption relative to no system scenarios.

3.4.5 Improving the energy overhead and exploiting PMM

The pre-calculations presented in Listing 3.1 are further optimized by applying loop transformations in the application. This optimization is applied as part of the exploitation step in the system scenario design methodology presented in Section 2.1.2. Instead of using redundant re-calculations, we calculate the loop parameters at the beginning of the application. These calculations are then re-used to estimate the total execution time. These modifications resulted in a significant reduction to the run-time overhead of our scenario mechanisms, with only 0.7% overhead as shown in Figure 3.6.

The techniques described in earlier sections are also applied to exploit the SAM4L board's available sleep modes at run-time with our system scenario mechanisms. This exploitation is implemented by using a dedicated inline function similar to the AMU_1 function. As a result, two different scenario implementations are added to the same application, each controlling different system knobs. The implementation details of our loop transformations and our sleep mode exploitation are described in detail in Appendix D.

3.5 Data dependent application development for system scenarios

In Section 2.8 we presented the theory behind a realistic challenge neural scientists face when studying the behavior of neural cells and networks in animal's brains. During the research work of this thesis we developed a data variable dependent application which reduces the amount of data to be transmitted from high-density electrode arrays significantly. In this section we will briefly present our application, which is described in full detail in the journal paper presented in Appendix F. Instead of giving implementation details, the focus of this section is on how we can potentially apply our techniques for system scenario to our developed application in order to reduce the energy consumption of the processing platform.

3.5.1 Automatic data reduction of wireless read-out in high-density electrode arrays

In our application we focus on spike data extraction, using low power and simple digital circuitry. Samples around detected spikes are selected locally on the probe, and all other data are discarded before the selected samples are transmitted to a remote receiver for post-processing. This solution is scalable with increasing number of electrodes, and uses a minimal amount of local storage. Pruning away unneeded data results in large reductions in transmit energy, because we transmit significantly less data with minimal increase in processing/memory access energy only.

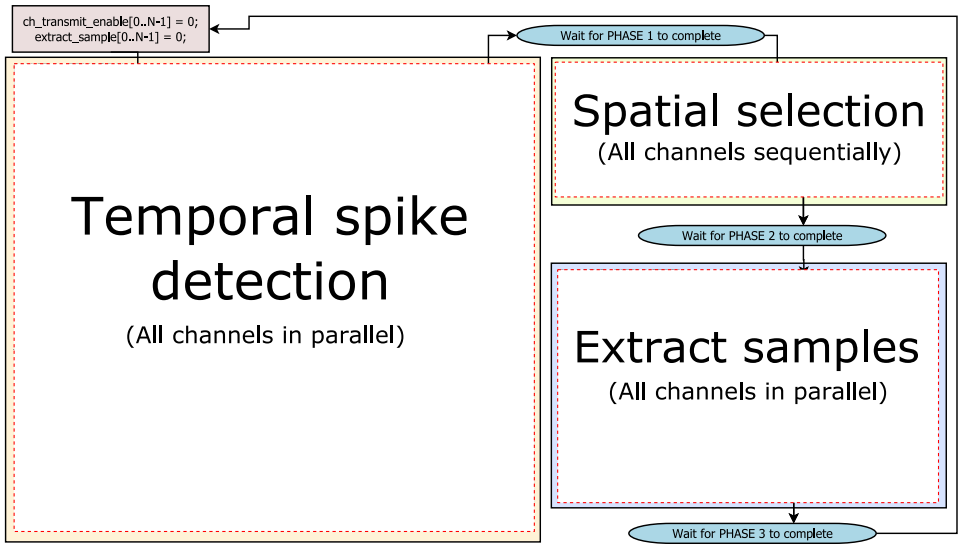


Figure 3.7: 3-phase spike data extraction

We have divided the spike data extraction in three consecutive phases as illustrated in Figure 3.7. All these phases are performed for each sample per electrode. In the initial phase, a temporal spike detection is performed using a crude amplitude thresholding mechanism. The second phase performs a spatial analysis of all detected spikes and marks neighboring samples for extraction. In the final phase, all samples marked for extraction are transmitted. All three phases are described in more detail in Appendix F.

3.5.2 RTS parameters applicable for our system scenario techniques

In this section we will discuss possible ways to apply our system scenario techniques for prediction and switching to our neural science application. The goal of the system scenario design methodology is to reduce the overall energy consumption by exploiting the available dynamism in the application. In our neural science application we could exploit the dynamism in different ways, and during different phases of the application. Normally, system scenarios are used to tune the platform configuration, i.e., system knobs, to reduce the energy consumption. However, we can also use the system scenario design methodology to tune algorithmic knobs, which affect the energy consumption of the platform, by for example reducing the amount of data to be transmitted to a remote receiver.

In Phase 1 in our application, spike data exceeding the statistical threshold in Equation 3.1 are marked for further analysis in the next phase. This is done individually for all electrodes in parallel before advancing to Phase 2.

$$Th_k = M_k + (ThFactor \cdot STD_k) \tag{3.1}$$

In Equation 3.1, Th is the running threshold of sample k, M is the running mean, STD is the standard deviation, and ThFactor is a floating point configurable factor that is used to tune the sensitivity of the threshold. This threshold tuning impacts the energy consumption by how much data is transferred. More data to be transferred results in higher energy consumed.

Phase 2 is a sequential phase that iterates through all electrodes one by one. If one electrode channel detects a spike, then it marks this sample for transmission by setting the variable ch_transmit_enable to 1 (Figure 3.7). A specific number (chFactor) of closely placed electrode channels surrounding the marked channel are also marked for transmission, and thus the chFactor also impacts the amount of data to be transmitted by including neighbors not detected by the threshold in Equation 3.1. Phase 1 has to be completed for all channels before Phase 2 can start, because the analysis of each electrode sample is dependent on the situation of all other electrode channels in the same time stamp. In other words, all channels are processed sequentially in Phase 2, where ch_transmit_enable is set for all channels that are close (in distance) to a channel with extract_sample marked (determined by the chFactor tuning parameter).

How the neighboring channels are selected is determined by a geometrical matrix unique for each probe design based on the placement of each electrode. It is up to the probe designer and neuroscience expert to determine which electrodes are neighbors to each other in the geometric matrix. The HW mechanism which selects the electrode neighbors should be as simple as possible because the sequential Phase 2 can become the bottleneck of the application if it takes too much time. One simple solution is to assign a unique channel id for each electrode, such that the neighboring electrodes can be marked by incrementing and decrementing the channel id according to the chFactor. In our experiments, we use electrode channel ids starting from 0 for the electrode at top of the probe and using ascending channel ids on electrodes placed below. The spatial extraction is then done by marking channels with an ID number equal to the channel for the detected sample in Phase 1 +/- chFactor.

As explained in Appendix F, it is hard to choose an optimal threshold factor (both ThFactor and chFactor) that would balance the detection of false positives and false negatives with our application. However, with the use of our techniques for scenario prediction and switching, the available dynamism in the application stream of inputs could be exploited to fine-tune the ThFactor and chFactor parameters dynamically, and hence adapt to the statistical decision making at run-time. These can hence be seen as algorithm knobs.

If the neural probe is bounded by an energy limitation, i.e., not all channels are able to transmit its samples simultaneously within the power budget, then the ThFactor can be increased to make the electrodes less sensitive. The ThFactor could then be decreased after a time interval or based on statistical changes in the probe environment when the power consumption falls below the power limit. Our FSS framework does not require any significant modifications in the application, and the power budget monitor could be combined with the threshold calculation in Phase 1 through the use of our scenario LUTs. However, the outcome of the thresholding mechanism is very sensitive to the ThFactor. This factor may be challenging to tune because small variations in this factor could lead to an increase in false positives or false negatives.

Instead of hard-coding the chFactor, we can apply a predictor that calculates the amplitude difference between neighboring samples. Phase 2 could then continue including channel IDs until the amplitude difference increases significantly. That makes chFactor usable as another algorithm knob. One potential risk with this approach is that too many neighbors could be in-

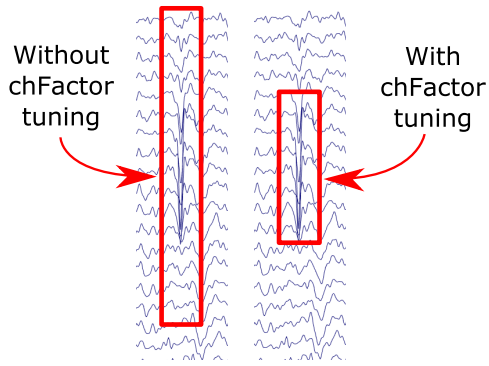


Figure 3.8: Potential spatial selection improvements when using chFactor as an algorithm knob.

cluded in noisy situations. On the other hand, chFactor is easier to tune compared to the sensitive ThFactor. Figure 3.8 shows an illustration of how we can improve the spatial selection of Phase 2 by dynamically tuning the chFactor. As shown in Figure 3.8, we risk including unwanted samples spatially if we have a fixed chFactor. If we allow the chFactor to be dynamically tuned, then we can potentially reduce the transmitted data even more by only including the samples of interest. Similar results are expected if the ThFactor is tuned dynamically.

So far we have discussed possible ways to use our system scenario techniques to optimize the algorithmic behavior of the application. We could also use DVFS and RTH as system platform knobs to speed up the bottleneck of the application, i.e., Phase 2, through system scenarios in a similar way as we did for our H264 application. Since Phase 2 is a sequential phase, we can tune the speed of Phase 2 by increasing the frequency and voltage of the processor when many electrode channels detect a spike during Phase 1. One bit in a flag register (one bit per channel) can be set each time a channel detects a spike (and cleared at the beginning of the next phase). The number of one's in the flag register could then be added together in order to determine how many channels have detected a spike in parallel. If we combine this DVFS tuning with RTH then we could reduce the energy consumed on the neural probe itself in this phase by exploiting the available slack time until the next set of samples arrive. All our proposals in this section are subject to future work and the implementation details of our proposals are out of the scope of this thesis.

CHAPTER

4

RESEARCH RESULTS AND CONTRIBUTIONS

This thesis is a collection of papers and technical reports that have been authored during the PhD work. Each paper is presented in the appendixes. Rather than including the double-column PDF files, each paper is formatted to increase readability of the graphs and text. However, the text or figures are not altered, only the layout and size. This chapter presents an overview of all the papers in this paper collection. The order of the papers presented here is in rough chronological order and corresponds to the different research contributions. In practice, some of the research in these papers have been conducted concurrently.

4.1 Paper 1: Adaptive Run-Time Prefetch Manager using Sequential On-Access and DCPT Prefetching

The work related to this contribution is presented as a technical report in Appendix A.

4.1.1 Abstract

Advanced prefetching schemes are able to achieve significant speedup improvements compared to solutions without prefetching. We combine prefetching schemes with different complexities using different adaptive prefetch manager (APM) solutions. Compared to a DCPT prefetching scheme, our best APM solution (DMAP-APM) reduces the static energy consumption by up to 70.9%, with 4.5% relative increase in dynamic energy consumption. Our DMAP-APM solution can adapt to different applications by detecting known sequential memory access patterns, and switching to a dedicated scheme that consumes less static power and energy than an always active standalone advanced scheme, i.e., the DCPT scheme.

4.1.2 Main Contributions

This work had a starting point as a semester assignment from the course TDT4260 Computer Architecture at the Norwegian University of Science and Technology (NTNU). Through the study of different prefetching schemes different adaptive prefetch managers (APMs) were developed, which considers the running application as a black box. Only the sequential memory access patterns are monitored and these patterns are used by the APM to find the best applicable memory prefetching scheme. This technical report highlights the limitations when treating an application as a black box without exploiting the application behavior. The result is often the proposal of more advanced HW mechanisms that do not necessarily reduce the system's energy consumption significantly. This work motivates the need for more insight in the application behavior and to exploit it for further energy reductions, which is a major goal of the system scenario design methodology.

4.1.3 Roles of the Authors

In this technical report, Yassin contributed significantly with ideas and the project planning. Yassin also participated in and supervised the co-authors with the development of the DCPT prefetcher scheme and the adaptive prefetch manager implementation. Yassin modified available scripts to make GEM5 simulation files compatible with McPAT v1.0 and used McPAT to

perform power/energy simulations. Finally Yassin contributed significantly to the composition of the technical paper. Kjeldsberg, Catthoor, and Natvig mainly reviewed and provided feedback on the contents of this work, which resulted in this report.

4.2 Paper 2: System Scenario Framework evaluation on EFM32 using the H264/AVC encoder control structure

The work related to this contribution is published as a conference paper [126] and is presented in Appendix B.

4.2.1 Abstract

Hand held battery powered devices are becoming increasingly complex, with embedded applications that regularly demonstrate very dynamic behavior and requiring advanced functionality. To avoid prohibitively high power consumption, novel techniques are needed to handle the increased dynamicity and complexity, e.g., through use of system scenario based design. In this paper a framework for system scenario resource management is presented and evaluated with the H264/AVC encoder control structure on an EFM32 microcontroller. By using DFS as a system knob, we obtain significant power reductions for different system scenarios occurring dynamically while the application is running. The framework overhead is negligible.

4.2.2 Main Contributions

The results of this contribution is based on the knowledge obtained from the work summarized in Section 4.1, and from an extensive literature study of previous work on the system scenario design methodology. A generalized and flexible framework for system scenario resource management (FSS) is proposed, that can be used by researchers in system scenario based designs. This framework takes into account the limitations observed from the research results in related work. The system scenario design methodology is

applied on a H264 encoder control structure. The whole framework and the application is implemented on an EFM32 microcontroller board from Silicon Labs, where the processor frequency is used as a system knob.

4.2.3 Roles of the Authors

This work is a result from an extensive literature study of previous work related to the system scenario design methodology. Yassin identified the limitations and requirements for the prediction and switching steps of the methodology from the study of previous work, which resulted in his framework proposal for system scenario designs. Furthermore, Yassin tested this framework on an H264/AVC encoder control structure, which he derived from the MediaBench II website, before implementing the complete framework and application on an EFM32 microcontroller board from Silicon Labs. The power consumption was measured with the built-in power monitor on the EFM32 board. Kjeldsberg and Catthoor provided Yassin with overall guidance and feedback on the work and composition of this paper.

4.3 Paper 3: Dynamic hardware management of the H264/AVC encoder control structure using a framework for system scenarios

The work related to this contribution is published as a conference paper [124] and is presented in Appendix C. Section 4.4 presents the journal extension of this initial conference contribution.

4.3.1 Abstract

Many modern applications exhibit dynamic behavior, which can be exploited for reduced energy consumption. We employ a two-phase combined design-time/run-time methodology that identifies different run-time situations and clusters similar behaviors into system scenarios. This methodology is integrated with our framework for system scenario based designs, which dynamically tune the hardware to match the application behavior.

We achieve significant energy reductions for an extracted control structure of a video codec widely used in hand-held devices today. We encode a video stream consisting of different frame sizes based on measured available wireless bandwidth. Energy consumption is measured with a modified microcontroller board from Atmel with two alternative voltage and frequency settings. While maintaining the perceptual video quality and frame rate, our method results in up to 44% energy reduction for our encoded streams, even after including an average worst-case tuning overhead of 8.5%. In reality this overhead is typically negligible since the worst-case assumes very frequent tuning, while in realistic situations it will be performed at a much lower rate. This makes the expected gain over 50%.

4.3.2 Main Contributions

The FSS framework presented in Section 4.2 is extended with control of voltage and frequency for DVFS. First, the FSS framework is ported from the EFM32 microcontroller board to a SAM4L microcontroller board from Atmel. The FSS framework is then extended with DVFS knobs to control the voltage and frequency. The H264 encoder control structure is profiled, and key RTS parameters are combined in order to estimate the total execution time of the application. System scenarios are classified based on the different ranges of the data variable dependent RTS parameters, preventing an explosion in possible scenarios at run-time. The scenario mechanisms are implemented and how to obtain significant energy reduction in most situations is shown, with only minor worst-case energy overhead.

4.3.3 Roles of the Authors

In this paper Yassin implemented the framework for system scenarios (FSS) from previous work on the SAM4L microcontroller board. Yassin further developed a technique for scenario prediction and switching on data variable dependent run-time situation parameters, which he used to extend the DFS mechanisms applied to the H264 encoder control structure from previous work with DVFS management. In addition to review feedback, Perkis provided Yassin with ideas on how to use the H264 encoder to generate realistic streams while maintaining the perceptual video quality and frame rate.

Kjeldsberg and Catthoor provided guidance, ideas and feedback on the research work. All co-authors contributed with feedback on the composition of this paper.

4.4 Paper 4: Techniques for dynamic hardware management of streaming media applications using a framework for system scenarios

The work related to this contribution is published as a journal paper [125] and is presented in Appendix D.

4.4.1 Abstract

Many modern applications exhibit dynamic behavior, which can be exploited for reduced energy consumption. We employ a two-phase combined design-time/run-time methodology that identifies different run-time situations and clusters similar behaviors into system scenarios. This methodology is integrated with our framework for system scenario based designs, which dynamically tune the hardware to match the application behavior. We focus on streaming media applications, and achieve significant energy reductions for an extracted control structure of a video codec widely used in hand-held devices today. We encode a video stream consisting of different frame sizes based on measured available wireless bandwidth. Energy consumption is measured with a modified microcontroller board from Atmel with two alternative voltage and frequency settings. While maintaining the perceptual video quality and frame rate, our method results in up to 49% energy reduction for our encoded streams. The average tuning overhead of our mechanism is negligible after applying simple loop transformations to the encoder control structure. We furthermore show how to obtain up to 34.3% energy reductions for a system with limited slack available, by dynamically exploiting the available sleep modes.

4.4.2 Main Contributions

In this journal paper the work presented in Section 4.3 is extended by improving the overhead of the scenario mechanisms with loop transformations, based on further analysis of the H264 encoder control structure. In addition, the available low power modes on the SAM4L board are profiled, and the system scenario design methodology is applied to select the best feasible sleep mode after the execution of a workload. This additional scenario implementation works independently and do not depend on the initial scenario implementation. As a result, there are two independent system scenario implementations controlling different system knobs for the application at run-time.

4.4.3 Roles of the Authors

This paper is a journal extension of the previous paper "Dynamic hardware management of the H264/AVC encoder control structure using a framework for system scenarios". In this journal extension Yassin applied loop transformations to previous work, which resulted in system scenario overhead improvements. Yassin further exploited existing low power modes on the SAM4L board, and profiled the wakeup time and power consumption for each mode. Yassin used this profiled information to develop a system scenario prediction and switching technique, motivated by related work, which automatically select the best feasible low power mode after the completion of a workload. All co-authors provided Yassin with guidance and feedback on the work and composition of this paper.

4.5 Paper 5: Fast and accurate DVFS implementation in GEM5 using system call emulation mode

The work related to this contribution is submitted as a journal paper and is currently under review. The submitted version can be found in Appendix E.

4.5.1 Abstract

Stringent power budgets in battery powered mobile platforms have led to the development of energy saving techniques such as Dynamic Voltage and Frequency scaling (DVFS). In this paper, we propose a novel non-intrusive application controlled DVFS management implementation in the GEM5 simulator, used with GEM5's system call emulation mode. We also propose a novel architecture independent energy model based on classification of different measurable workload classes and power measurements from real HW. Our energy model is calibrated with power measurements from a SAM4L microcontroller board, containing one ARM Cortex M4 processor. Together with the GEM5 output statistics this model estimates the total energy consumption of our simulated system accurately. The results from our modified GEM5 simulator are validated with two different, representative media applications. Our results deviate with less than 4% after systematic offset error corrections, compared to measurements from our SAM4L board. We obtain a 36.3% improvement in the energy consumption when comparing a 6 DVFS mode with a 2 DVFS mode solution. Our contributions in this paper can easily be tailored to other in-order processor models in GEM5, and will therefore enable system architects to explore new techniques and compare the improvements relative to existing architectures.

4.5.2 Main Contributions

In this work two contributions are provided; 1) a novel non-intrusive application controlled DVFS management in GEM5 SE mode, and 2) a novel Python script-based energy model used together with GEM5. The application triggers DVFS through one custom pseudo-instruction, implemented in GEM5's ARM instruction-set model. This instruction takes advantage of GEM5's utility functions, making it easily portable to other processor architectures. The rest of the proposed DVFS mechanisms are architecture independent. The novel energy model accurately parametrizes different workload classes, which are calibrated with power measurements from real HW. The results from the modified GEM5 simulator are validated with two different, representative media applications. The energy results from the GEM5 simulation are compared to and calibrated with execution results from a SAM4L microcontroller from Atmel, resulting in less than 4% deviation after systematic offset error correction. The proposed implementation

of the DVFS mechanism in GEM5 is, according to the authors' knowledge, the first implementation of an application controlled DVFS mechanism for the GEM5 SE mode. Other published works in GEM5 have targeted DVFS implementation in full system mode (not SE mode).

4.5.3 Roles of the Authors

In this work Yassin extended one revision of the GEM5 simulator with DVFS functionality, and created an energy model. All the practical work was done by Yassin based on his own ideas. Jahre provided Yassin with various suggestions on how to implement DVFS functionality in GEM5, which led to Yassin's novel DVFS implementation. Aunet helped Yassin with understanding and validating Yassin's proposed energy formulas for static power consumption. All co-authors provided Yassin with overall guidance and ideas, including help and feedback with the review process.

4.6 Paper 6: Algorithm/architecture co-optimisation technique for automatic data reduction of wireless read-out in high-density electrode arrays

The work related to this contribution is published as a journal paper [123] and is presented in Appendix F. Together with IMEC, NTNU and KULeuven, a patent application is submitted for this work and the status is currently patent pending.

4.6.1 Abstract

High-density electrode arrays used to read out neural activity will soon surpass the limits of the amount of data that can be transferred within reasonable energy budgets. This is true for wired brain implants when the required bandwidth becomes very high, and even more so for untethered brain implants that require wireless transmission of data. We propose an energy

efficient spike data extraction solution for high-density electrode arrays, capable of reducing the data to be transferred by over 85%. We combine temporal and spatial spike data analysis with low implementation complexity, where amplitude thresholds are used to detect spikes and the spatial location of the electrodes is used to extract potentially useful sub-threshold data on neighboring electrodes. We tested our method against a state-of-the-art spike detection algorithm, with prohibitively high implementation complexity, and found that the majority of spikes are extracted reliably. We obtain further improved quality results when ignoring very small spikes below 30% of the voltage thresholds, resulting in 91% accuracy. Our approach uses digital logic and is therefore scalable with increasing number of electrodes.

4.6.2 Main Contributions

In this work an application is developed that significantly reduces the transmission energy for wireless read-out in high density electrode arrays. In relation to the system scenario design methodology, the goal of this application is to serve as an example of a data variable dependent application that is applicable for the system scenario design methodology.

4.6.3 Roles of the Authors

This work was done during Yassin's research stay at IMEC, Belgium, as part of the Dual-PhD agreement between NTNU and KULeuven. It was initiated based on ideas and proposals from Catthoor and Kloosterman. Sun and Couto were also involved to provide access to databases and setting up tools, general guidance and help with understanding existing solutions related to thresholding and post-processing steps in neural science. Yassin developed the proposed solution based on his own ideas, which were influenced by feedback, requirements and proposals from Catthoor, Kloosterman, Sun and Couto. Kjeldsberg and Van Helleputte were involved later in the work process to give feedback and guidance on the proposed solutions and results. All co-authors provided Yassin with overall guidance and ideas, including help and feedback with the paper writing process.

CHAPTER

5

CONCLUSIONS AND FUTURE WORK

5.1 Conclusions

The main scope of this dissertation is to develop techniques for system scenario prediction and switching for data variable dependent applications.

We have developed a generic framework model that enables the use of a system scenarios approach through an application monitoring unit (AMU) and a platform adaption manager (PAM) for single-core and multi-core architectures. This framework is implemented on an ARM Cortex M3 based microcontroller from Silicon Labs with only DFS as available system knob. The framework is later ported to a ARM Cortex M4 based microcontroller from Atmel, and extended to combine RTH with DVFS. Techniques for scenario prediction and switching is developed for the H264 encoder control structure application from MediaBench II, where exploding data variable dependencies are contained within RTS parameter ranges instead of monitoring each parameter value separately. Efficient inline functions are used to monitor and predict upcoming scenarios in the application through LUTs, containing scenario parameters obtained through profiling.

Our techniques for scenario prediction and switching are later improved with simple loop transformations, resulting in negligible energy overhead from the scenario mechanisms. A 500 second video sequence is used together with 500 seconds of bandwidth measurements on three different wireless networks to generate three streams of inputs to our H264 application. Our developed scenario mechanisms are validated on these three streams, resulting in upto 49% improvement in the energy consumption compared to only using the widely used RTH technique based on worst case performance analysis.

We extend the widely used GEM5 simulator in order to add more system knobs to our system scenario framework. A novel, non-intrusive application controlled DVFS management mechanism is developed for GEM5 SE mode, with custom DVFS pseudo-instruction in the ARM instruction-set architecture. In addition, we develop a novel architecture independent energy model based on engineering friendly classification of different workload classes. Power measurements from real HW are used together with the GEM5 output statistics to estimate the total energy consumption of the simulated system. After correction of systematic errors, our GEM5 simulator extension and energy model framework results deviated less than 4% compared to the results from our ARM Cortex M4 board from Atmel. Our H264 is simulated with six available DVFS states and compared to the two DVFS state version, resulting in up to 39% improvement in the energy consumption when enough dynamism is present.

Finally, a novel data dependent application is developed, which significantly reduces the amount of data to be transmitted from high-density electrode arrays. These arrays are used to study the behavior of neural cells and networks through extracellular monitoring of the action potentials in animals' brains *in vivo*. Our system scenario techniques may be applied to this application for future improvements, because it's behavior is very dynamic, with complex data dependencies.

5.2 Future Work

Future research to improve the current work can focus on further developing the system scenario methodology for multi-core architectures. Our multi-core FSS framework can be developed and implemented on our GEM5 simulator extension, where the AMU and PAM mechanisms are running in parallel with the application core. When implementing our AMU and PAM

mechanisms as parallel HW, we must consider to include proper synchronization mechanisms between the application and our scenario predictor and switching mechanisms. Additional handshaking is required between processor cores in parallel homogeneous and heterogeneous multi-core applications.

Additional improvements to our FSS framework could be considered such as the inclusion of heterogeneous cores. These cores could be HW accelerators, DSPs and GPUs. Alternatively we could use (re)configurable HW such as small FPGAs for our AMU and PAM mechanisms. Such extensions would allow the calibration step of the scenario methodology to trigger a reconfiguration of the prediction and switching mechanisms at run-time. If the main application is executing multiple sub-applications at different stages of its lifetime, then such calibration may be useful. Each sub-application may require different scenario sets and system knobs to be used, and a re-configuration of the prediction and switching mechanisms is required if there is limited space for all the different prediction and switching mechanisms to co-exist on the platform. This concept is based on the idea of partial reconfiguration of the scenario mechanisms.

Our DVFS implementation in GEM5 currently only supports simple in-order processor models. Another direction for future work is to extend our DVFS implementation to support timing and detailed processor models. This requires additional extensions to the pipelines of the generic or specific processor model in GEM5. Heterogeneous multi-core support does not exist in our current version of GEM5. By implementing heterogeneous multi-core support we can update our DVFS policies to handle different cores with DVFS either separately or though an oracle mechanism. Then we can use this extension to implement different scenario mechanisms in each core through a distributed network of AMU monitors in a scenario hierarchy.

Furthermore, our system scenario techniques can be applied to our data dependent neural science application based on the suggested implementations in this thesis to improve the algorithm, and enable dynamic tuning of the thresholding parameters. The application can easily be implemented with our FSS framework, and simulated on our GEM5 platform with DVFS functionality in order to further reduce the platforms energy consumption.

BIBLIOGRAPHY

[1] Iperf. Networking with iperf. [Online]. Available: http://openmaniak.com/iperf.php. Accessed January 5, 2016.

[2] Hervé Abdi and Lynne J. Williams. Principal component analysis. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(4):433–459, 2010.

[3] Prashant Agrawal. *Application Partitioning and Architecture Instantiation along with Technology Exploration for MPSoCs*. PhD thesis, Katholieke Universiteit Leuven (KULeuven), 2015.

[4] Muhammad Aleem, Muhammad A. Islam, and Muhammad A. Iqbal. A comparative study of heterogeneous processor simulators. *International Journal of Computer Applications*, 148(12), 2016. Copyright - Copyright Foundation of Computer Science 2016; Last updated - 2016-08-21.

[5] M. Alioto. Ultra-low power vlsi circuit design demystified and explained: A tutorial. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 59(1):3–29, Jan 2012.

[6] Oskar Andersson, Babak Mohammadi, and Joachim Neves Rodrigues. A wide-operating range standard-cell based memory in 28nm fd-soi. In *1st Intl. Workshop on Emerging Memory Solutions, DATE Conference 2016, Dresden, Germany*, 2016.

[7] A. G. Andreou, K. A. Boahen, P. O. Pouliquen, A. Pavasovic, R. E. Jenkins, and K. Strohbehn. Current-mode subthreshold mos circuits for analog vlsi neural systems. *IEEE Transactions on Neural Networks*, 2(2):205–213, Mar 1991.

[8] Atmel. Sam4l xplained pro user guide, 2014. http://www.atmel.com/Images/Atmel-42074-SAM4L-Xplained-Pro_User-Guide.pdf on March 7, 2014.

[9] T. Austin, E. Larson, and D. Ernst. Simplescalar: an infrastructure for computer system modeling. *Computer*, 35(2):59–67, Feb 2002.

[10] Muhammad Ali Awan and Stefan M. Petters. Race-to-halt energy saving strategies. *Jour. of Systems Architecture*, 60(10):796 – 815, 2014.

[11] Emilia Biffi et al. Development and validation of a spike detection and classification algorithm aimed at implementation on hardware devices. *Computational intelligence and neuroscience*, 2010:15, 2010. Article ID 659050.

[12] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, August 2011.

[13] N.L. Binkert, R.G. Dreslinski, L.R. Hsu, K.T. Lim, A.G. Saidi, and S.K. Reinhardt. The m5 simulator: Modeling networked systems. *IEEE Micro*, 26(4):52–60, July 2006.

[14] David Brooks, Vivek Tiwari, and Margaret Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. *SIGARCH Comput. Archit. News*, 28(2):83–94, May 2000.

[15] T. E. Carlson, W. Heirmant, and L. Eeckhout. Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation. In *2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pages 1–12, Nov 2011.

[16] Francky Catthoor, Eddy de Greef, and Sven Suytack. *Custom Memory Management Methodology: Exploration of Memory Organisation for Embedded Multimedia System Design.* Kluwer Academic Publishers, Norwell, MA, USA, 1998.

[17] Hsiao-Lung Chan et al. Detection of neuronal spikes using an adaptive threshold based on the max–min spread sorting method. *Journal of Neuroscience Methods*, 172(1):112–121, 2008.

[18] Tony F. Chan, Gene H. Golub, and Randall J. LeVeque. Algorithms for computing the sample variance: Analysis and recommendations. *The American Statistician*, 37(3):242–247, 1983.

[19] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen. Low-power cmos digital design. *IEEE Journal of Solid-State Circuits*, 27(4):473–484, Apr 1992.

[20] Guillaume Charvet et al. *BioMEA*^{TM}: A versatile high-density 3d microelectrode array system using integrated electronics. *Biosensors and Bioelectronics*, 25(8):1889 – 1896, 2010.

[21] Tien-Fu Chen and Jean-Loup Baer. Effective hardware-based data prefetching for high-performance processors. *Computers, IEEE Transactions on*, 44(5):609–623, May 1995.

[22] Kihwan Choi, R. Soma, and M. Pedram. Fine-grained dynamic voltage and frequency scaling for precise energy and performance tradeoff based on the ratio of off-chip access to on-chip computation times. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 24(1):18–28, Jan 2005.

[23] E. Dahlman, P. Beming, J. Knutsson, F. Ovesjo, M. Persson, and C. Roobol. Wcdma-the radio interface for future mobile multimedia communications. *Vehicular Technology, IEEE Transactions on*, 47(4):1105–1118, Nov 1998.

[24] M. Damavandpeyma, S. Stuijk, T. Basten, M. Geilen, and H. Corporaal. Throughput-constrained dvfs for scenario-aware dataflow graphs. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2013 IEEE 19th*, pages 175–184, April 2013.

[25] Bruce Powel Douglass. *Real time UML: advances in the UML for real-time systems*. Addison-Wesley Professional, 2004.

[26] Jelena Dragas et al. An unsupervised method for on-chip neural spike detection in multi-electrode recording systems. In *2013 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 2535–2538. IEEE, 2013.

[27] S. Kaxiras et al. *Computer Architecture Techniques for Power-Efficiency*. Morgan and Claypool Publishers, 1st edition, 2008.

[28] Tom Fawcett. An introduction to roc analysis. *Pattern Recognition Letters*, 27(8):861 – 874, 2006. ROC Analysis in Pattern Recognition.

[29] Iason Filippopoulos, Francky Catthoor, and Per Gunnar Kjeldsberg. Exploration of energy efficient memory organisations for dynamic multimedia applications using system scenarios. *Design Automation for Embedded Systems*, 17(3):669–692, Sep 2013.

[30] Harleen K Flora, Xiaofeng Wang, and Swati V Chande. An investigation on the characteristics of mobile applications: A survey study. *International Journal of Information Technology and Computer Science (IJITCS)*, 6(11):21, 2014.

[31] M. Flynn and P. Hung. Microprocessor design issues: thoughts on the road ahead. *IEEE Micro*, 25(3):16–31, May 2005.

[32] Jeremy Fowers, Greg Brown, Patrick Cooke, and Greg Stitt. A performance and energy comparison of fpgas, gpus, and multicores for sliding-window applications. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA '12, pages 47–56, New York, NY, USA, 2012. ACM.

[33] Antônio Augusto Fröhlich. A comprehensive approach to power management in embedded systems. *International Journal of Distributed Sensor Networks*, 7(1):807091, 2011.

[34] John W. C. Fu, Janak H. Patel, and Bob L. Janssens. Stride directed prefetching in scalar processors. In *Proceedings of the 25th Annual International Symposium on Microarchitecture*, MICRO 25, pages 102–110, Los Alamitos, CA, USA, 1992. IEEE Computer Society Press.

[35] T. Gemmeke, M. M. Sabry, J. Stuijt, P. Raghavan, F. Catthoor, and D. Atienza. Resolving the memory bottleneck for single supply near-threshold computing. In *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1–6, March 2014.

[36] Hassan Ghasemzadeh and Roozbeh Jafari. Ultra low-power signal processing in wearable monitoring systems: A tiered screening architecture with optimal bit resolution. *ACM Transactions on Embedded Computing Systems (TECS)*, 13(1):9, 2013.

[37] Stefan Valentin Gheorghita. *Dealing with dynamism in embedded system design: application scenarios*. PhD thesis, Technische Universiteit Eindhoven, 2007.

[38] Stefan Valentin Gheorghita, Twan Basten, and Henk Corporaal. Scenario selection and prediction for dvs-aware scheduling of multimedia applications. *Signal Processing Systems*, 50(2):137–161, 2008.

[39] Stefan Valentin Gheorghita, Martin Palkovic, Juan Hamers, Arnout Vandecappelle, Stelios Mamagkakis, Twan Basten, Lieven Eeckhout, Henk Corporaal, Francky Catthoor, Frederik Vandeputte, and Koen De Bosschere. System-scenario-based design of dynamic embedded systems. *ACM Trans. Design Autom. Electr. Syst.*, 14(1), 2009.

[40] Stefan Valentin Gheorghita, Sander Stuijk, Twan Basten, and Henk Corporaal. Automatic scenario detection for improved wcet estimation. In *Proceedings of the 42Nd Annual Design Automation Conference*, DAC '05, pages 101–104, New York, USA, June 2005. ACM.

[41] R. Gonzalez, B. M. Gordon, and M. A. Horowitz. Supply and threshold voltage scaling for low power cmos. *IEEE Journal of Solid-State Circuits*, 32(8):1210–1216, Aug 1997.

[42] Marius Grannæs. *Reducing Memory Latency by Improving Resource Utilization*. PhD thesis, Norwegian University of Science and Technology, Trondheim, June 2010.

[43] Marius Grannæs, Magnus Jahre, and Lasse Natvig. Storage efficient hardware prefetching using delta-correlating prediction tables. *J. Instruction-Level Parallelism*, 13:1–16, 2011.

[44] Daniel Halperin et al. Demystifying 802.11 n power consumption. In *Proceedings of the 2010 international conference on Power aware computing and systems*, page 1, 2010.

[45] E. Hammari, F. Catthoor, P.G. Kjeldsberg, J. Huisken, K. Tsakalis, and L. Iasemidis. Identifying data-dependent system scenarios in a dynamic embedded system. *The International Conference on Engineering of Reconfigurable Systems and Algorithms, ERSA'12, Las Vegas, USA*, July 2012.

[46] P. Haririan and A. Garcia-Ortiz. Non-intrusive dvfs emulation in gem5 with application to self-aware architectures. In *2014 9th International*

Symposium on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC), pages 1–7, May 2014.

[47] John L. Hennessy and David A. Patterson. *Computer Architecture, Fifth Edition: A Quantitative Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 5th edition, 2011.

[48] O.I. Hillestad. *Evaluating and enhancing the performance of IP-based streaming media services and applications*. PhD thesis, Norwegian university of science and technology, 2007. Doctoral thesis.

[49] M. Horowitz, T. Indermaur, and R. Gonzalez. Low-power digital design. In *Low Power Electronics, 1994. Digest of Technical Papers., IEEE Symposium*, pages 8–11, Oct 1994.

[50] Mugurel Theodor Ionita. *Scenario-based system architecting: a systematic approach to developing future-proof system architectures*. PhD thesis, Technische Universiteit Eindhoven, 2005.

[51] Sandy Irani, Sandeep Shukla, and Rajesh Gupta. Algorithms for power savings. *ACM Trans. Algorithms*, 3(4), November 2007.

[52] Shantanu P Jadhav, Caleb Kemere, P Walter German, and Loren M Frank. Awake hippocampal sharp-wave ripples support spatial memory. *Science*, 336(6087):1454–1458, 2012.

[53] Shantanu P Jadhav, Caleb Kemere, P Walter German, Loren M Frank, and J Macarthur. Supporting online material for awake hippocampal sharp-wave ripples support spatial memory materials and methods data were collected using the nspike data acquisition system (11) som text real-time swr disruption during behavior using vhc stimulation. 2012.

[54] Magnus Jahre. Single-ISA Heterogeneous MAny-core Computer (SHMAC) Project Plan, 2013.

[55] S. Jain, S. Khare, S. Yada, V. Ambili, P. Salihundam, S. Ramani, S. Muthukumar, M. Srinivasan, A. Kumar, S. K. Gb, R. Ramanarayanan, V. Erraguntla, J. Howard, S. Vangal, S. Dighe, G. Ruhl, P. Aseron, H. Wilson, N. Borkar, V. De, and S. Borkar. A 280mv-to-1.2v wide-operating-range ia-32 processor in 32nm cmos. In *2012 IEEE International Solid-State Circuits Conference*, pages 66–68, Feb 2012.

[56] Norman P. Jouppi. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. In *Proceedings of the 17th Annual International Symposium on Computer Architecture*, ISCA '90, pages 364–373, New York, NY, USA, 1990. ACM.

[57] Awais M Kamboh et al. Analysis of lifting and b-spline dwt implementations for implantable neuroprosthetics. *Journal of Signal Processing Systems*, 52(3):249–261, 2008.

[58] Fabian Kloosterman, Stuart P Layton, Zhe Chen, and Matthew A Wilson. Bayesian decoding using unsorted spikes in the rat hippocampus. *Journal of neurophysiology*, 111(1):217–227, 2014.

[59] E. Koutsos et al. A 1.5 μw neo-based spike detector with adaptive-threshold for calibration-free multichannel neural interfaces. In *2013 IEEE International Symposium on Circuits and Systems (ISCAS2013)*, pages 1922–1925, May 2013.

[60] C. Lee, M. Potkonjak, and W.H. Mangione-Smith. Mediabench: a tool for evaluating and synthesizing multimedia and communications systems. In *Microarchitecture, 1997. Proceedings., Thirtieth Annual IEEE/ACM International Symposium on*, pages 330–335, Dec 1997.

[61] Y.-H. Lee, K.P. Reddy, and C.M. Krishna. Scheduling techniques for reducing leakage power in hard real-time systems. In *Real-Time Systems, 2003. Proc. on 15th Euromicro Conference*, pages 105–112, July 2003.

[62] Sheng Li, Jung Ahn, J. B. Brockman, and N. P. Jouppi. Mcpat 1.0: An integrated power, area, and timing modeling framework for multicore architecture, 2009. HP Labs Tech. Rep. HPL-2009-206.

[63] Sheng Li, Jung Ho Ahn, Richard D. Strong, Jay B. Brockman, Dean M. Tullsen, and Norman P. Jouppi. Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Proceedings of the 42Nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 42, pages 469–480, New York, NY, USA, 2009. ACM.

[64] C. M. Lopez, S. Mitra, J. Putzeys, B. Raducanu, M. Ballini, A. Andrei, S. Severi, M. Welkenhuysen, C. Van Hoof, S. Musa, and R. F. Yazicioglu. 22.7 a 966-electrode neural probe with 384 configurable channels in 0.13 μm soi cmos. In *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, pages 392–393, Jan 2016.

[65] Z. Ma. *Systematic Methodology for Real-Time Cost-Effective Mapping of Dynamic Concurrent Task-Based Systems on Heterogeneous Platforms*. Springer, 2007.

[66] P. Marchal, C. Wong, A. Prayati, N. Cossement, F. Catthoor, R. Lauwereins, D. Verkest, and H. De Man. *Dynamic Memory Oriented Transformations in the MPEG4 IM1-Player on a Low Power Platform*, pages 40–50. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[67] Pierre Marchal, Chun Wong, Aggeliki Prayati, Nathalie Cossement, Francky Catthoor, Rudy Lauwereins, Diederik Verkest, and Hugo De Man. Dynamic memory oriented transformations in the mpeg4 im1-player on a low power platform. In *Proceedings of the First International Workshop on Power-Aware Computer Systems-Revised Papers*, PACS '00, pages 40–50, London, UK, UK, 2001. Springer-Verlag.

[68] M. Marinoni, M. Bambagini, F. Prosperi, F. Esposito, G. Franchino, L. Santinelli, and G. Buttazzo. Platform-aware bandwidth-oriented energy management algorithm for real-time embedded systems. In *ETFA2011*, pages 1–8, Sept 2011.

[69] E. J. McLellan and D. A. Webb. The alpha 21264 microprocessor architecture. In *Proceedings of the International Conference on Computer Design*, ICCD '98, pages 90–, Washington, DC, USA, 1998. IEEE Computer Society.

[70] MediaBench. Mediabench consortium, 2014. http://euler.slu.edu/ fritts/mediabench/ on March 11, 2015.

[71] Antti P. Miettinen and Vesa Hirvisalo. Energy-efficient parallel software for mobile hand-held devices. In *Proceedings of the First USENIX Conference on Hot Topics in Parallelism*, HotPar'09, pages 12–12, Berkeley, CA, USA, 2009. USENIX Association.

[72] Sparsh Mittal. A survey of techniques for improving energy efficiency in embedded computing systems. *International Journal of Computer Aided Engineering and Technology*, 6(4):440–459, 2014.

[73] Shivajit Mohapatra, Radu Cornea, Nikil Dutt, Alex Nicolau, and Nalini Venkatasubramanian. Integrated power management for video streaming to mobile handheld devices. In *Proceedings of the Eleventh ACM International Conference on Multimedia*, MULTIMEDIA '03, pages 582–591, New York, NY, USA, 2003. ACM.

[74] S. Mu, C. Wang, M. Liu, D. Li, M. Zhu, X. Chen, X. Xie, and Y. Deng. Evaluating the potential of graphics processors for high performance embedded computing. In *2011 Design, Automation Test in Europe*, pages 1–6, March 2011.

[75] Jens-Oliver Muthmann et al. Spike detection for large neural populations using high density multielectrode arrays. *Frontiers in neuroinformatics*, 9, 2015.

[76] L. Nagy, J. Koscelánsky, and V. Stopjaková. Design of a globally asynchronous locally synchronous digital system. In *2014 IEEE 12th IEEE International Conference on Emerging eLearning Technologies and Applications (ICETA)*, pages 529–533, Dec 2014.

[77] National Instruments. Ni mydaq measurement board, 2015. http://www.ni.com/mydaq on December 7, 2015.

[78] Kyle J. Nesbit and James E. Smith. Data cache prefetching using a global history buffer. In *Proceedings of the 10th International Symposium on High Performance Computer Architecture*, HPCA '04, pages 96–, Washington, DC, USA, 2004. IEEE Computer Society.

[79] Linwei Niu and Gang Quan. Reducing both dynamic and leakage energy consumption for hard real-time systems. In *Proceedings of the 2004 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, CASES '04, pages 140–148, New York, NY, USA, 2004. ACM.

[80] Pouya Ostovari, Jie Wu, and Abdallah Khreishah. *Network Coding Techniques for Wireless and Sensor Networks*, pages 129–162. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[81] Karim G Oweiss et al. A scalable wavelet transform vlsi architecture for real-time signal processing in high-density intra-cortical implants. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 54(6):1266–1278, 2007.

[82] M. Palkovic, F. Catthoor, and H. Corporaal. Dealing with variable trip count loops in system level exploration. In *Proc. of the 4th Workshop on Optimizations for DSP and Embedded Systems*, pages 19–28, March 2006.

[83] JoAnn M Paul, Donald E Thomas, and Alex Bobrek. Scenario-oriented design for single-chip heterogeneous multiprocessors. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 14(8):868–880, 2006.

[84] J. Power, J. Hestness, M. S. Orr, M. D. Hill, and D. A. Wood. gem5-gpu: A heterogeneous cpu-gpu simulator. *IEEE Computer Architecture Letters*, 14(1):34–36, Jan 2015.

[85] David Martin Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. 2011.

[86] Rodrigo Quian Quiroga. Spike sorting. *Current Biology*, 22(2):R45 – R46, 2012.

[87] Jan M Rabaey and Massoud Pedram. *Low power design methodologies*, volume 336. The Springer International Series in Engineering and Computer Science, 1996.

[88] B. C. Raducanu, R. F. Yazicioglu, C. M. Lopez, M. Ballini, J. Putzeys, S. Wang, A. Andrei, M. Welkenhuysen, N. van Helleputte, S. Musa, R. Puers, F. Kloosterman, C. van Hoof, and S. Mitra. Time multiplexed active neural probe with 678 parallel recording sites. In *2016 46th European Solid-State Device Research Conference (ESSDERC)*, pages 385–388, Sept 2016.

[89] Jitendra Kumar Rai, Atul Negi, and Rajeev Wankar. Machine learning based performance prediction for multi-core simulation. In *Proceedings of the 5th International Conference on Multi-Disciplinary Trends in Artificial Intelligence*, MIWAI'11, pages 236–247, Berlin, Heidelberg, 2011. Springer-Verlag.

[90] A. Ramalingam, S. V. Kodakara, A. Devgan, and D. Z. Pan. Robust analytical gate delay modeling for low voltage circuits. In *Asia and South Pacific Conference on Design Automation, 2006.*, pages 6 pp.–, Jan 2006.

[91] G.N. Rao, R.S.V. Prasad, D.J. Chandra, and S. Narayanan. Real-time software implementation of h.264 baseline profile video encoder for mobile and handheld devices. In *2006 IEEE International Conference on Acoustics, Speech and Signal Processing. ICASSP 2006 Proc.*, volume 5, May 2006.

[92] Hernan Gonzalo Rey, Carlos Pedreira, and Rodrigo Quian Quiroga. Past, present and future of spike sorting techniques. *Brain Research Bulletin*, 119, Part B:106 – 117, 2015. Advances in electrophysiological data analysis.

[93] Alberto Rodriguez-Perez et al. A low-power programmable neural spike detection channel with embedded calibration and data compression. *IEEE transactions on biomedical circuits and systems*, 6(2):87–100, 2012.

[94] Cyrille Rossant, Shabnam N Kadir, Dan FM Goodman, John Schulman, Maximilian LD Hunter, Aman B Saleem, Andres Grosmark, Mariano Belluscio, George H Denfield, Alexander S Ecker, et al. Spike sorting for large, dense electrode arrays. Technical report, Nature Publishing Group, 2016.

[95] Stefania Sesia, Issam Toufik, and Matthew Baker. *LTE - The UMTS Long Term Evolution*, pages i–xxxv. John Wiley & Sons, Ltd, 2009.

[96] Youngsoo Shin, Kiyoung Choi, and Takayasu Sakurai. Power optimization of real-time embedded systems on variable speed processors. In *Proceedings of the 2000 IEEE/ACM International Conference on Computer-aided Design*, ICCAD '00, pages 365–368, Piscataway, NJ, USA, 2000. IEEE Press.

[97] Silicon Labs. User manual starter kit efm32gg-stk3700, 2012. http://www.produktinfo.conrad.com/datenblaetter/450000-474999/467735-an-01-en-User_Manual_GIANT_GECKO_STARTER_KIT.pdf on January 18, 2014.

[98] Alan Jay Smith. Cache memories. *ACM Comput. Surv.*, 14(3):473–530, September 1982.

[99] V. Spiliopoulos, A. Bagdia, A. Hansson, P. Aldworth, and S. Kaxiras. Introducing dvfs-management in a full-system simulator. In *2013 IEEE 21st International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 535–545, Aug 2013.

[100] Micha E Spira et al. Multi-electrode array technologies for neuroscience and cardiology. *Nature nanotechnology*, 8(2):83–94, 2013.

[101] Eran Stark and Moshe Abeles. Predicting movement from multiunit activity. *Journal of Neuroscience*, 27(31):8387–8394, 2007.

[102] Stefan Steinke, Nils Grunwald, Lars Wehmeyer, Rajeshwari Banakar, M. Balakrishnan, and Peter Marwedel. Reducing energy consumption by dynamic copying of instructions onto onchip memory. In *Proceedings of the 15th International Symposium on System Synthesis*, ISSS '02, pages 213–218, New York, NY, USA, 2002. ACM.

[103] STM. Stm32f4 datasheet, 2017. http://www.st.com/content/ccc/resource/technical/document/datasheet/ef/92/76/6d/bb/c2/4f/f7/DM00037051.pdf/files/DM00037051.pdf/jcr:content/translations/en.DM00037051.pdf on June 28, 2017.

[104] Lukasz Strozek and David Brooks. Efficient architectures through application clustering and architectural heterogeneity. In *Proceedings of the 2006 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, CASES '06, pages 190–200, New York, NY, USA, 2006. ACM.

[105] Shuwei Sun, Dong Wang, and Shuming Chen. A highly efficient parallel algorithm for h.264 encoder based on macro-block region partition. In Ronald Perrott, BarbaraM. Chapman, Jaspal Subhlok, RodrigoFernandes de Mello, and LaurenceT. Yang, editors, *High Performance Computing and Communications*, volume 4782 of *Lecture Notes in Computer Science*, pages 577–585. Springer Berlin Heidelberg, 2007.

[106] Nicholas V Swindale et al. Spike detection methods for polytrodes and high density microelectrode arrays. *Journal of computational neuroscience*, 38(2):249–261, 2015.

[107] T. Torfs, A. A. A. Aarts, M. A. Erismis, J. Aslam, R. F. Yazicioglu, K. Seidl, S. Herwik, I. Ulbert, B. Dombovari, R. Fiath, B. P. Kerekes, R. Puers, O. Paul, P. Ruther, C. Van Hoof, and H. P. Neves. Two-dimensional multi-channel neural probes with electronic depth control. *IEEE Transactions on Biomedical Circuits and Systems*, 5(5):403–412, Oct 2011.

[108] L. Traver et al. Adaptive-threshold neural spike detection by noise-envelope tracking. *Electronics Letters*, 43(24):1333–1335, Nov 2007.

[109] Y. Y. Tsai and C. H. Chen. Energy-efficient trace reuse cache for embedded processors. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 19(9):1681–1694, Sept 2011.

[110] Rafael Ubal, Byunghyun Jang, Perhaad Mistry, Dana Schaa, and David Kaeli. Multi2Sim: A Simulation Framework for CPU-GPU Computing . In *Proc. of the 21st International Conference on Parallel Architectures and Compilation Techniques*, Sep. 2012.

[111] P. Van Dyke and B.S. Rai. Apparatus and method for resizing an image, June 8 2010. US Patent 7,733,405.

[112] Steven P. VanderWiel and David J. Lilja. When caches aren't enough: Data prefetching techniques. *Computer*, 30(7):23–30, July 1997.

[113] H. J. M. Veendrick. Short-circuit dissipation of static cmos circuitry and its impact on the design of buffer circuits. *IEEE Journal of Solid-State Circuits*, 19(4):468–473, Aug 1984.

[114] Weixun Wang, Prabhat Mishra, and Sanjay Ranka. Dynamic cache reconfiguration and partitioning for energy optimization in real-time multi-core systems. In *Proceedings of the 48th Design Automation Conference*, DAC '11, pages 948–953, New York, NY, USA, 2011. ACM.

[115] Paul T Watkins et al. Validation of adaptive threshold spike detector for neural recording. In *Engineering in Medicine and Biology Society, 2004. IEMBS'04. 26th Annual International Conference of the IEEE*, volume 2, pages 4079–4082. IEEE, 2004.

[116] Eric W. Weisstein. Variance. From MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/Variance.html.

[117] B. P. Welford. Note on a method for calculating corrected sums of squares and products. *Technometrics*, pages 419–420, 1962.

[118] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra. Overview of the h.264/avc video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):560–576, July 2003.

[119] S. J E Wilton and N.P. Jouppi. Cacti: an enhanced cache access and cycle time model. *IEEE Journal of Solid-State Circuits*, 31(5):677–688, May 1996.

[120] Patrick D Wolf. Thermal considerations for the design of an implanted cortical brain–machine interface (bmi). *Indwelling neural implants: strategies for contending with the in vivo environment*, pages 33–38, 2008.

[121] D. Wu et al. Scenario-based system design with colored petri nets: an application to train control systems. *Software & Systems Modeling*, pages 1–23, 2016.

[122] Yuning Yang et al. Adaptive threshold spike detection using stationary wavelet transform for neural recording implants. In *2010 Biomedical Circuits and Systems Conference (BioCAS)*, pages 9–12. IEEE, 2010.

[123] Y. H. Yassin, F. Catthoor, F. Kloosterman, J. Couto, J.-J. Sun, P. G. Kjeldsberg, and N. V. Helleputte. Algorithm/architecture co-optimisation technique for automatic data reduction of wireless read-out in high-density electrode arrays. *ACM Transactions on Embedded Computing Systems (TECS)*, 2018. In press.

[124] Y. H. Yassin, P. G. Kjeldsberg, A. Perkis, and F. Catthoor. Dynamic hardware management of the h264/avc encoder control structure using a framework for system scenarios. In *2016 Euromicro Conference on Digital System Design (DSD)*, pages 222–229, Aug 2016.

[125] Yahya H. Yassin, Francky Catthoor, Per Gunnar Kjeldsberg, and Andrew Perkis. Techniques for dynamic hardware management of streaming media applications using a framework for system scenarios. *Microprocessors and Microsystems*, 56:157 – 168, 2018.

[126] Y.H. Yassin, P.G. Kjeldsberg, and F. Catthoor. System scenario framework evaluation on efm32 using the h264/avc encoder control structure. In *2015 European Conference on Circuit Theory and Design (ECCTD)*, pages 1–4, Aug 2015.

[127] Chantal Ykman-Couvreur, Philipp A. Hartmann, Gianluca Palermo, Fabien Colas-Bigey, and Laurent San. Run-time resource management based on design space exploration. In *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, CODES+ISSS '12, pages 557–566, New York, USA, October 2012. ACM.

[128] J. Zhang et al. An efficient and compact compressed sensing microsystem for implantable neural recordings. *IEEE Transactions on Biomedical Circuits and Systems*, 8(4):485–496, Aug 2014.

[129] Zhuo Zhao and Ping Liang. A highly efficient parallel algorithm for h.264 video encoder. In *Acoustics, Speech and Signal Processing, 2006.*

ICASSP 2006 Proceedings. 2006 IEEE International Conference on, volume 5, pages V–V, May 2006.

[130] B. Zimmer, Y. Lee, A. Puggelli, J. Kwak, R. Jevtic, B. Keller, S. Bailey, M. Blagojevic, P. F. Chiu, H. P. Le, P. H. Chen, N. Sutardja, R. Avizienis, A. Waterman, B. Richards, P. Flatresse, E. Alon, K. Asanović, and B. Nikolić. A risc-v vector processor with tightly-integrated switched-capacitor dc-dc converters in 28nm fdsoi. In *2015 Symposium on VLSI Circuits (VLSI Circuits)*, pages C316–C317, June 2015.

APPENDIX

A

ADAPTIVE RUN-TIME PREFETCH MANAGER USING SEQUENTIAL ON-ACCESS AND DCPT PREFETCHING

Yahya H. Yassin, Emil Grunt, Eirik Flogard, Håkon Kaurel,
Per Gunnar Kjeldsberg, Francky Catthoor and Lasse Natvig
Technical report
2014

Abstract

Advanced prefetching schemes are able to achieve significant speedup improvements compared to solutions without prefetching. We combine prefetching schemes with different complexities using different adaptive prefetch manager (APM) solutions. Compared to a DCPT prefetching scheme, our best APM solution (DMAP-APM) reduces the static energy consumption by up to 70.9%, with 4.5% relative increase in dynamic energy consumption. Our DMAP-APM solution can adapt to different applications by detecting known sequential memory access patterns, and switching to a dedicated scheme that consumes less static power and energy than an always active standalone advanced scheme, i.e., the DCPT scheme.

A.1 Introduction

In recent years, high end processor performance has increased at a much higher rate than DRAM performance, increasing the importance of memory hierarchy utilization. This results in the Memory Gap, implying that the rate of memory requests to the main memory increase at a much faster rate than DRAM accesses [47]. In applications which contain large amounts of data transfers, the utilization of the memory hierarchy is an important topic with respect to power consumption as well as system performance. A typical example of applications that have large amounts of data transfers are wireless, multimedia, multifunctional and ubiquitous applications [16]. These applications have gained even more attention in recent years due to the popularity of hand-held devices.

According to Hennessy and Patterson [47], programmers desire unlimited amounts of fast memories. Since large memories are usually much slower than small memories, an economical solution to improve the speed of large memories is the usage of a memory hierarchy. In a memory hierarchy, the application's temporal and spatial locality, and architectural trade-offs are exploited in order to improve the memory communication. This has resulted in the use of multiple levels of fast memories, i.e., caches. The main memory latency can be hidden by using caches in a hierarchy [42]. This latency can be reduced even more by prefetching data speculatively [42], [47]. Speculatively prefetching data into the cache lowers the access latency at the cost of higher bandwidth usage.

Prefetching is a speculative technique, leading to a higher probability of increased bandwidth usage. In some cases, data are loaded into the cache unnecessarily, evicting other data from the cache into the main memory. If the evicted data is later needed by the processor we experience the phenomenon cache pollution, which degrades system performance. Therefore, when using prefetching, trade-off considerations have to be made with respect to the system overhead of prefetching (in terms of added complexity, bandwidth, and possible cache pollution), and the added gain of reducing the memory latency.

Another trade-off to be considered is power consumption. An advanced prefetcher scheme will store large amounts of temporary history data per load instruction, which requires additional memory. The amount of additional memory needs to be taken into account since a large part of a system's power consumption is due to data storage and data transfer [16].

In a situation where the application does not have a predictable data

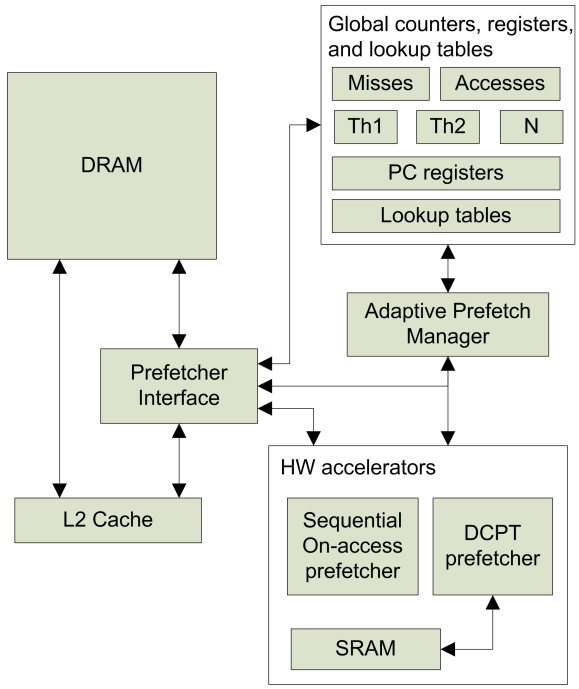


Figure A.1: The adaptive prefetch manager solution in a customized low power general purpose system.

transfer pattern, then the average power consumption most probably increases due to needless prefetches. Using a complex prefetching scheme with large data structures also increases the static power consumption in the required memories. When designing a general purpose low power system, the hardware engineer normally does not have any information about the application that is going to run on the system. Therefore, when an application that does not perform large amounts of data transfers is running, a no prefetching alternative, or a very simple prefetching scheme, is probably the best solution in terms of power consumption. On the other hand, if the application performs large amounts of data transfers prefetching will typically be beneficial. Whether an advanced or a simple prefetcher is needed depends on the complexity of the memory access patterns. With complex memory access patterns, an advanced prefetcher scheme reduces the memory latency and speeds up the processing. In the case where the memory access patterns are more easily predictable, we can reduce the prefetching complexity and save energy by selecting a dedicated prefetcher.

This paper shows how a set of state-of-the-art prefetching schemes with

different levels of complexity can be combined and controlled by a dynamic adaptive prefetch manager (APM) solution. We have studied three different implementations and complexities of our APM.

Our first APM solution monitors the accesses of the L2 cache memory, and activates or changes the prefetching scheme based on the hit rate over a predefined number of accesses. Our second APM solution monitors memory access patterns and adapts a sequential on-access prefetching scheme at the beginning of the application when it detects known patterns. The detected scheme is then fixed for the rest of the application runtime. Our third and best APM is an extension of our second APM solution, and monitors memory access patterns and adapts a sequential on-access prefetching scheme dynamically when it detects known patterns.

Our main contributions are 1) developing a novel priority based APM (DMAP-APM) which dynamically analyzes and detects interleaved memory access patterns and access patterns from memory load instructions at runtime, and 2) evaluating our APM solutions and fixed prefetching scheme solutions through various benchmark programs. To our knowledge, no solution that analyzes interleaved and normal memory access patterns in a prioritized order exist today.

The simplest prefetcher uses the sequential on-access prefetching scheme presented in [98]. It fetches the next cache block when a cache block is accessed. This scheme is used as a starting point in this paper. The Delta Correlation Prediction Table (DCPT) prefetching scheme presented in [42], is also chosen for our case study. In our initial implementation, the prefetching mechanism predicts when a higher degree of prefetching is needed by monitoring the L2 cache hit rate. Hence, aggressive speculative prefetches are avoided by switching to a simpler prefetching scheme when the hit rate is above a certain threshold. In our more advanced APM implementations, we add monitors to track for specific memory access patterns in order to adapt a sequential on-access prefetching scheme to its prefetching distance, degree and stride.

At the end of this paper, we discuss how the APM can be used in a system where all prefetching schemes are implemented as HW accelerators. As illustrated in Figure A.1, these accelerators are controlled by the APM solution, which is able to choose a prefetching scheme according to the memory activity.

The rest of the paper is organized as follows. Related work in prefetching schemes is presented in Section A.2. Section A.3 describes the simulation environment and platform our work is based on. We describe the imple-

mented prefetching schemes and our different APM solutions in Section A.4. Our benchmark, power and energy results are presented and discussed in Section A.5 and A.6, respectively. We conclude our work in Section A.7.

A.2 Related work

Improvements and alternative schemes to sequential on-access prefetching have been proposed, such as Tagged Prefetching (TP) [112], prefetching using Stream Buffers (SB) [56], and Stride Directed Prefetching (SDP) [34].

TP is a simple extension to the sequential on-access prefetching scheme, where the prefetched data is marked with a tag bit for every block in the cache. When this tag bit is set, it indicates that the block is fetched. This improvement creates a domino effect where cache block $b+2$ is fetched when the prefetched block $b+1$ is accessed [112].

The use of stream buffers is an alternative approach. It prefetches cache lines starting at a cache miss address [56]. According to [42], a stream buffer is a structure that holds prefetched data tailored to the type of prefetcher used.

The SDP scheme avoids needless prefetches when the processor's referencing pattern strides through nonconsecutive memory blocks. It uses a table indexed by the load address, where a stride is calculated after a load instruction is encountered two times [42]. SDP prefetches data using an address consisting of the sum of the current address and the stride/delta.

Reference prediction tables (RPT) extends SDP by adding state information in such a way that the scheme can be updated with new delta values when the load pattern changes [21]. Thus, the state information indicates how further prefetching should be generated.

Further enhancements are made with the global history buffer (GHB) scheme, by using a First-In-First-Out (FIFO) buffer containing a history of the last cache misses observed by the memory system [78]. GHB creates opportunities to design more effective prefetching methods such as Program Counter/Delta Correlation Prefetching (PC/DC). The PC/DC scheme stores a table of calculated delta values between consecutive misses in addition to the FIFO buffer [78]. The most recent pair of deltas are used as a prefetch pattern after the computation of the deltas in the delta history table.

The DCPT prefetching scheme [43] is a state-of-the-art prefetching scheme that is based on RPT [21] and PC/DC [78]. In addition to the usage of a

large table, the DCPT scheme uses a longer history like the GHB [78]. Each entry of the DCPT table is indexed by the program counter (PC) of a load instruction. As illustrated in Figure A.2, each entry includes the last miss address, the address of the last issued prefetch, and a circular buffer of delta values controlled by a pointer. The stored delta values are the detected strides in earlier memory accesses made by the load instruction referenced by the PC-address. Therefore, the DCPT table records the access history of all registered load instructions, which are in turn used for prefetching on the occurrence of a cache miss.



Figure A.2: An entry in the DCPT table.

Our APM implementations are based on the idea of a Run-time Resource Management (RRM) framework, such as the RRM framework for embedded heterogeneous multi-core platforms presented by Ykman-Couvreur et. al. [127]. The framework is targeting to support multiple applications running concurrently, where the applications can start and stop at any time. The RRM framework tries to optimize the platform continuously, and seeks an optimal HW configuration dynamically at runtime. In this environment, quality of experience (QoE) requirements and platform resources are scaled dynamically by adjusting clock frequencies and voltages, or by switching off some functions. Hence, the RRM is able to efficiently control the energy/power consumption and heat dissipation of the platform.

Our APM solutions optimize the memory latency through prefetching using an approach similar to the RRM framework presented by Ykman-Couvreur et. al. In addition to differentiating through our focus on data prefetching, we apply a finer granularity level than in their RRM framework. With our APM concepts, our goal is to dynamically detect the memory access patterns and select the best available prefetching scheme at runtime. Hence, we improve the platform's energy efficiency.

A.3 M5 simulation environment, platform architecture and benchmark programs

The prefetcher evaluation is done using the M5 simulator [13], with the Alpha 21264 processor [69]. This processor is superscalar and capable of

doing out-of-order execution and register renaming. Alpha 21264 has two levels of cache, where the size of the first level is 64 kB, and the size of the second level is 1 MB. The cache block size is 64 B. In addition, each cache block in the L2 cache contains a bit that can be controlled by cache block bit functions, e.g., for detection of recently visited cache blocks.

Our prefetch manager mechanism and prefetcher schemes are implemented in C++, using a simple interface for communication with the simulated memory system. This interface is structured in an event-driven manner, and as a consequence the prefetcher responds to events in the simulated hardware.

All required data structures needed by the prefetching schemes are initialized through a single function call provided by the prefetcher interface. An event is triggered when a cache line prefetch is complete, or when a cache line is being accessed. These events are read by our implementation in order to predict future prefetches and evaluate cache access statistics. The interface provides functions for cache line prefetches, functions to check whether or not a cache line is already present in the cache, and get the number of pending prefetches in the Miss Status Holding Register (MSHR) queue. Furthermore, the interface provides basic data structures and functions used to implement our prefetcher schemes. All tests are executed using the M5 configuration listed in Table A.1. The interface functions are listed in Table A.2.

L2 cache size	1 MB
Cache block size	64 B
Memory bus width	64 bits
Memory bus clock	400 MHz
Memory latency	30 ns
Warmup instructions	10 million
Total number of instructions	100 million

Table A.1: M5 system parameters.

We evaluate our prefetcher performance using the SPEC CPU2000 benchmark suite. It contains a set of programs based on actual end user applications, as opposed to benchmark programs built from the ground up for benchmarking purposes. It focuses on evaluating a computer's CPU and memory architecture, covering both floating point and integer performance. The programs listed in Table A.3 are selected because of the variety of data transfer patterns. Some benchmark programs have unpredictable memory

access patterns while other programs have more predictable memory access patterns.

prefetch_init	Called to initialize the prefetcher data structures.
prefetch_access(stat)	Notifies the prefetcher about a cache access.
prefetch_complete(addr)	Notifies the prefetcher about a prefetch load that just completed.
issue_prefetch(addr)	Initiate a prefetch.
[get/set/clear]_prefetch_bit(addr)	Set, get or clear the status of the prefetch bit for an address.
in_cache(addr)	Returns true if addr is in L2 cache.
in_mshr_queue(addr)	Returns true if there is a prefetch request for addr in the MSHR queue.
current_queue_size	Returns the number of queued prefetch requests.

Table A.2: Interface functions.

Name of program	*Type of program / Category*
ammp	Computational Chemistry
applu	Parabolic / Elliptic Partial Differential Equations
apsi	Meteorology: Pollutant Distribution
art110 and art470	Image Recognition / Neural Networks
bzip2_graphic, bzip2_program and bzip2_source	Compression
galgel	Computational Fluid Dynamics
swim	Shallow Water Modeling
twolf	Place and Route Simulator
wupwise	Physics / Quantum Chromodynamics

Table A.3: Our selection of the programs from the SPEC CPU2000 benchmark suite, and their category description.

Each benchmark simulation outputs a set of statistics intended to reflect the different aspects of the system's performance. Every program in CPU2000 outputs the performance in terms of instructions per cycle (IPC),

coverage, accuracy, and speedup. The number of prefetches identified by the prefetcher, and the number of issued prefetches by the cache controller is also recorded for each program. These recordings are used to calculate the harmonic mean of identified vs issued prefetches by using the formula given in Equation A.1, where x_i is the result from program i, and n is the number of programs. If an identified prefetch address is already in the MSHR or prefetch queue, or if the prefetch queue is already full, then the prefetch is dropped. This results in a lower issued prefetch count.

$$H_{\text{avg}} = \frac{n}{\sum \frac{1}{x_i}_{i=1}^{n}} \tag{A.1}$$

When a new feature is implemented in a system, the impact on system performance is compared to the old system in terms of IPCs as shown in Equation A.2. A speedup larger than one indicates that the new feature is improving performance. However, the IPC can be misleading. It reflects the average number of instructions executed per clock cycle. Even if the simulated platform uses a single core processor, the IPC number can exceed 1 by means of utilizing instruction level parallelism (ILP). High IPC values indicate high performance. However, IPC does not necessarily reflect real-world performance, e.g., spinlocks can increase IPC values needlessly.

$$\text{Speedup} = \frac{\text{IPC}_{\text{new}}}{\text{IPC}_{\text{old}}} \tag{A.2}$$

Our prefetcher's accuracy is measured as the ratio of good prefetches to the total number of prefetches as shown in Equation A.3. A good prefetch requires a cache line to be prefetched into the cache before it is referenced by the application. In all other cases, i.e., when the prefetched data is not referenced at all, or when it is not prefetched before it is referenced, it is considered a bad prefetch.

$$\text{Accuracy} = \frac{\text{Good prefetches}}{\text{Good prefetches} + \text{Bad prefetches}} \tag{A.3}$$

If the accuracy is low, the prefetcher is mainly polluting the cache. In addition, the load on the memory system is increased, and thereby also the power consumption. A high accuracy indicates that a prefetcher is prefetching the correct cache lines.

The total impact on the systems performance is not only reflected by good prefetch accuracy and timeliness. If a prefetcher issues too few prefetches, it will not have any real impact on system performance, despite having high accuracy and good timeliness. Therefore, a coverage metric is used

as shown in Equation A.4. It measures the ratio between good prefetches and the number of cache misses occurring when no prefetching is present. Instead of having a metric which compares cache misses with and without prefetching, the coverage metric shows how many cache misses that could occur without prefetching is replaced with good prefetches.

$$\text{Coverage} = \frac{\text{Good prefetches}}{\text{Cache misses without prefetching}} \tag{A.4}$$

The metrics presented in Equation A.1, A.2, A.3 and A.4 are defined by Grannæs [42]. In our case-study, we limit the size of all data structures used for prefetching to approximately 8 kB. This limitation will make our prefetchers more realistic and prevent artificial high scores in our benchmark programs.

A.4 Prefetcher concepts and APM extension

We implemented the sequential on-access prefetcher scheme [98] and DCPT scheme [43] in accordance with the descriptions in Section A.1 and A.2. The DCPT table is implemented as a FIFO ring buffer where each entry represents a structure. Each structure contains the variables for one DCPT table entry. In Table A.4, all variables in this structure are listed with their corresponding data types. The size of each delta and the delta buffer where chosen by experimentation, i.e., where all programs were simulated with different sizes in a brute force manner. The combination which gave the best average speedup was chosen. For our benchmark suite, 32-bit deltas and a delta buffer of 18 elements resulted in the best speedup.

Name of structure element	*Data type*
pc	64-bit unsigned int
last_addr	64-bit unsigned int
last_prefetch	64-bit unsigned int
delta[18]	32-bit signed int
delta_ptr	8-bit signed int

Table A.4: Each element in the structure, representing one row in the DCPT table.

Considering our 8 kB data structure limit, we implemented a DCPT table consisting of 81 entries. A 32 bit wide, and 2048 element 8 kB standard

cell scratch pad memory, i.e., SRAM, is preferred for the DCPT history table because each row in the DCPT table requires 25 memory rows, leaving 92 reserved bytes at the end of the memory for other purposes. In addition, each DCPT table row will consist of 3 spare bytes to ensure fast access to each 32-bit wide memory row. Memories with wider bit widths (i.e., 64 bit) increases the amount of reserved bytes, and memories with smaller bit widths (i.e., 16 bit) require longer time to read out the DCPT table row elements. In our results we evaluate the static power consumption of the DCPT table based on a model for a 32 bit wide, and 2048 element 8 kB, 40 nm standard cell SRAM memory provided by the Interuniversity Micro-Electronics Centre (IMEC) in Belgium [35].

A.4.1 Simple APM based on cache hit rate

Assuming that each prefetching scheme available in the system can be implemented as a HW accelerator, the system can save power by switching to a more power efficient variant as long as the applications real-time constraints are met. All other accelerators are then switched off to save power.

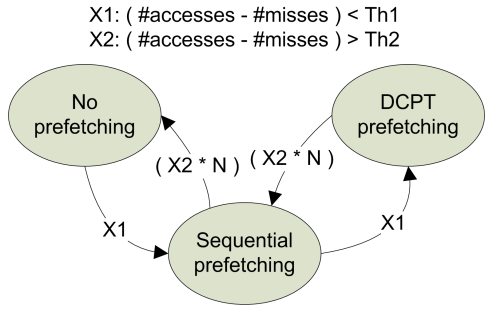


Figure A.3: The simple prefetch manager state machine.

In order to decide what prefetch algorithm to use, the current run-time situation must be predicted. When a change in the access pattern is detected, then the APM switches prefetching scheme, or disables it if prefetching is not required. This idea is inspired by the work of Ykman-Couvreur et. al. [127] as described in Section A.2.

The first simple APM implementation can be viewed as a state machine where the different prefetcher schemes correspond to the domain of states. An example is shown in Figure A.3, where no prefetching, a simple sequential on-access prefetching scheme that fetches the next cache block on a

cache miss, and the DCPT prefetching scheme are controlled by the APM. The APM is implemented in software with the use of a lightweight inline function, i.e., with fast and simple state prediction mechanisms. This function is initialized in a no-prefetching state. Additionally, global counters are used to track memory accesses and cache misses, which are used to calculate hit rates over a predefined number of L2 cache accesses between APM measurements. If the hit rate drops below a certain threshold (Th1), then the APM will change the prefetch state, and turn on or choose a better prefetch scheme. When the hit rate exceeds a certain threshold (Th2) a predefined number of measurements (N) consecutively, then the state machine selects a simpler prefetcher. A third counter is used to keep track of the number of times the hit rate is exceeding Th2.

Th1, Th2 and N are parameters that can easily be implemented as setup registers in order to calibrate the sensitivity of the APM. In our simulated environment, we profiled the benchmark suite by testing different values for Th1, Th2 and N. The combination that gave the best overall result in terms of speedup were chosen as listed in Table A.5.

L2 cache accesses between APM measurements	101
Th1	85
Th2	100
N	100

Table A.5: The APM threshold parameters, and the access interval between measurements that resulted in the best speedup.

The output of the switch is the active prefetch state determining which prefetching scheme to use when the cache is accessed. To avoid a state switch each time the cache is accessed, we implemented two global counters which counts the number of accesses to the cache and the number of misses. When the amount of accesses exceeds a predefined limit, the switch manager function is triggered to determine the next state. The global counters are then re-initialized.

Assume the values in Table A.5. If now ”Sequential prefetching” has just become the current state, we count the number of misses during 101 L2 cache accesses. If X1 is true, i.e., ”101 - #misses $<$ 85”, the APM changes to ”DCPT prefetching”. If X2 is true, i.e., ”101 - #misses $>$ 100”, we stay in ”Sequential prefetching” and repeat the test. If X2 is true N = 100 times the APM changes to ”No prefetching”.

A.4.2 Memory access pattern matching based APM (MAP-APM)

As will be seen in Section A.5 and A.6, our initial approach did not result in significant performance improvement as compared to DCPT. The best Th1, Th2, and N configuration forced the system to use the DCPT scheme most of the time for all benchmark applications.

In order to understand the problem further, all benchmark programs were profiled. The access pattern was recorded to look for sequential patterns with different distance, strides and occurrence in order to specify various alternatives to prefetch distance, degree and stride according to Figure A.4. The sequential prefetcher scheme implemented in our simple APM solution in Figure A.3 corresponds to a sequential prefetcher with distance and stride equal to one, and degree equal to zero (SEQ-1-0-1 in Table A.6).

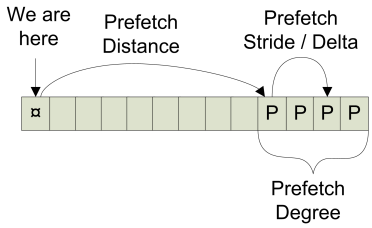


Figure A.4: Prefetch distance, degree and stride.

Name	Distance	Degree	Stride
SEQ-1-0-1	1	0	1
SEQ-1-5-1	1	5	1
SEQ-9-5-9	9	5	9
SEQ-30-5-30	30	5	30
SEQ-1-80-1	1	80	1

Table A.6: Variations of the sequential prefetching scheme.

DIFF	-1	-9	-30
State	0	1	2

Table A.7: MAP-APM lookup table entry.

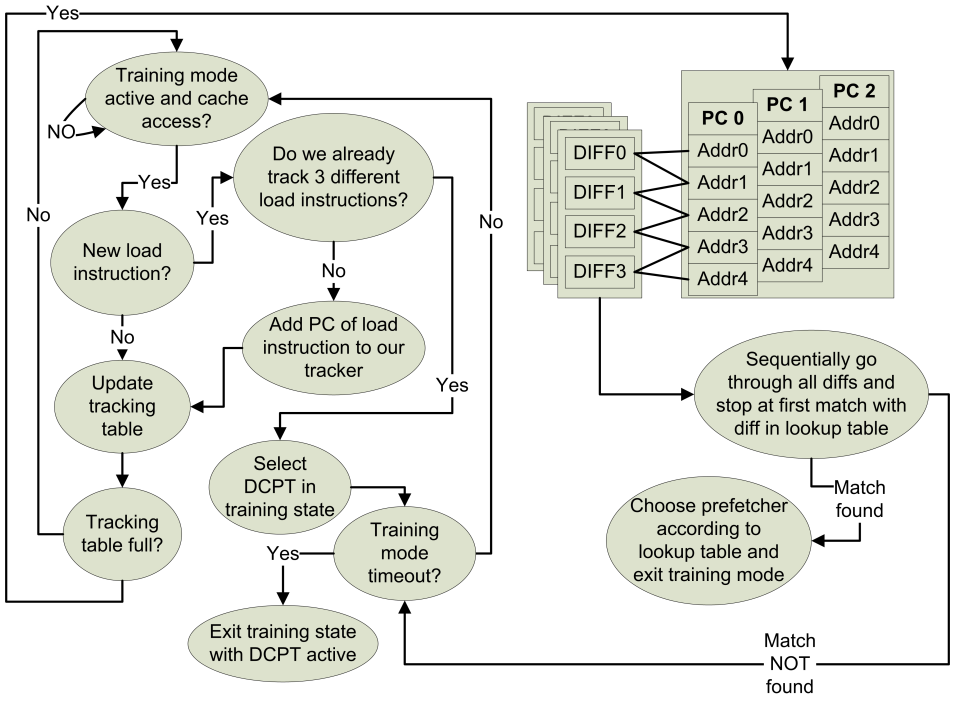


Figure A.5: Memory access pattern matching based APM flow.

Based on our profiled results, we selected a few simple access patterns and added customized parameters to our sequential on-access prefetching scheme. The customized parameters for each access pattern are listed in Table A.6. E.g., the SEQ-9-5-9 setting means that the next address to be fetched is "current address + 9 (distance)", followed by 5 (degree) consecutive prefetches with an address jump of 9 (stride) each time. The sequential-prefetching state of our simple prefetch manager in Figure A.3 is split into multiple states to accommodate for the different sequential access patterns. Unfortunately, this solution gave significant speedup degradation due to the increased complexity. No results from its use is therefore reported in Section A.5 and A.6.

In order to accommodate for additional patterns we instead made a different Memory Access Pattern matching APM (MAP-APM) scheme without the dynamic continuous adaption to changing patterns. A training mode is run when a new application starts. The flow of this prefetch manager is illustrated in Figure A.5. Again based on our profiled results, we selected three different patterns and stored the stride values in a lookup table as

shown in Table A.7. E.g., in State 0 of Table A.7 a DIFF of -1 means that the stride between two consecutive accesses is 1. This case represents a sequential access pattern, where the load instruction increments the address with 1 for each consecutive access.

During the training mode, the MAP-APM scheme monitors the memory access patterns for the known strides of Table A.7. We first identify three different load instructions after a predefined number of memory accesses. The delayed start of the access pattern detection prevents false detection as a result from initialization instructions, i.e., loading of configuration data at the start of an application run. The PC address of the load instructions is stored in a register. For each PC address, five consecutive accesses are recorded in a small array. When five memory addresses are recorded, the MAP-APM calculates the stride (DIFF) between each element in the array. The calculated DIFFs for each PC address are then compared with the values in Table A.7. MAP-APM stops when the first match is found, exits the training mode and uses the corresponding sequential prefetching state for the remainder of the application run. If no known stride is detected for any PC address, MAP-APM selects the DCPT prefetcher as the default prefetching scheme.

In Section A.5 and A.6 the results from use of the MAP-APM prefetching scheme will be presented.

A.4.3 Dynamic MAP-APM (DMAP-APM)

In our benchmark profiling results we observed that some of the dominating access patterns for a group of load instructions were spatial and/or temporal. The way all load instructions were interleaved also affected the detected access pattern significantly. In our MAP-APM implementation the load instructions were monitored individually, where the interleaved access pattern was not considered, i.e., the consecutive loads reflected by the program trace as a result of interleaved load instructions. There was therefore no guarantee that the needed data could be prefetched in time due to the interleaved behavior of all load instructions.

We forced our MAP-APM to restart its detection periodically instead of determining the prefetching scheme only at the start of an application run. This attempt was made to make our MAP-APM adapt dynamically to interleaved access patterns, but it resulted in significant speedup degradation.

The interleaved nature of access patterns were studied for all benchmark tests. Some benchmarks resulted in a very undeterministic access pattern where a sequential on-access prefetching scheme was not feasible. In those situations the DCPT scheme was the preferable solution since it updates the access pattern continuously in its large history table. We improved our monitors to accommodate for the access pattern from interleaved load instructions. Our resulting DMAP-APM flow is shown in Figure A.6. The light green parts of the figure are similar to those in MAP-APM. The blue parts are slightly updated to take the dynamic behavior into account, while the red parts are new to DMAP-APM.

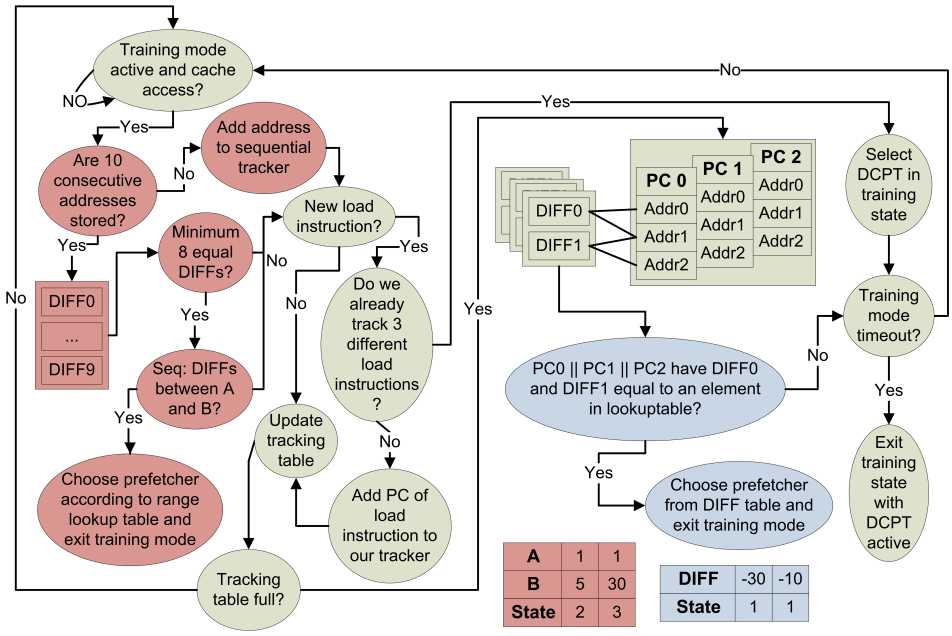


Figure A.6: DMAP-APM flow.

As illustrated in Figure A.6, the DMAP-APM records 10 consecutive memory accesses in an array without monitoring the load instruction that issued the memory access. If the stride from nine addresses in that array are identical and the stride value is between a range A and B in the range lookup table (red table in Figure A.6), then a sequential on-access prefetching scheme is detected. The range lookup table is traversed column by column sequentially, and stops when a match is found. It is therefore a prioritizing lookup table where each column has a higher priority than consecutive columns.

The DMAP-APM exits its training mode if a match is found, and selects the sequential on-access prefetching scheme indicated by the state number. Otherwise it monitors the access pattern by recording three consecutive accesses for three load instructions similar to the MAP-APM scheme. The difference is that DMAP-APM monitors fewer consecutive accesses per load instruction (identified by the PC address), resulting in two strides (DIFFs) per load PC address. Unlike MAP-APM, DMAP-APM detects if DIFF0 and DIFF1 are identical for any of the load instructions (PC0, PC1 or PC2) and if they exist in the DIFF lookup table (this list is also prioritized by lower indices). If a match is found, the corresponding state number switches the prefetching scheme accordingly. If a match is not found, then the DCPT scheme is selected after a timeout. During the DMAP-APM training mode the DCPT scheme is also enabled in training mode. In training mode, the DCPT history tables are updated, but no prefetches are issued. If DMAP-APM fails to detect a sequential access pattern at the end of its training mode, then the DCPT scheme is chosen in active mode with an updated history table. DMAP-APM re-initiates the training sequence after every 10 000 memory read access. The interval of the training sequence was found in a brute force manner, where we experimented with different intervals. 10 000 memory read access intervals gave the best results in terms of speedup.

With our DMAP-APM solution, we open up possibilities to reduce the systems static power and energy significantly by turning off the DCPT history table memory when the DCPT scheme is not active. The reduction in static energy consumption is estimated and discussed in Section A.6.

A.5 Performance results

We tested all our implementations with the SPEC CPU2000 benchmark suite described in Section A.3. The speedup, coverage, IPC and accuracy per benchmark is shown in Figure A.7, A.8, A.9, and A.10 respectively.

The MAP-APM works well when the load instruction patterns at the start of the application runtime dominates the application runtime. Otherwise, this prefetching scheme resulted in significant speedup degradation. The benchmarks used in our research contained dynamically changing access pattern behavior. Therefore, our MAP-APM needs to be adaptive in order to guarantee that the optimal prefetching scheme is chosen throughout the application runtime.

Our DMAP-APM resulted in no speedup improvement. However, it is

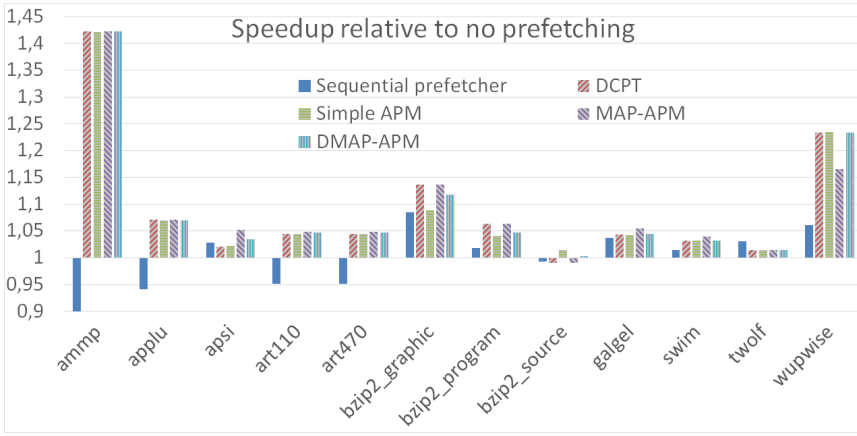


Figure A.7: Speedup per benchmark program relative to a no prefetching solution.

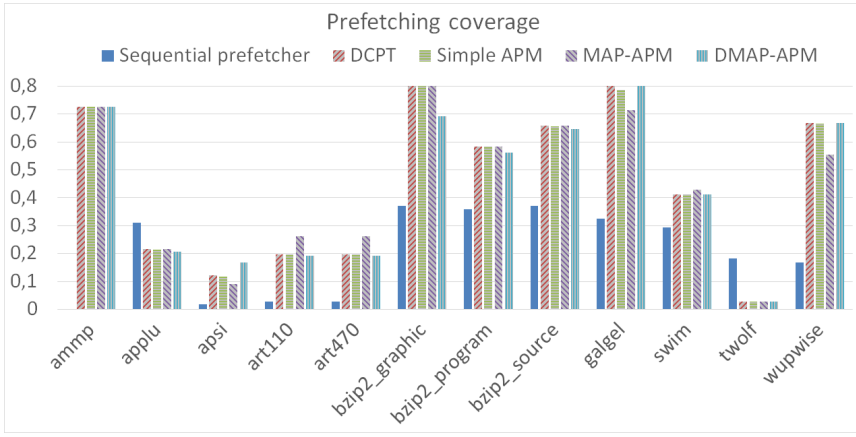


Figure A.8: Coverage per benchmark program.

able to dynamically detect sequential access patterns and switch prefetching scheme with limited speedup degradation (0,02% on average and less than 1,6% in all cases as compared to a standalone DCPT scheme). This is clearly shown in Figure A.7. The DCPT table memory is not used in situations where a sequential access pattern is detected and the prefetching scheme is switched. The sequential on-access prefetching schemes do not require history tables to record the access history.

Our DMAP-APM solution introduces more freedom to power and energy optimizations than the DCPT scheme. This is an advantage in a general

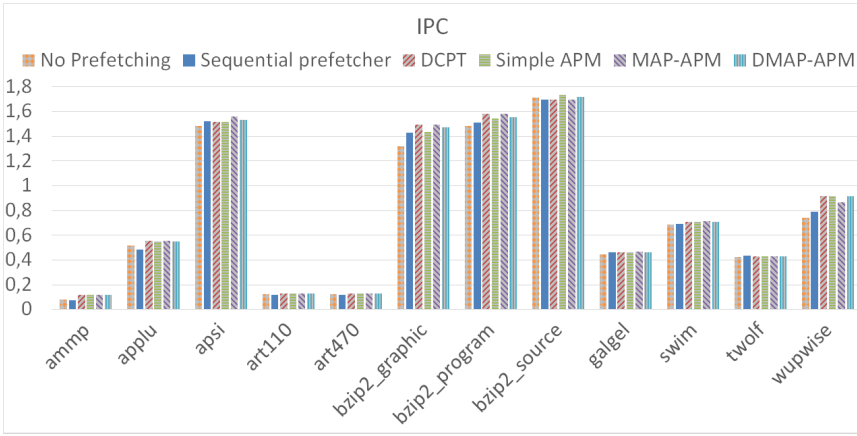


Figure A.9: IPC per benchmark program.

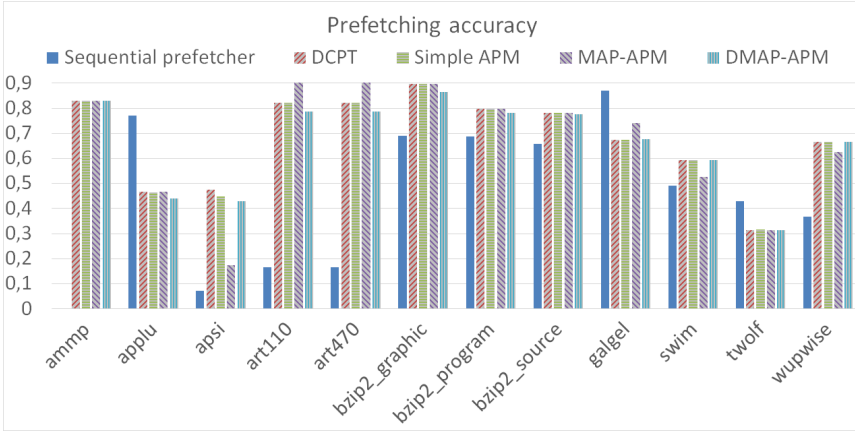


Figure A.10: Accuracy per benchmark program.

purpose platform, which we will discuss in Section A.6. Our DCPT implementation is dependent on a 8 kB large memory to update its history table at runtime. When DMAP-APM detects a sequential memory access pattern, then the large 8 kB memory for the DCPT table is no longer needed. Assuming that the DCPT table can be implemented in a separate scratch pad memory, i.e., SRAM, then this memory can be turned off when it is not used in order to save power. The assumption is in accordance with the observations made by Catthoor et. al. [16], where the power consumption is increased when we have a high degree of data transfer and storage. In Section A.6 we show how much this opportunity reduces the static power

consumption.

Our speedup results suggest that prefetching is beneficial with up to 40% performance improvement as compared to no prefetching. The coverage and accuracy rating in Figures A.8 and A.10, show that a high degree of cache pollution or bad timeliness is present when the sequential on-access prefetching scheme is not able to prefetch the needed data due to unpredictable memory access patterns, based on the low coverage and accuracy. Therefore, the probability of cache hits decreases, and the load instructions require longer time to fetch the correct data from DRAM. Hence, the processor executes fewer instructions per cycle on average.

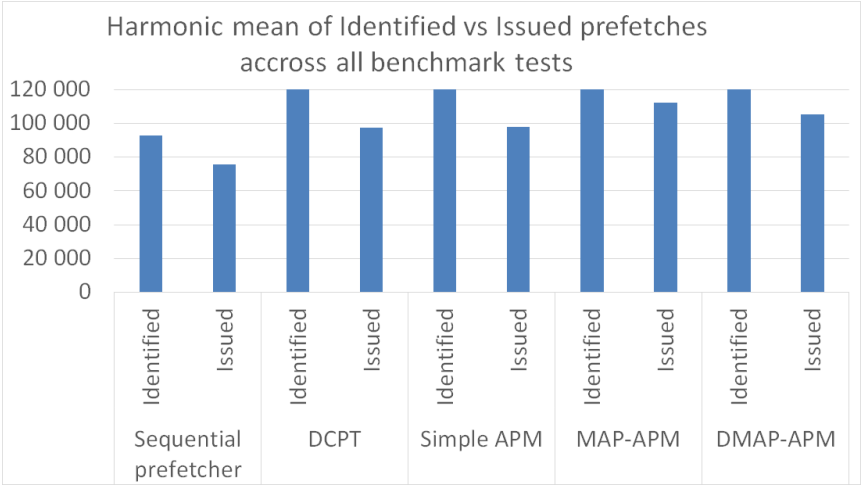


Figure A.11: Harmonic mean of identified vs issued prefetches.

Figure A.11 shows that DMAP-APM is able to identify and prefetch approximately 9% and 7,7% more data, respectively, than the DCPT scheme on average. In general, the DCPT scheme outperforms the sequential on-access scheme in speedup, accuracy and coverage. Both the accuracy and coverage ratings, with the exception of a few cases, are significantly better in the DCPT scheme. The only cases where the sequential on-access scheme has better accuracy and coverage than the DCPT scheme is when the access pattern is sequential and predictable.

A.6 Power and energy consumption estimation

In each test run, the M5 [13] simulation output statistics and configuration files were used as inputs to the McPAT integrated power, area, and timing modeling framework for multi-core and many-core architectures [63], [62]. This framework also integrates the memory models from CACTI [119] in order to estimate the power consumption of the DRAM and SRAM memory activity.

A.6.1 Dynamic power and energy evaluation

We used McPAT v1.0 to simulate our platform and estimate the dynamic run-time power consumption. Each prefetching scheme is simulated standalone, and the estimated power consumption results are compared to the case when no prefetching is used.

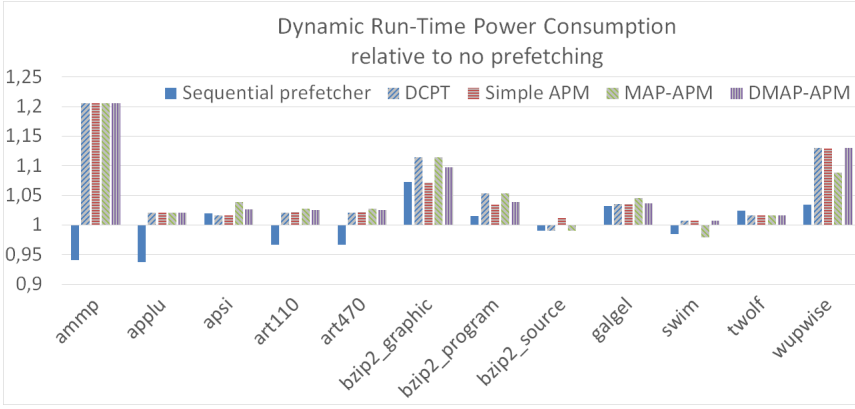


Figure A.12: Dynamic run-time power consumption relative to no prefetching.

Figure A.12 represents the dynamic run-time power consumption relative to no prefetching for each benchmark test. In most cases, enabling a prefetching scheme resulted in an increase of the dynamic run-time power consumption. In cases where the speedup of a prefetching scheme is worse than no prefetching, we observe a reduction in dynamic run-time power consumption. By comparing the power consumption results with the speedup results in Figure A.7, we observe that the scheme with the best speedup in each benchmark test is the most power consuming scheme. This is not

surprising since large amounts of data are prefetched into the cache in order to speed up the execution time of the application. All prefetching schemes are simulated with and without the APM detection mechanism in order to estimate the power consumption of the APM mechanisms. In all of our APM variations, the overhead in power consumption is too small to be measurable.

The dynamic energy consumption for each benchmark program, relative to no prefetching is shown in Figure A.13. These results are based on a simulation of 100 million benchmark instructions at a core clock frequency of 2 GHz. From Figure A.13, it becomes clear that the sequential on-access, DCPT scheme and our APM solutions reduce the total energy consumption compared to no prefetching when the speedup is greater than one. The prefetching schemes consume more power to perform prefetching, resulting in an increased rate of L2 cache hits. Therefore, the total number of cycles used to perform 100 million instructions is reduced significantly, since the majority of these instructions are data transfers. Compared to no prefetching, the dynamic energy consumption for the DCPT scheme and our DMAP-APM solution is reduced by up to 15.20%, and 15.15%, respectively.

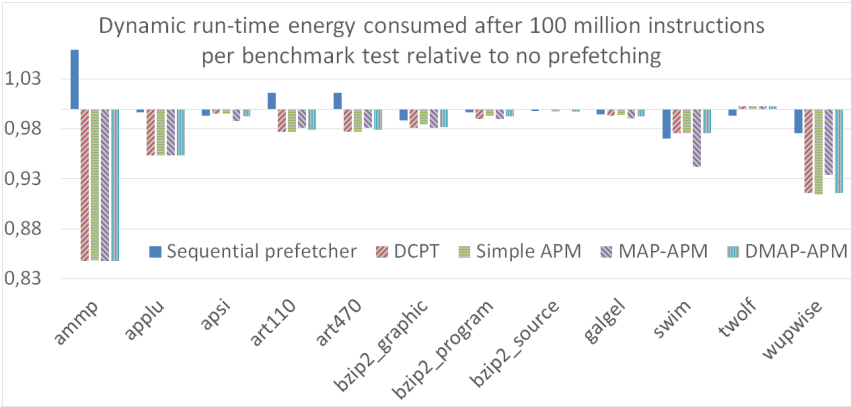


Figure A.13: Dynamic energy consumption relative to no prefetching.

In the current M5 implementation, the platform is not able to turn off parts of the memory used by the DCPT table when not needed. Otherwise, we expect that the static power and energy consumption would decrease. Furthermore, we expect significant gain in power and energy efficiency if the sequential on-access and DCPT schemes are implemented as HW accelerators. Each HW accelerator could then be isolated as a separate unit, and powered on or off when needed by the APM. In this case, the large 8 kB

DCPT table could be implemented as a separate scratchpad memory, i.e., SRAM, dedicated to the DCPT accelerator. When the DCPT scheme is not in use, it is turned off completely. Additional prefetching schemes can be integrated as HW accelerators with different complexity levels. This allows us to improve our dynamic run-time memory access pattern detection at the cost of increased chip area.

A.6.2 Static power and energy evaluation

We profiled our different APM implementations in order to expose the possibilities for static power and energy optimizations. Figures A.14, A.15, and A.16 show the percentage of time that simple APM, MAP-APM and DMAP-APM use the different prefetchers. These results clearly shows that our DMAP-APM is able to dynamically adapt the prefetching schemes according to predictable sequential memory access patterns, compared to our previous implementations.

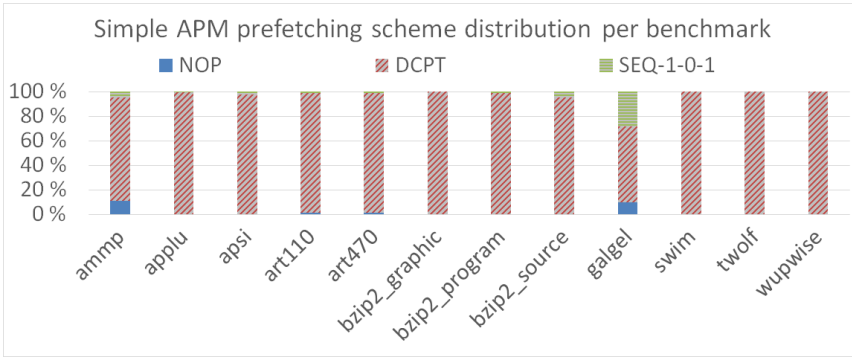


Figure A.14: Runtime prefetching scheme selection for our simple APM.

Based on the profiled DMAP-APM results, we estimated the reduction in static energy consumption. In our estimation we assume that the DCPT history table are implemented in a dedicated SRAM memory with normal, sleep, non destructive deep sleep and destructive shut down modes. We estimate the energy consumption based on the standard cell SRAM power model provided by IMEC. The results are shown in Figure A.17. Our DMAP-APM scheme results in up to 70.9% reduction in static energy consumption as compared to the DCPT scheme. This reduction corresponds to the amount of time the DCPT scheme is not active in our DMAP-APM so-

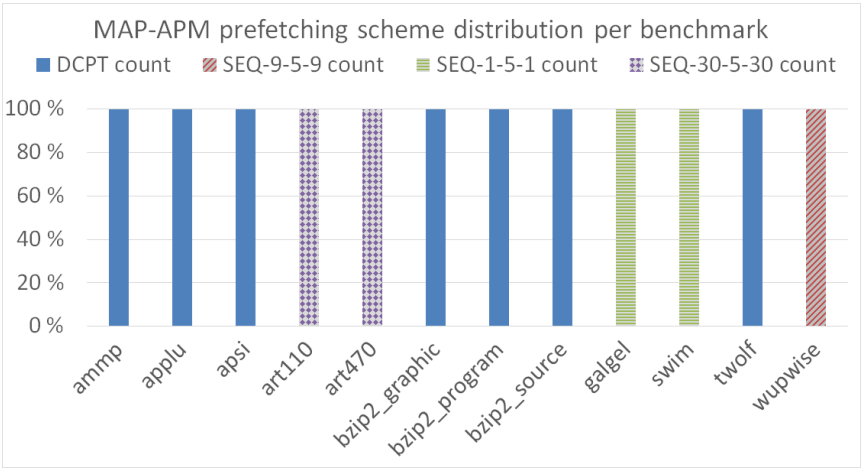


Figure A.15: Runtime prefetching scheme selection for our MAP-APM.

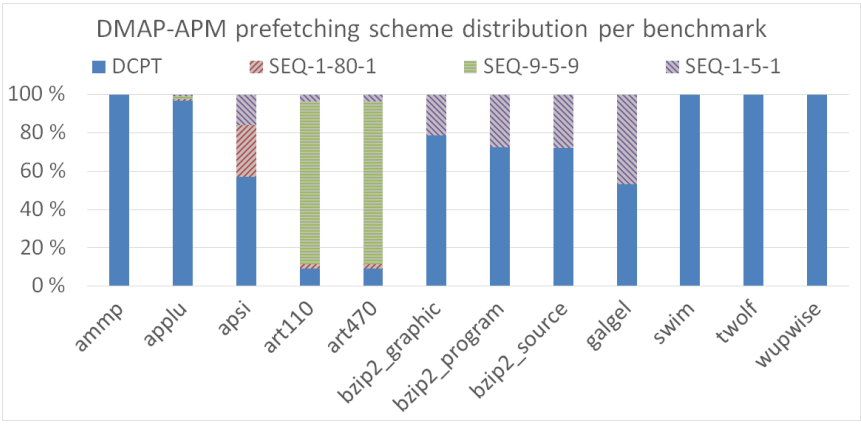


Figure A.16: Runtime prefetching scheme selection for our DMAP-APM.

lution. The increase in overall system dynamic energy consumption caused by the DMAP-APM scheme corresponds to 4.5% of the static energy reduction.

A.7 Conclusions

We combine prefetching schemes with different complexities by using an adaptive prefetch manager (APM). Three different APM implementations

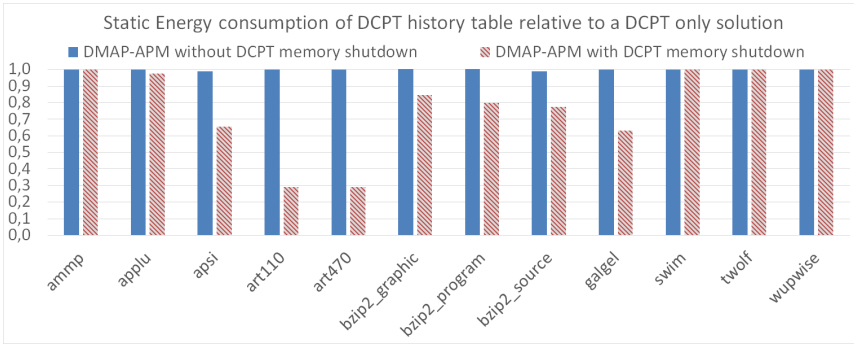


Figure A.17: Static energy consumption consumed by the DCPT memory.

are designed and evaluated. Our best APM solution (DMAP-APM) use lightweight monitoring mechanisms to detect known sequential memory access patterns at run-time. When a known pattern is detected the active prefetching scheme is switched dynamically in order to reduce the static power consumption of the DCPT history table memory. The DMAP-APM is a novel priority based APM which dynamically analyzes and detects interleaved and individual access patterns from memory load instructions at runtime. To our knowledge, no similar solution exist today.

Compared to no prefetching, the dynamic energy consumption for the DCPT and DMAP-APM scheme is reduced by up to 15.2% and 15.15%, respectively. DMAP-APM is able to identify and prefetch approximately 9% and 7,7% more data, respectively, than the DCPT scheme on average. More prefetched data leads to a speedup improvement if they are prefetched before they are referenced by the processor. This is the case for the DMAP-APM solution.

Additional power and energy reductions are expected if each prefetcher scheme is implemented as a separate HW accelerator. Each accelerator could be equipped with small dedicated scratchpad memories, i.e., SRAMs, for their private data structures. These memories replaces the current space allocated for them by the prefetching history tables in the systems main memory. Hence, the systems main memory size requirement is reduced accordingly. At run-time, only the needed accelerator is switched on, and all other accelerators are turned off. The chosen accelerator switch on the needed SRAM memory at runtime, and turns it off when the accelerator is disabled. Our solution will therefore consume less static energy compared to a similar solution without this flexibility.

Exact evaluation of the above mentioned accelerator approach is left for

future work. As proof of the potential, a 40 nm standard cell 8 kB SRAM model from the Interuniversity Micro-Electronics Centre (IMEC) is used to estimate the static power and energy consumption of the DCPT history table. Our DMAP-APM solution resulted in up to 70.9% reduction of the static energy consumption as compared to a standalone DCPT scheme solution. The dynamic energy overhead of our DMAP-APM mechanism is 2%, and corresponds to 4.5% of the static energy reduction. The effective reduction in the history table energy consumption is therefore 66.4% compared to a standalone DCPT scheme solution.

APPENDIX

B

SYSTEM SCENARIO FRAMEWORK EVALUATION ON EFM32 USING THE H264/AVC ENCODER CONTROL STRUCTURE

Yahya H. Yassin, Per Gunnar Kjeldsberg and Francky Catthoor
2015 European Conference on Circuit Theory and Design (ECCTD)
Trondheim, 2015
pages 1-4
DOI: 10.1109/ECCTD.2015.7300121

Abstract

Hand held battery powered devices are becoming increasingly complex, with embedded applications that regularly demonstrate very dynamic behavior and requiring advanced functionality. To avoid prohibitively high power consumption, novel techniques are needed to handle the increased dynamicity and complexity, e.g., through use of system scenario based design. In this paper a framework for system scenario resource management is presented and evaluated with the H264/AVC encoder control structure on an EFM32 microcontroller. By using DFS as a system knob, we obtain significant power reductions for different system scenarios occurring dynamically while the application is running. The framework overhead is negligible.

B.1 Introduction

In the recent years, energy efficiency and power consumption has become a major design constraint. Hand held devices are becoming more popular, and the applications embedded in these devices require more advanced functionality. This often results in increased power consumption. Thus, energy efficiency is an essential design criterion for mobile hand-held devices. Miettinen et al. [71] point out that software parallelism is becoming a trend of future embedded systems, often in combination with heterogeneous architectures. However, parallel architectures introduce communication challenges that could lead to more energy consuming software applications. Industrial tools and practices are fundamentally lacking in this area [71]. Other researchers have concluded that the trend of future embedded systems is clearly toward wireless, multimedia, multifunctional, and ubiquitous applications [65]. Many of these embedded applications turn out to be data dominated in the multimedia and telecommunication domain [16]. Furthermore, Catthoor et al. [16] concludes, through experiments, that a very large part of the power consumption is due to data storage and data transfer. Similar conclusions are made by the Compilers Creating Custom Processors (CCCP) Research Group at the University of Michigan (http://cccp.eecs.umich.edu/research/poweraware.php).

The system scenario methodology exploits the inherent dynamism of many systems, i.e., the fact that the functionality and resource requirement changes at run time under the influence of the environment [65]. Conventional hardware/software co-design approaches focus mainly on static analysis where no dynamism is present. When dynamism is introduced, then only the worst case behavior is used to implement the system. We project the generic but high-level description of the system scenario design methodology of Gheorghita et al. [39] to our problem formulation, resulting in novel instantiations. This methodology is used to efficiently tune system resources according to the applications needs, e.g., by utilizing Dynamic Voltage and/or Frequency Scaling, dynamically turning on and off idle memory banks and other processing units, or by remapping tasks and data into partitions implemented as hardware accelerators.

We propose a generic framework model that enables the use of a system scenarios approach through an application monitoring unit (AMU) and a platform adaption manager (PAM) for single-core and multi-core architectures. This is illustrated in Fig. B.1. In single-core architectures, the AMU and PAM are implemented in software (SW) as lightweight inline functions

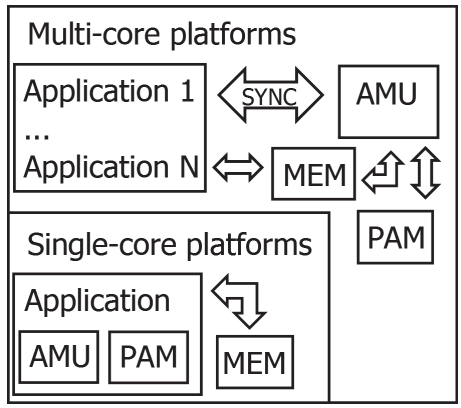


Figure B.1: Framework for single-core and multi-core platforms.

with minimal overhead. In homogeneous or heterogeneous multi-core architectures, the AMU and PAM are implemented in hardware (HW), running in parallel with the application. Our modeling approach is also extensible to multi-core platforms and then it is also applicable for customized single-core architectures with AMU and PAM implemented as customized units.

Ma et al. [65] propose a systematic design methodology for dynamic embedded real-time processing applications, known as the Task Concurrency Management (TCM) methodology. This methodology has further been used by Gheorghita et al. [39] for handling the dynamic nature of modern embedded systems. Through different case studies it is shown that this methodology obtain substantial energy reductions. They define five steps for the design-time and run-time phases. These steps include the identification, prediction, exploitation, switching and calibration of system scenarios. The run-time phase mechanisms are determined by the design-time design space exploration.

Commonly used techniques exist for controlling system configurations, such as Dynamic Voltage and Frequency Scaling (DVFS) and Dynamic Frequency Scaling (DFS). The DFS and DVFS techniques are proven to be a highly effective method of achieving low power consumption while meeting the performance requirements [22], [49]. However, these techniques are usually applied between application runs or when powering down the system to preserve energy. The methodology presented by Gheorghita et al. [39] enables the designer to apply these techniques while the application is running, by tuning the system knobs according to the predicted system scenario.

Ykman-Couvreur et al. [127] present a Run-time Resource Management (RRM) framework for embedded heterogeneous multi-core platforms. This framework is based on the system scenario design methodology, and allows dynamic adaption to change the application context, and transparent optimization of the platform resource usage. The limitations of this framework is that the total energy available has to be fixed and known, as well as the total size of input sequences. If the application changes some constraints dynamically at run-time, or if the input size is not known (e.g., in applications where the constraints are based on dynamically changing data-dependent variables), it cannot be handled by the current RRM. Furthermore, the existing RRM framework considers switching between applications running on different processing elements.

In this paper we propose an extension to the RRM framework, making it able to adapt the platform and configurations according to different system scenarios occurring dynamically within a running application. Several contributions are introduced. Custom platform related commands, such as a write to a clock division register, are implemented using inline function calls. These functions belongs to our PAM module, which consists of platform specific functionality. Our AMU mechanism consists of lightweight inline functions that triggers the PAM functions dynamically at run-time. Our AMU mechanism also monitors loop variables of nested and consecutive loops in the running application, and predicts at run-time which system scenario is about to take place before the loops execute. Lookup tables (LUTs) are used to store the AMU's prediction parameters. As will be shown, the overhead of our mechanisms are negligible in terms of power, compared to a system running the worst case scenario. Our framework introduces more freedom to apply power and energy reduction techniques, and provides a generalized framework for the system scenario methodology.

The rest of this paper is organized as follows. In Section B.2, we describe how we implement a portion of the H264/AVC encoder with the use of our Framework for System Scenario based designs (FSS). Our results are presented and discussed in Section B.3. We conclude this paper and present our plans for future work in Section B.4.

B.2 FSS evaluation using partial H264/AVC

Our single-core FSS framework is prototyped on an EFM32GG-STK3700 board from Silicon Labs [97]. The open source H264/AVC code obtained

from the Mediabench II website [60] is analyzed and profiled using video samples with different input sizes. The whole encoder part is extracted, but we focus on the control and data dependent control structure which is determining the system scenario behavior. Everything not related to the input frame size is replaced with delay statements delaying the code with the same number of clock cycles as the original code. Delay statements are inserted under the assumption that memory load and store instructions only need one clock cycle to finish its operations. Different control variables are tracked throughout the profiling, and system scenarios are classified based on the length of the nested loops controlled by the Run-Time Situation (RTS) parameters, i.e., control and data variables that have the highest impact on the run-time cost of the system. DFS is the only available system knob that can be tuned without disabling the CPU at run-time in the GG990F microcontroller on the EFM32 board. This microcontroller has an ARM Cortex-M3 core, and enables DFS through clock division. A frequency switch takes one system clock cycle when the system is set up properly, and is triggered by a write to dedicated registers. Furthermore, this kit includes an advanced energy monitoring unit which enables us to monitor the systems power and energy consumption while the application is running.

Our FSS framework will be used at run-time to actually perform the scenario prediction and switching according to the techniques determined during the design-time design-space exploration. FSS contains generic inline functions for communication, synchronization and prediction. The functions can easily be extended, and serve as a starting point for researchers and designers using the system scenario based design methodology.

The extracted H264/AVC code contains nested loops with memory accesses. The number of memory accesses are dependent on the loop variables. These loop variables change depending on the input data frame size. In our experiments, we isolated the nested loop behavior and implemented it in our framework on the EFM32 board. Instead of performing data manipulation on video frames we implemented delay operations as the core of our nested loops. In reality, the video frames are most likely stored in large off-chip memories, where each load and store instruction require multiple clock cycles. In such situations the use of our FSS framework would be even more beneficial due to the increased delay of the memory operations. Our goal is to demonstrate the effect of using our framework to determine the platform configuration on a defined control structure in nested loops. Therefore the actual manipulation of video frames are not relevant for our experiments.

In general, our FSS framework targets all applications with a long chain of control and data var-iable dependency. E.g., loop variables determined in earlier stages of the application which affect the total run-time.

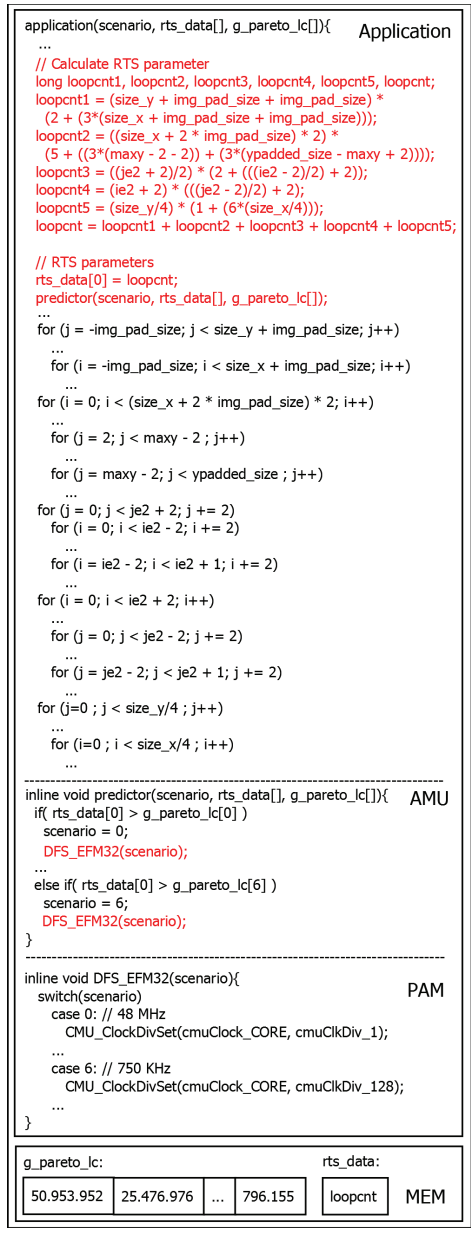


Figure B.2: Realization of the FSS framework on EFM32.

Fig. B.2 illustrates the realization of the FSS framework on EFM32. The PAM and AMU mechanisms are implemented in SW. The AMU monitors RTS parameters in the application at run-time, i.e., control and data dependent variables that dynamically determines the application behavior. The RTS parameters in our case consist of the total loop count (loopcnt), calculated from the loop control variables of the nested loops. A prediction mechanism (predictor) within this AMU predicts when a scenario switch takes place by reading the calculated loopcnt variable from memory, and comparing the value with the loop count boundaries in a lookup table in memory (g_pareto_lc). The predictor mechanism performs its task based on a set of constraints derived as a result from the design-time identification step. The PAM function (DFS_EFM32) consists of a HW system knob, with registers that can be written to change the frequency of the platform. The PAM is triggered by the predictor when it predicts a scenario. Tab. B.1 shows different video frame sizes, their total number of loop iterations detected by the loopcnt variable, and their predicted system scenario. Frame sizes that produces almost similar numbers of loop iterations are grouped together into one scenario. Each scenario operates at a different running frequency, which slows down or speeds up the application to match the run-time of our worst case scenario (Scenario 0 in Tab. B.1). Hence, we guarantee that the power consumed is not larger than the case of 1536x1536 frame sizes in the worst case. For lower input frame sizes we obtain significant power reductions, assuming that the processing time of 1536x1536 video samples are acceptable for all frame sizes.

The disadvantage with our single-core FSS model is that the AMU and PAM functions stall the application when they are executed. Therefore it is important that these functions are as lightweight as possible, and not executed too often. Tradeoff considerations must be made to make sure that the gain in power and/or energy reductions is significantly larger than the overhead of the AMU and PAM. This is the case when inherent dynamism is present in the application. A multi-core extension to our FSS model has the advantage that predictors and monitoring functions are performed concurrently with the application at run-time. When the AMU triggers a platform configuration change at safe application checkpoints, then the application is stalled through synchronization protocols. The disadvantage with this model is that the resource usage overhead will be larger than the single-core model. However, given that the application has significant amount of inherent dynamism, the added gain is seen in many applications to be significantly larger than the overhead introduced. That is demonstrated clearly

Table B.1: Scenario assignment based on run time

Input frame size	Run time (in million cycles)	Scenario
1536 x 1536	50.95	0
1920 x 1080	44.83	0
1280 x 720	20.04	1
720 x 576	9.09	2
720 x 480	7.59	2
640 x 360	5.10	3
352 x 288	2.28	4
288 x 162	1.07	5
176 x 144	0.60	6

in the results in the next section also.

The principles of our proposed FSS framework can be generalized for a broader set of applications. The procedures illustrated in Fig.2 are showing already the potential to reuse similar concepts for other applications which contain similar primitives, array computations and loops. In particular we need RTS parameter calculations, predictor/detectors, and a DFS/DVFS controller. That generalization goes beyond the scope of the current limited-size paper though.

B.3 Results

Fig. B.3 shows the systems power consumption with the fastest clock enabled compared to possible reductions with slower clock speeds for different input frame sizes. The slower clock speeds guarantees that the time budget is not exceeded for the corresponding frame size. As also shown in Fig. B.3, the implemented APM and PAM mechanisms introduce a 0.3% power overhead compared to only using the fastest clock available, which is negligible. The processing time for all frame sizes are stretched to approximately have the same processing time for 1536x1536 video resolution.

In our experiments, we sequentially encode different frame sizes before going to idle mode. Fig. B.3 clearly shows that we obtain significant power reductions for lower video frame sizes because we use DFS to stretch the encoding process over the available time budget. If we introduce DVFS, then energy reductions proportional to V^2 can be reached based on the formula

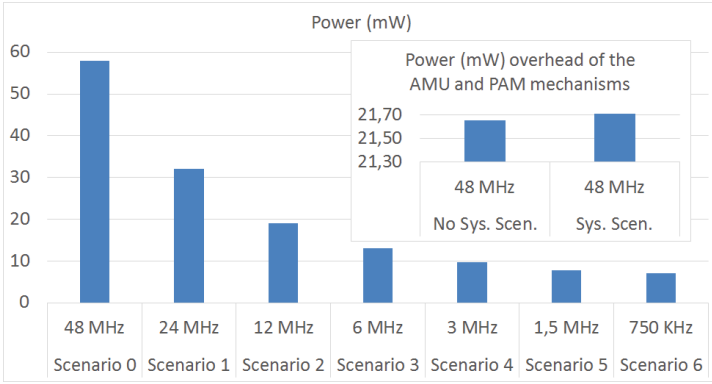


Figure B.3: Power consumption of the system at available frequencies and the FSS AMU and PAM power overhead.

for dynamic power for CMOS technologies [27], as shown in Equation B.1.

$$E = Pt = CV^2Aft \tag{B.1}$$

A multi-core FSS model is subject to future research, and we expect it to benefit from multi-core platforms that run different applications concurrently. Centralized AMU and PAM modules can monitor and adapt to RTS parameters in all concurrently running applications, either through localized and smaller AMU and PAM modules or directly. Tradeoff considerations have to be made in either case, and we expect to see similar challenges as cache coherence protocols in multi-core cache architectures.

B.4 Conclusion

Our FSS framework is developed to enable researchers and designers to use the system scenario based design methodology. Dedicated AMU and PAM units for run-time monitoring, prediction and platform adoption are used to implement different system scenario configurations in the platform through shared memory. These mechanisms are extendible and separated from the actual application, which in turn lowers the threshold to port this framework across different platforms. Dedicated HW mechanisms which are platform dependent are implemented in the PAM unit while application

monitoring and prediction mechanisms, dedicated for the applications system scenarios, are implemented in the AMU. We evaluate this framework on an EFM32 board with DFS capability. The H264/AVC encoder control structure, including delay statements to mimic the remaining behavior, is used as our test application. Our experiments shows that we can obtain significant power reductions when inherent dynamism is present in the application. We obtain up to 87.9% power reduction at the cost of longer execution time for small video frame sizes. The FSS overhead is shown to be negligible.

A second prototype is planned using an Atmel SAM4L board [8] to demonstrate the improved gain by dynamically tuning of multiple system parameters. Dynamic Voltage and Frequency Scaling (DVFS) will be used for this purpose. The work will later be ported to an instance of a Single-ISA Heterogeneous MAny-core Computer (SHMAC) on an FPGA based platform [54], where PAM and AMU run in parallel in a multi-core extension of our FSS. HW techniques will be used to improve our framework by implementing a customized AMU and PAM.

APPENDIX

C

DYNAMIC HARDWARE MANAGEMENT OF THE H264/AVC ENCODER CONTROL STRUCTURE USING A FRAMEWORK FOR SYSTEM SCENARIOS

Yahya H. Yassin, Per Gunnar Kjeldsberg, Francky Catthoor and Andrew Perkis
19th EUROMICRO Conference on Digital System Design (DSD'16)
Limassol, Cyprus, 2016
pages 222-229
DOI: 10.1109/DSD.2016.20

Abstract

Many modern applications exhibit dynamic behavior, which can be exploited for reduced energy consumption. We employ a two-phase combined design-time/run-time methodology that identifies different run-time situations and clusters similar behaviors into system scenarios. This methodology is integrated with our framework for system scenario based designs, which dynamically tune the hardware to match the application behavior. We achieve significant energy reductions for an extracted control structure of a video codec widely used in hand-held devices today. We encode a video stream consisting of different frame sizes based on measured available wireless bandwidth. Energy consumption is measured with a modified microcontroller board from Atmel with two alternative voltage and frequency settings. While maintaining the perceptual video quality and frame rate, our method results in up to 44% energy reduction for our encoded streams, even after including an average worst-case tuning overhead of 8.5%. In reality this overhead is typically negligible since the worst-case assumes very frequent tuning, while in realistic situations it will be performed at a much lower rate. This makes the expected gain over 50%.

C.1 Introduction

In embedded systems today, energy efficiency and power consumption is a major design constraint. Hand held devices are becoming more complex, because applications embedded in these devices require advanced functionality, in addition to increased parallelism. Many high end embedded systems today are more dynamic in nature, and conventional embedded system design techniques are not able to meet the requirements of today's demanding software functionality. State-of-the-art design methodologies try to cope with this challenge by identifying typical use cases and dealing with them separately, in order to reduce the increased complexity of the system [121]. Conventional techniques focus mainly on static analysis where no dynamism is present. When dynamism is introduced, then only the worst case behavior is used to perform the mapping. A gap exists between the high-level specification and the detailed implementation of embedded software design. A well balanced combined design-time/run-time two-phase methodology exist, and is known as the system scenario based design [65], [39]. This methodology consists of 5 design steps, and are described in Section C.2.1. We use this methodology in our framework for system scenarios (FSS) [126].

Reducing the power consumption does not necessarily result in energy efficiency. As an alternative, state-of-the-art techniques, such as race-to-halt (RTH) [10], are often used to speed up the execution of an application to achieve longer idle times. The static power consumption consumed with idle times is then reduced significantly due to the utilization of extremely low power states. RTH techniques reduces the total energy consumed by the system, and is considered state of the art for CPU bound workloads when the arrival time of the next set of tasks to be processed is unknown. According to Awan et al. [10], Dynamic Voltage and Frequency Scaling (DVFS) techniques are still more suitable for memory bound workloads and real time systems with deadlines and long idle times.

In general, multimedia applications consume significant amount of energy due to the stringent Quality of Service (QoS) requirements, as described by Hillestad [48]. Two-way interactive multimedia applications in particular have stringent requirements with respect to the quality of service delivered by the underlying transport network. Hillestad [48] generalizes two important requirements for IP-based streaming media applications. The first requirement is the real-time delivery to prevent buffer-underflow events, and the second requirement is smooth rate changes to prevent oscillations in perceived quality (e.g., quality as measured subjectively by end users).

For instance, if the frame size of a streaming video changes with less than 10 seconds intervals, or if the next frame size is significantly larger than the previous frame, then the perceptual video quality may affect the end-user's experience.

Interactive multimedia applications normally require the use of advanced video codecs, such as the widely used H264/AVC video codec [91]. Implementing the H264/AVC video codec in hand-held devices is a challenge by itself because of the stringent requirements described by Hillestad [48] above, and because custom high performance digital signal processors (DSPs) are needed. These high performance DSPs are in general more energy consuming than standard low-power of-the-shelf processors. On the other hand, general low power of-the-shelf processors struggle to process large video resolutions with acceptable processing times. Many researchers have therefore suggested multi-core solutions, where the H264/AVC codec is parallelized in order to concurrently process multiple frames or groups of pictures at the same time [129], [105].

Our goal is not only to improve the energy efficiency of the H264/AVC encoder, but generally improve the energy efficiency for data-dependent applications with similar behavior as the H264/AVC encoder. Therefore, we focus on the control structure of the H264/AVC encoder, as obtained from the Mediabench II website [60], on which we apply our framework for system scenarios (FSS) [126]. Our FSS framework is extended from only using Dynamic Frequency Scaling (DFS), to handling DVFS and RTH. The system scenarios approach is combined with RTH and DVFS on a SAM4L microcontroller board from Atmel [8], which contains an ARM Cortex M4 low power processor. In this paper we reduce the energy consumption by combining state-of-the-art techniques, i.e., DVFS and RTH, with the system scenario based design methodology [39] in our FSS framework. The experimentation is also substantially extended from our previous work [126].

In Section C.2.1 we give a brief overview of the system scenarios based design methodology presented by Gheorghita et al. [39], and in Section C.2.2 we highlight the importance of targeting data-variables. Related work is presented in Section C.3, and a motivational example is given in Section C.4. Our experimental setup is explained in Section C.5, how we identify our system scenarios is discussed in Section C.6, and the implementation of our prediction and switching mechanism is presented in Section C.7. We present and discuss the energy consumption in Section C.8, and conclude our work in Section C.9.

C.2 Definitions

C.2.1 The system scenario based design methodology

A two-phase design-time and run-time system scenario design methodology is presented by Gheorghita et al. [39]. It exploits the dynamic variation in an application, where system knobs, such as frequency, voltage, and low power modes, are tuned dynamically at run-time in order to gain significant reductions in energy consumption.

At design-time, different run-time situations (RTSs) are identified, usually through profiling. RTSs with similar costs, e.g., execution time, energy consumption, or memory requirement, are clustered into scenarios. These system scenarios are different from typical use case scenarios because they take detailed knowledge of the actual resource requirements into account, thus giving rise to more optimized designs [39].

A run-time scenario prediction mechanism is then determined at design-time, based on the output from the identification step. At run-time, the prediction mechanism determines whether or not to switch from one system scenario to another based on the actual system parameter values.

The exploitation step at design-time applies further optimizations to each scenario, similarly to what would have been done to the static implementation without the scenario approach. The system knob settings of each scenario are decided here. At run-time, the exploitation step is simply the execution of the scenario.

In order to change the system from one set of run-time platform configuration parameters (system knobs) to another, a scenario switching mechanism must be implemented. The switching mechanism to be used at run-time is determined at design-time. This mechanism will switch scenario based on the output of the prediction step at run-time.

An optional calibration configuration can be designed (at design-time) and used (at run-time) in order to fine-tune the decision making.

C.2.2 Control variables vs data variables

In order to better understand the challenge of data variable dependencies, Figure C.1 illustrates a simplified code snippet illustrating the difference between a control variable and data variable dependency.

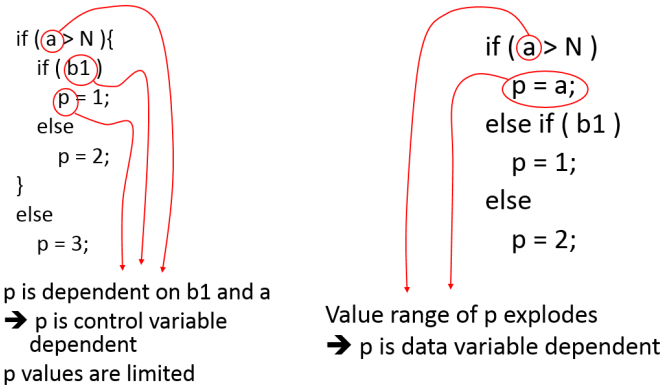


Figure C.1: Control vs data variable dependency

The code snippet on the left in Figure C.1 illustrates a control variable dependency. The variable p in this case have a limited value range, and it will be relatively easy to monitor the application behavior if p is later used, for example, as a loop parameter.

The code snippet on the right shows a data variable dependent situation, where the value range of p equals the value range of a. If a is an input parameter, and p is used as a loop parameter in succeeding computations, then the loop size varies with the input parameter value range. Assuming the input parameter is a 16-bit integer value, then the value range of a will be equal to 2^{16}. This value range would simply produce too much run-time overhead for the scenario prediction and switching mechanisms.

C.3 Related work

Many researchers have optimized energy efficiency in embedded systems by proposing improved dynamic scheduling algorithms, such as Leakage Control EDF (LC-EDF) and Leakage Control Dual Priority (LC-DP) for reducing leakage in hard real-time systems [61], 3-competitive offline and constant competitive ratio online algorithms for power saving while considering shutdown in combination with DVFS [51], and a combination of DVFS and shutdown to minimize the overall energy consumption based on the latest arrival time of jobs [79]. Most of these approaches produce significant analysis overhead. Awan et al. [10], propose a more efficient solution by combining an online and offline algorithm that selects the best feasible

sleep mode based on the available time slack for a given workload with significantly less run-time overhead.

A method for automatic discovery of system scenarios that incorporate correlations between different parts of applications is proposed by Gheorghita et al. [40], [38]. When many-valued parameters are considered, the overhead for scenario prediction is significant, however. Hammari et al. [45] propose a method based on the projection of scenarios onto an RTS parameter space, where many-valued parameters are handled with the use of polyhedral techniques. These techniques are needed to fully automate the scenario analysis, which is not required for partly automated systems where the scenario information is inserted by the designer.

In general, the prediction mechanism is highly dependent on how the identification step is performed, as indicated by previous work [40], [38], [45]. The authors in previous work conclude that the number of variables used in the prediction should be low. This is also the same for other identification approaches, e.g., the method of using loop-trip count profiling [82].

Previous work mainly focuses on control structures in applications with limited or no data variable dependency. When data dependent RTS-parameters dominates the application behavior, the value range increases significantly. Increased value range in the RTS parameter space usually increases the run-time prediction overhead.

Damavandpeyma et al. [24] use DVFS as a system knob, and propose a compact scenario aware data flow graph (SADF). Their approach assumes that the system scenarios have been clustered as part of the identification step, where the clustered result does not contain any many-valued parameters, and the number of control- and/or data-variables used in their analysis is limited. Their approach works well for data dependent variables with limited dependency variation.

The data variable challenge increases with the combination of multiple data-dependent RTS parameters and nested loops.

We combine the FSS framework from our previous work [126] with DVFS and RTH to achieve significant energy reductions when data variable dependency is present. The data variable dependency is handled by using inline functions that combine multiple RTS parameters, and classify different ranges as scenarios. Low overhead and globally placed detection equations are defined that can detect the required cycle count of the application. The output of these detection equations are input to our prediction mechanism. The prediction mechanism then compares its input to different ranges in a lookup table, before deciding which scenario to choose. Our approach is

scalable with increasing numbers of system knobs and an increased number of RTS-parameters within an application. In the following sections we will show how we were able to achieve significant energy reductions with our FSS framework implemented on a SAM4L microcontroller board.

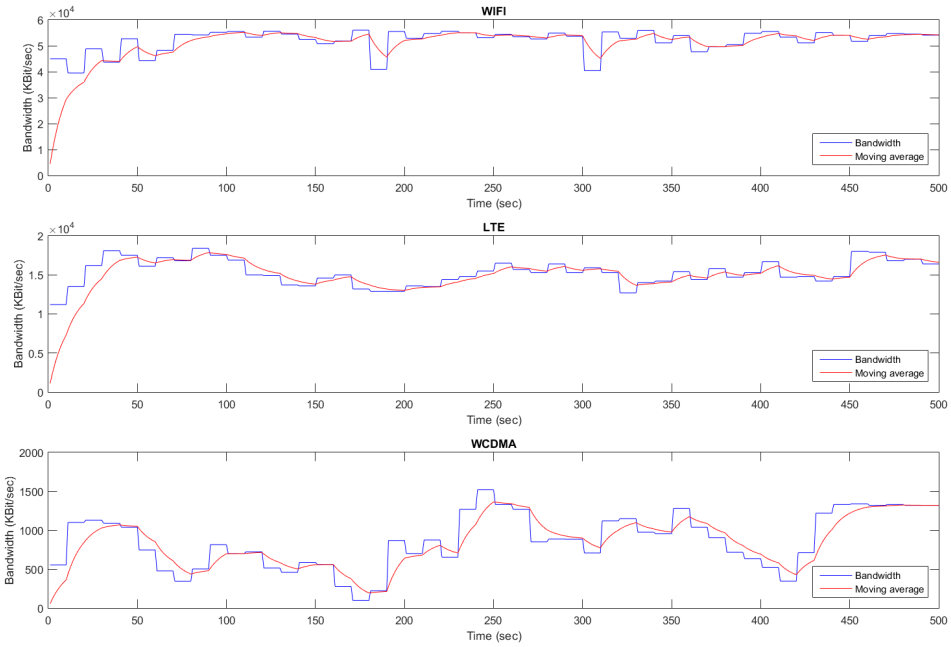


Figure C.2: Available bandwidth measurement and moving average over WIFI, LTE and WCDMA

C.4 Motivational example

Hand-held video communication via the internet has become more common in recent years through the use of programs such as Skype, Viber, and Hangouts. A major challenge with hand-held devices is that the power consumption increases significantly during a video call. The phone's camera is used to encode video and stream it over the internet continuously. At the same time, video from the caller is received and decoded continuously. Considering that video is streamed with a rate of 24-30 frames per second, the amount of simultaneous encoding and decoding is significant. One of the commonly known challenges is that the user experience of a video call can

vary based on the internet connection speed, the processing limitations of the hand-held device, and battery level. A popular and advanced codec such as H264/AVC is suitable for hand-held devices [91], and is considered as a state-of-the-art codec. The H264/AVC codec is far more advanced than its predecessors, and introduce more dynamism, especially with varying input frame sizes. This dynamicity can be exploited using the system scenarios design methodology presented by Gheorghita et al. [39].

There exist proposals for video resizers today, such as the "Apparatus and method for resizing an image" [111], which rescales a camera's video frame size. A simple control logic could easily be added to change the frame size of a picture, such that the dynamic behavior of the internet connectivity could be exploited and used to decide the scale factor of the video before it is encoded and transmitted.

Figure C.2 shows the results of measurements we have performed of available internet bandwidth for a 500 second period on three different wireless networks. The upper graph shows the wireless local area network (WIFI) at NTNU, while the middle and lower graphs shows measurements over mobile Long-term Evolution (LTE) [95] and wideband code-division multiple-access (WCDMA) [23] networks, respectively. The measurements are performed using the bandwidth measurement tool IPERF [1]. IPERF is used to measure available bandwidth between two IP-addresses. Our measurements are performed with 10 seconds intervals between a Sony Xperia Z3 android-based smartphone and a computer running Ubuntu. Figure C.2 clearly shows significant dynamicity in the available bandwidth, which we can exploit by using our FSS-framework [126]. In addition we calculate the moving average over the available bandwidth, as seen in Figure C.2.

We focus on the H264/AVC encoder control structure obtained from the Mediabench II website [60], where we adapt the platform's voltage and frequency according to the input frame size, assuming that a video resizer circuit rescales the input frames based on the moving average measurement of the available bandwidth in Figure C.2.

C.5 Experimental setup

The energy consumption of the H264/AVC encoder control structure is measured on a SAM4L board from Atmel [8], and we measure live current consumption and core voltage using a NI myDAQ measurement device [77]. Time stamped data is logged with a locally developed Labview program.

The single-core FSS framework is used to implement the H264/AVC encoder control structure with system scenarios.

Our version of the SAM4L board contains the ATSAM4LC4CA microcontroller. The microcontroller contains an ARM Cortex M4 processor, and supports two different run modes. RUN0 operates with a core voltage of 1.8V, while RUN1 operates at 1.2V. The microcontroller also supports various low power states, i.e., sleep modes, where the processor clock is turned off, and multiple oscillators with different frequency ranges. For the scope of our research, we have selected the internal oscillators RCFAST (12 MHz) and RC80M (80 MHz) as the main clock sources for our DVFS settings. Since the ARM Cortex M4 processor only supports frequencies up to 48 MHz, the RC80M oscillator is clock divided to 40 MHz when it is selected as the main clock source. Hence, our DVFS settings are PS0 (1.8 V and 40 MHz) and PS1 (1.2 V and 12 MHz).

Based on the SAM4L board datasheet, the DVFS is triggered by a write to configuration registers after a write to protective lock registers. In other words, a switch is triggered after a few clock cycles. We have measured the switching time between our two clock settings, which take approximately 15 clock cycles when both oscillators are on continuously, and approximately 50 clock cycles if the currently unused oscillator is powered down. The added power consumption by having both oscillators enabled is negligible because unused oscillators that are idling does not contribute to any measurable power consumption. We have measured the total static power, of which the oscillators only contribute a minor part, to be less than 16% of the total power consumption in PS0, and less than 29% in PS1. In our scope, we trade a slightly increased static power for a significantly reduced switching time. The static power consumption on the SAM4L board is approximated by measuring the total power consumption for two different frequencies in both power modes, and estimating the effective capacitance under the assumption that the activity factor (α) is equal to 0.5 in Equation C.1.

$$P_{\text{Total}} = P_{\text{Static}} + P_{\text{Dynamic}} = P_{\text{Static}} + \alpha C V^2 f \tag{C.1}$$

When performing dynamic voltage scaling on the SAM4L board, the main clock frequency supported in the new voltage level has to be configured first. According to the datasheet, the 1.2 V (PS1) configuration does not support frequencies larger than 12 MHz. The frequency setting must therefore be switched from 40 MHz to 12 MHz before the voltage switch from 1.8 V (PS0) to 1.2 V (PS1). When changing configuration from PS1 to PS0 the frequency must be scaled after the voltage switch to maintain

stability.

In addition, the voltage output from the voltage regulator has to be stable after a voltage switch before the CPU can continue processing. After a switch from PS1 to PS0 we have to wait for a status flag which signals that the voltage level has stabilized. The CPU is therefore stalled while waiting for this flag to be set in the SAM4L board. However, when a voltage switch is triggered from PS0 to PS1 then waiting for this flag is not necessary before continuing the CPU execution. As described by Atmel Support, when switching from a higher voltage level to a lower level, the voltage regulator is stable and will regulate fast enough to continue execution in succeeding cycles. The measured current consumption will gradually decrease after this switch until the charge on the decoupling capacitor connected to the core voltage has been consumed down to its PS1 level. The power delivered from the regulator on the other hand is equal to the PS1 level immediately after the voltage switch.

Our SAM4L board came initially with a 22 µF decoupling capacitor connected between the core voltage input and ground. The large decoupling capacitor resulted in extremely large switching times. This was confirmed by Atmel Support, and they recommended a change to 4.7 µF to guarantee that all modules would function properly. Since we are only interested in the CPU, we measured the DVFS switching times with lower decoupling capacitors. Our experiments showed that the CPU functioned correctly using a 1 µF decoupling capacitor, giving a DVFS switching time comparable to state-of-the-art processors. Table C.1 shows the switching times and energy overhead when switching from PS0 to PS1 and vice versa. The energy overhead is measured by measuring the time between the last instruction before and the next instruction after the DVFS switch, before multiplying this time with the power consumption for the run mode before the switch. An oscilloscope was used for the timing measurement.

Table C.1: DVFS switching times between PS0 and PS1.

From	To	Cycles @ 12 MHz	Switching time (µS)	Overhead Energy (µJ)
PS1	PS0	197	16.40	0.0623
PS0	PS1	15	1.25	0.0281

C.6 Scenario identification methodology

The H264/AVC encoder control structure is modeled under the assumption that all memory accesses take one clock cycle. This model is profiled at design-time in order to explore the dynamic range of loop iteration count variations for various input frame sizes. In addition, different voltage and frequency settings are profiled in order to find optimal DVFS configurations. Our profiled results showed that the optimal energy consumption was achieved if the maximum available frequency setting for each voltage level is selected. This observation resulted in two scenarios, PS0 (1.8V and 40 MHz) and PS1 (1.2V and 12 MHz).

Table C.2 shows an evenly distributed selection of true 16:9 resolutions that we have selected, which our SAM4L board can process within a one second time frame. It also shows the corresponding scenario selections and calculated run times, based on our available power modes in the SAM4L board. In Table C.2, the Scenario* column represents the scenario distribution for a platform with more available voltage settings, and is discussed in Section C.8.

The largest frame size the platform can handle from our selection is measured to be 816 x 459. The required run time (in clock cycles) for each frame size is estimated by counting all loop iterations and multiplying with all instructions within each loop based on the program dis-assembly. These numbers are compared to the measured run time for each frame size. From the measured results, we observed that the cycle count could be approximated by multiplying the number of all loop iterations for any input frame size with a factor of 4.22. This factor corresponds to the additional assembly instructions needed to set up and execute all for loops. The number of all loop iterations is calculated as shown in Listing C.1. The total number of iterations are combined together with other parameters for all the nested loops in the application, resulting in the loopcnt variable. This loopcnt variable is multiplied by the approximated factor of 4.22. The separation of the approximated factor allows for easy portability of the scenario prediction mechanism to other platforms, because other platforms might implement the loop handling instructions differently. This difference would result in a higher or lower factor. The largest frame processing time is therefore approximated to 34.7 million clock cycles in an unoptimized setting where no pipelines are utilized. We use this frame size processing time as our worst case scenario, and we exploit it in combination with DVFS and RTH for lower frame sizes. We benefit from DVFS if the processing time of a smaller

frame size with a lower frequency setting is less than the processing time of our worst case frame size. Therefore, we have to take our available system knobs into consideration when we classify our system scenarios.

Table C.2: Scenario assignment based on run time

Input frame size	Run time (million cycles)	Scenario	Scenario*
816 x 459	34.70	0	0
752 x 423	29.54	0	0
688 x 387	24.79	0	1
624 x 351	20.46	0	2
560 x 315	16.54	0	2
496 x 279	13.04	0	3
432 x 243	9.96	1	4
368 x 207	7.29	1	4
288 x 162	4.54	1	5
224 x 126	2.80	1	5

Our scenario selection will be dependent on the platform's running frequency. In PS1 the platform is processing a frame 70% slower than in PS0. Therefore, frame sizes that need more than 30% of the run time of our worst case frame size are processed with PS0 (Scenario 0), and smaller frame sizes are processed with the PS1 configuration (Scenario 1).

Our arguments above are generalized in Equation C.2, where we set the Scenario 1 upper limit for the maximum frame processing time in PS1 to be equal to the frame processing time of our largest frame (816 x 459) in PS0. The Scenario 1 upper limit is calculated as shown in Equation C.3. In Equation C.2 and C.3, $T_{\text{WC frame}}$ is the execution time in seconds of the worst case frame size. T_{sc_1} is the maximum allowed execution time for Scenario 1 in seconds, LC_{sc_0} and LC_{sc_1} are the total number of loop iterations for a specific frame in PS0 or PS1, respectively. β is the approximated scale factor used to get the total number of clock cycles, and f_{sc_0} is the platform's main clock frequency when Scenario 0 is active. In other words, if the required frame processing time is above 30% of our worst case frame processing time, then the platform switches from Scenario 1 to 0.

$$T_{\text{WC frame}} = T_{\text{sc}_1} = \frac{\beta \cdot LC_{\text{sc}_0}}{f_{\text{sc}_0}} = \frac{\beta \cdot LC_{\text{sc}_1}}{0.3 \cdot f_{\text{sc}_0}} \tag{C.2}$$

$$LC_{\mathrm{sc}_1} = 0.3 \cdot LC_{\mathrm{sc}_0} = 0.3 \cdot 8,222,691 = 2,466,807 \tag{C.3}$$

C.7 Prediction and switching mechanism

The input frame sizes of the H264/AVC encoder is used directly in the calculations to determine succeeding loop sizes and memory accesses. The input data variable dependency is modeled as part of our application monitoring unit (AMU) in the FSS framework [126]. This dependency is shown in Listing C.1, where inline application dependent equations are pre-calculated (lines 2-7), before a global lookup-table is updated with the combined RTS parameter (line 10) as part of the overall AMU. This RTS parameter is then read by the AMU_1 function, which detects the next scenario (line 13). The application dependent calculations in lines 2-6 correspond to succeeding (nested) for- and/or while-loops in the encoder control structure, which are combined together in line 7 to estimate the required frame processing time in terms of loop iterations.

```
1  // Functions for monitoring RTS parameters
2  long loopcnt1 = (size_y + img_pad_size + img_pad_size) * (2 + (3*(
       size_x + img_pad_size + img_pad_size)));
3  long loopcnt2 = ((size_x + 2 * img_pad_size) * 2) * (5 + ((3*(maxy
       - 2 - 2)) + (3*(ypadded_size - maxy + 2))));
4  long loopcnt3 = ((je2 + 2)/2) * (2 + (((ie2 - 2)/2) + 2));
5  long loopcnt4 = (ie2 + 2) * (((je2 - 2)/2) + 2);
6  long loopcnt5 = (size_y/4) * (1 + (6*(size_x/4)));
7  long loopcnt = loopcnt1 + loopcnt2 + loopcnt3 + loopcnt4 +
       loopcnt5;
8
9  // Update combined RTS parameter
10 (g_rts_data)[0] = loopcnt;
11
12 // Scenario detection/prediction
13 AMU_1(&g_scenario);
```

Listing C.1: Functions combining data-dependent RTS parameters

The AMU_1 function is a light weight inline function which compares the global RTS lookup-table values with global scenario lookup table values in order to determine which scenario the platform will adopt to. This approach is a generalization that takes into account multiple scenario parameters and RTS parameters. The generalization with the use of lookup tables would simplify an extension to multiple independent RTS parameters

and/or scenarios, while in our particular case it would have been enough to compare the RTS parameter with the threshold between Scenario 0 and 1 directly. The values in the global scenario lookup table is found from our identification step, from Equation C.3. If the value of the global RTS lookup table is greater than the value in the global scenario lookup table, then the platform changes its configurations to PS0 if it is in PS1, and vice versa. The platform adaption manager (PAM) function is then called by the AMU and performs the platform re-configuration using the DVFS system knob as described in Section C.5.

C.8 Experimental results

In our experiments we simulate a realistic video sequence of 500 seconds, assuming it can be processed with 25 ARM Cortex M4 cores in parallel in order to meet the timing requirements for a frame rate of 25 frames per second (fps). The parallel cores are assumed able to process each frame individually with negligible synchronization overhead. This solution requires buffering and software pipelining of the frames to maintain forward frame dependency requirements. Other researchers have suggested similar approaches where their focus targets the parallelization of the H264/AVC encoder [129], [105].

Our scenario implementation is independent of this parallelization, and can be used on a faster processor for a single-core solution. After each frame is processed, we force the core into a low power retention mode, where the measured energy consumption is negligible. Before entering retention mode, a configurable timer is set to wake up the system just before the next frame arrives.

Our video sequence is measured using all three IPERF network measurements in Figure C.2. To process a frame, we specify a threshold requirement for the available bandwidth to be equal to at least five times the required bandwidth for a frame rate of 25 fps with a particular frame size. This threshold allows other activity to continue on the network, and avoids using up all the available bandwidth. The limited number of available voltage levels in the SAM4L board would otherwise force the system to use the most power consuming voltage setting for lower thresholds instead of exploiting the dynamism in the available bandwidth. Our simulated 500 seconds video sequence is created by combining different frame sizes from Table C.2 in a sequence that fits with the available bandwidth of the measured networks in Figure C.2, and our threshold requirements.

The application code is compiled and measured in a conventional way with compiler optimization parameter O3. When we optimize the application code with the O3 configuration, we assume it is a good enough optimization for the exploitation step in the system scenario design methodology presented by Gheorghita et. al. [39]. The energy consumption is measured in real time using our experimental setup, with and without the scenario mechanism. The overhead energy consumption of our scenario mechanism is measured by forcing the scenario mechanism to select PS0 each time a scenario switch takes place. This approach would not reduce the energy, but would trigger the scenario mechanisms in our AMU and PAM modules. The results are then compared to the energy consumption without using the scenario mechanism.

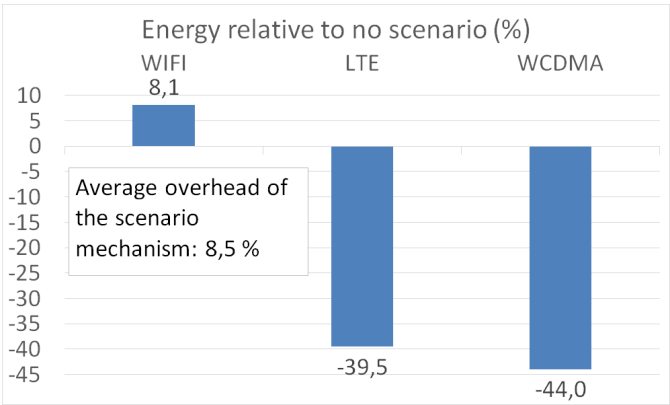


Figure C.3: Energy consumption relative to no system scenarios.

Figure C.3 shows the energy reductions relative to the energy consumed by a no scenario implementation utilizing a race to halt (RTH) setup in each of the measured networks of Figure C.2. A retention mode is used when the system is halted, which consumes less than 4.4 μW . The average energy overhead of our scenario mechanism is measured to be 8.5% for the 500 second sequence. These measurements are taken using an extreme conservative approach, however, where the scenario mechanism is triggered between each frame even if the scenario for the next frame is the same as the previous one. We perform this conservative triggering approach in all our measurements. In reality, it is not necessary to trigger the scenario mechanisms if the next predicted scenario remains the same. The total amount of actual scenario switches are therefore significantly lower than per processed frame. In fact, our system scenario mechanism only switches

from PS0 to PS1 two times, and from PS1 to PS0 one time during the 500 second sequence for the LTE network. The rest of the time, our conservative triggering approach forces each frame to trigger a DVFS switch to the same DVFS setting. If this is avoided, for example by a simple if-sentence that only triggers the switching mechanism if the next DVFS setting is different than the existing setting, the energy overhead of the scenario mechanism will be negligible for the network examples shown here.

In the best case situation, all frame sizes are suitable for the PS1 configurations. We obtain in this situation up to 44% energy reduction as compared to a RTH solution without the use of the scenario mechanism. These results include the average scenario mechanism overhead of 8.5% energy increase for the 500 second sequence with our conservative triggering approach. When the available bandwidth is higher than our threshold for our largest frame size, then the scenario mechanism is not needed. It simulates our worst case situation, where the system is running constantly in the PS0 platform configuration. In this situation the energy consumption does not increase more than the measured scenario overhead. Our results meets Hillestad's [48] requirement for perceptual video quality, as mentioned in Section C.1, while maintaining the frame rate.

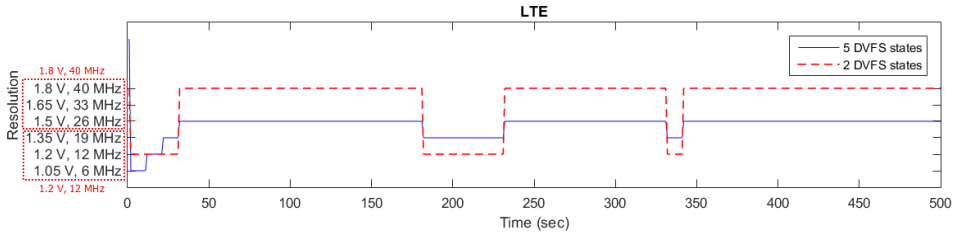


Figure C.4: DVFS state transitions during the 500 second video clip.

Our SAM4L board configuration capability is limited to two voltage states. If more voltage states were available, which is the case for modern Intel processors for example, the energy consumption is expected to decrease further since each frame size could correspond to a separate DVFS setting. To demonstrate the added gain of more DVFS settings, we have extrapolated our available DVFS settings with six evenly spaced voltage and frequency levels ranging from 1.05 V to 1.8 V and from 6 MHz to 40 MHz, respectively. The corresponding scenario assignment for the different frame sizes are shown in the scenario* column in Table C.2. We simulated our video sequence by manually applying the new scenario selection based on our bandwidth measurements shown in Figure C.2, and calculating the

dynamic energy. A lower threshold value, equal to four times the required bandwidth, is selected to utilize better our extrapolated DVFS settings. The increased number of DVFS settings allowed the system to use a lower threshold for the bandwidth requirement to process a particular frame. Our simulated result based on extrapolated DVFS settings is shown in Figure C.4 for the LTE network. From this figure, we observe that the the system switches to a higher frequency five times, and it scales down three times. Our existing solution with two DVFS settings would switch back and fourth between two settings 6 times in total. Taking Equation C.1 and the energy overhead we measured from Table C.1 into consideration, our extrapolated results indicates that the dynamic energy is now reduced 59% compared to not having scenarios.

C.9 Conclusion

Our framework for the systems scenario based designs (FSS) is combined with race to halt (RTH) and dynamic voltage and frequency scaling (DVFS) in order to achieve significant energy reductions for an H264/AVC encoder control structure. We measure our results on a modified SAM4L board from Atmel, which enables online DVFS tuning with switching times comparable to state-of-the-art switching times.

System scenarios are identified based on data-dependent control variables, varying with the input frame size. These scenarios are predicted by using inline estimation functions for the expected execution time. These functions are executed at run time, and are compared to a predefined set of ranges for the scenario prediction with minimum overhead. Our switching mechanism then re-configures the platforms voltage and frequency accordingly.

We use IPERF to measure the available bandwidth in WIFI, LTE, and WCDMA networks for 500 seconds. A video stream consisting of different frame sizes is then composed based on the measured network data and a threshold requirement. Compared to a no system scenario solution with RTH we achieve up to 44% energy reduction for the encoded streams, including a 8.5% average energy overhead with conservative tuning. The perceptual video quality and frame rate is maintained, and we show how we could improve the energy consumption a further factor 2 if a few more voltage levels are available by extrapolating our measured results. We expect similar results to be obtained by other applications with similar character-

istics. The techniques for scenario identification and switching presented here enables efficient use of our framework for system scenarios, especially for applications where the dynamic behavior is determined by many-valued data variables.

APPENDIX

D

TECHNIQUES FOR DYNAMIC HARDWARE MANAGEMENT OF STREAMING MEDIA APPLICATIONS USING A FRAMEWORK FOR SYSTEM SCENARIOS

Yahya H. Yassin, Per Gunnar Kjeldsberg,
Francky Catthoor and Andrew Perkis
Microprocessors and Microsystems (MICPRO)
Special issue on
"Selected Papers from DSD 2016"
2018
Volume 56, pages 157-168
DOI: 10.1016/j.micpro.2017.12.002

Abstract

Many modern applications exhibit dynamic behavior, which can be exploited for reduced energy consumption. We employ a two-phase combined design-time/run-time methodology that identifies different run-time situations and clusters similar behaviors into system scenarios. This methodology is integrated with our framework for system scenario based designs, which dynamically tune the hardware to match the application behavior. We focus on streaming media applications, and achieve significant energy reductions for an extracted control structure of a video codec widely used in hand-held devices today. We encode a video stream consisting of different frame sizes based on measured available wireless bandwidth. Energy consumption is measured with a modified microcontroller board from Atmel with two alternative voltage and frequency settings. While maintaining the perceptual video quality and frame rate, our method results in up to 49% energy reduction for our encoded streams. The average tuning overhead of our mechanism is negligible after applying simple loop transformations to the encoder control structure. We furthermore show how to obtain up to 34.3% energy reductions for a system with limited slack available, by dynamically exploiting the available sleep modes.

D.1 Introduction

In embedded systems today, energy efficiency and power consumption is a major design constraint. Hand held devices are becoming more complex, because applications embedded in these devices require advanced functionality, in addition to increased parallelism. Furthermore, many high-end embedded systems today are dynamic in nature, and conventional embedded system design techniques are not able to meet these requirements in an energy efficiently way. State-of-the-art design methodologies try to cope with this challenge by identifying typical use-cases and dealing with them separately, in order to reduce the increased complexity of the system [121]. Conventional techniques focus mainly on static analysis where no dynamism is present. When dynamism is introduced, then only the worst-case behavior is used to perform the mapping. As an alternative, a well balanced combined design-time/run-time two-phase methodology, known as the system scenario based design [65], [39], has been developed. Instead of focusing on different use cases, this methodology exploits the application behavior at system level, resulting in a substantially larger optimization potential. This methodology consists of five design steps, and are described in Section D.2.1. We use this methodology in our framework for system scenarios (FSS) [126].

Focusing on reducing the power consumption does not necessarily result in energy efficient designs. As an alternative, techniques such as race-to-halt (RTH) [10] can be used to speed up the execution of an application, thus increasing instantaneous power consumption, in order to finish early and achieve longer idle times. The static power consumed during idle times are reduced significantly due to the utilization of extremely low power states. RTH techniques thus reduces the total energy consumption of the system and is considered state of the art for CPU bound workloads when the arrival time of the next set of tasks to be processed is unknown. According to Awan et al. [10], Dynamic Voltage and Frequency Scaling (DVFS) techniques are on the other hand more suitable for memory bound workloads, and for real time systems with deadlines and long idle times.

In general, multimedia applications consume significant amounts of energy due to their high Quality of Service (QoS) requirements, as described by Hillestad [48]. Two-way interactive multimedia applications in particular have stringent requirements with respect to the quality of service delivered by the underlying transport network. Hillestad [48] generalizes two important requirements for IP-based streaming media applications. The first

requirement is the real-time delivery to prevent buffer-underflow events. The second requirement is smooth rate changes to prevent oscillations in perceived quality (e.g., quality as measured subjectively by end users). For instance, if the frame size of a streaming video changes with less than 10 seconds intervals, or if the next frame size is significantly larger than the previous frame, then the perceptual video quality may affect the end-user's perception.

Interactive multimedia applications normally require the use of advanced video codecs, such as the widely used H264/AVC video codec [91]. Implementing the H264/AVC video codec in hand-held devices is a challenge by itself because of the requirements described by Hillestad [48] above, and because custom high performance digital signal processors (DSPs) are needed. These high performance DSPs are in general more power consuming than standard low-power of-the-shelf processors. On the other hand, general low power of-the-shelf processors struggle to process large video resolutions with acceptable processing times. Many researchers have therefore suggested multi-core solutions, where the H264/AVC codec is parallelized in order to concurrently process multiple frames or groups of pictures at the same time [129], [105].

Our goal is to not only improve the energy efficiency of the H264/AVC encoder, but also generally improve the energy efficiency for data-dependent applications with similar control structures as the H264/AVC encoder. Therefore, we focus on the control structure of the H264/AVC encoder, as obtained from the Mediabench II website [60], on which we apply our framework for system scenarios (FSS) [126]. In [124] our FSS framework is extended from only using Dynamic Frequency Scaling (DFS), to handling DVFS and RTH. The system scenarios approach is combined with RTH and DVFS on a SAM4L microcontroller board from Atmel [8], which contains an ARM Cortex M4 low power processor. The energy consumption is reduced by combining state-of-the-art techniques, i.e., DVFS and RTH, with the system scenario based design methodology [39] in our FSS framework. The experimentation is also substantially extended from our previous work [126]. In this journal extension of [124] we reduce the run-time overhead of our scenario mechanism with loop transformations resulting in negligible energy overhead. Our FSS framework is also further upgraded to select the best feasible sleep mode in the SAM4L board based on the remaining slack time available.

In Section D.2.1 we give a brief overview of the system scenarios based design methodology presented by Gheorghita et al. [39], and in Section

D.2.2 we highlight the importance of targeting data-variables. Related work is presented in Section D.3, and a motivational example is given in Section D.4. Our experimental setup is explained in Section D.5, how we identify our system scenarios is discussed in Section D.6, and the implementation of our prediction and switching mechanism is presented in Section D.7. We present and discuss the energy consumption in Section D.8, and conclude our work in Section D.9.

D.2 Terminology

D.2.1 The system scenario based design methodology

A two-phase design-time and run-time system scenario design methodology is presented by Gheorghita et al. [39]. It exploits the dynamic variation in an application, where system knobs, such as frequency, voltage, and low power modes, are tuned dynamically at run-time in order to gain significant reductions in energy consumption. The steps are summarized in Figure D.1.

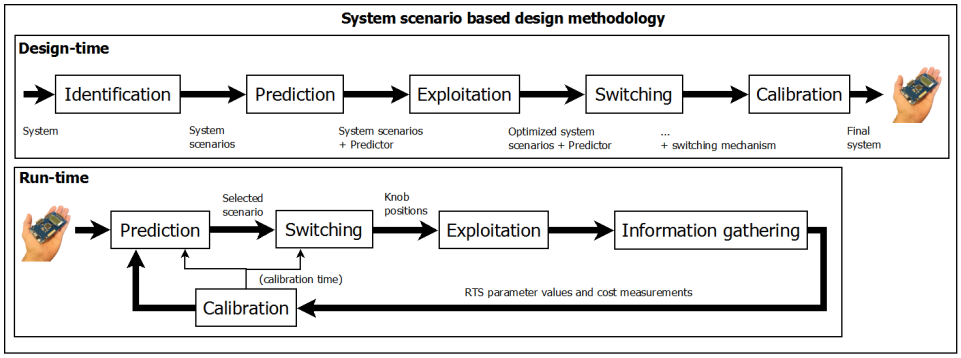


Figure D.1: The system scenario based design methodology overview adapted from [39].

As shown in Figure D.1, different run-time situations (RTSs) are identified at design-time, usually through profiling. RTSs with similar costs, e.g., execution time, energy consumption, or memory requirement, are clustered into scenarios. These system scenarios are different from typical use-case scenarios because they take detailed knowledge of the actual resource requirements into account, thus giving rise to more optimized designs [39].

Based on the output from this identification step (Figure D.1), a run-time scenario prediction mechanism is determined at design-time. At run-time, the prediction mechanism determines whether to switch from one system scenario to another based on dynamically changing system parameter values.

The exploitation step at design-time (in Figure D.1) applies further optimizations to each scenario, similarly to what would have been done to the static implementation without the scenario approach, e.g., simple loop transformations. The system knob settings of each scenario are decided here. At run-time, the exploitation step is simply the execution of the scenario.

In order to change the system from one set of run-time platform configuration parameters (system knobs) to another, a scenario switching mechanism must be implemented. The switching mechanism to be used at run-time is determined at design-time. This mechanism will switch scenario based on the output of the run-time prediction step.

An optional calibration configuration can be designed (at design-time) and used (at run-time) in order to fine-tune the decision making.

D.2.2 Control variables vs data variables

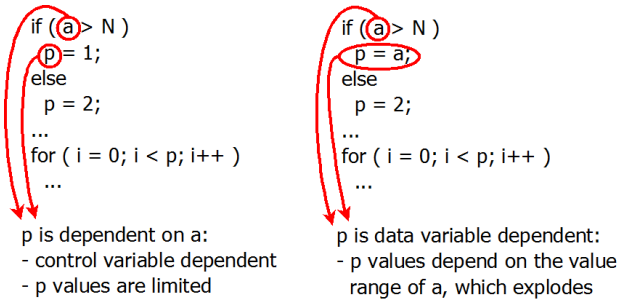


Figure D.2: Control vs data variable dependency

In order to better understand the challenge of data variable dependencies, Figure D.2 illustrates a simplified code snippet illustrating the difference between a control variable and data variable dependency.

The code snippet on the left in Figure D.2 illustrates a control variable dependency. The variable p in this case have a limited value range, and it will be relatively easy to monitor the application behavior if p is later used, for example, as a loop parameter.

The code snippet on the right shows a data variable dependent situation, where the value range of p equals the value range of a. If a is an input parameter, and p is used as a loop parameter in succeeding computations, then the loop size varies with the input parameter value range. Assuming the input parameter is a 16-bit integer value, then the value range of a will be equal to 2^{16}. This value range would simply produce too much run-time overhead for the scenario prediction and switching mechanisms.

D.3 Related work

Many researchers have optimized energy efficiency in embedded systems by proposing improved dynamic scheduling algorithms. Lee et al. [61] propose Leakage Control Earliest-Deadline-First (LC-EDF) and Leakage Control Dual Priority (LC-DP) for reducing leakage current in hard real-time systems. The first scheduling algorithm schedules tasks according to their deadlines and the second scheduling algorithm schedules tasks with soft deadlines together with periodic, sporadic and adaptive tasks. Both algorithms focus on maximizing the available idle time of the system, and hence reducing the leakage by setting the system in sleep mode during idle times. Irani et al. [51] propose 3-competitive offline and constant competitive ratio online algorithms for power saving while considering shutdown in combination with DVFS. The offline algorithms considers the best energy efficiency of all available power modes for a given task. The online algorithm uses the EDF algorithm to schedule arriving jobs in a way that minimizes total energy and meet their deadlines. Linwei et al. [79] uses a combination of DVFS and shutdown to minimize the overall energy consumption based on the latest arrival time of jobs. In a hard real-time system scheduled by the EDF strategy they reduce the energy consumption during idle intervals by utilizing DVFS to increase the speed of a task. Most of these approaches produce significant run-time analysis overhead. Awan et al. [10], propose a solution with significantly less run-time overhead. They combine an online and offline algorithm that selects the best feasible sleep mode based on the available time slack for a given workload. All these techniques have mainly one thing in common; scheduling of different tasks in combination with RTH or DVFS. Other researchers focus on machine-learning techniques to predict degradation in performance [89]. Such techniques are complex to implement on embedded platforms, and focus mainly on use cases with shared resources in multicore platforms.

A method for automatic identification of system scenarios that incorporate correlations between different parts of applications is proposed by Gheorghita et al. [40], [38]. When many-valued parameters are considered, the overhead for scenario prediction is significant, however. Hammari et al. [45] propose a method based on the projection of scenarios onto an RTS parameter space, where many-valued parameters are handled with the use of polyhedral techniques. These techniques are needed to fully automate the scenario analysis, but is not required for partly automated systems where the scenario information is inserted by the designer.

In general, the scenario prediction mechanism is highly dependent on how the identification step is performed, as indicated by previous work [40], [38], [45]. The authors in previous work conclude that the number of variables used in the prediction should be low. This is also the same for other identification approaches, e.g., the method of using loop-trip count profiling [82]. Previous work mainly focuses on control structures in applications with limited or no data variable dependency. When data dependent RTS-parameters dominates the application behavior, the value range increases significantly. Increased value range in the RTS parameter space usually increases the run-time prediction overhead.

Damavandpeyma et al. [24] use DVFS as a system knob, and propose a compact scenario aware data flow graph (SADF). Their approach assumes that the system scenarios have been clustered as part of the identification step, where the clustered result does not contain any many-valued parameters, and the number of control- and/or data-variables used in their analysis is limited. Their approach works well for control variables with limited dependency variation. Another approach is needed for data variable dependency where the data variables have a wide range of possible values.

The data variable challenge increases with the combination of multiple data-dependent RTS parameters and nested loops. Compared with the previous work described above we here combine the FSS framework from our previous work [126] with DVFS and RTH online within a running task to achieve significant energy reductions when a data variable dependency is present. The data variable dependency is handled by using inline functions that combine multiple RTS parameters, and classify different ranges as scenarios. Low overhead and globally placed detection equations are defined that can detect the required cycle count of the application. The output of these detection equations are input to our prediction mechanism. The prediction mechanism then compares its input to different ranges in a lookup table, before deciding which scenario to choose. Our approach is scalable

with increasing numbers of system knobs and an increased number of RTS-parameters within an application. In the following sections we will show how we were able to achieve significant energy reductions with our FSS framework implemented on a SAM4L microcontroller board.

D.4 Motivational example

Rapid advances in processor and wireless networking technology are opening up for a new class of streaming media applications for mobile hand-held devices such as mobile phones and tablets [73]. According to Mohapatra et al. [73], these devices have inadequate resources, such as lower processing power, memory, display capabilities, storage and limited battery lifetime as compared to desktop and laptop systems. Video communication via the internet on such devices have become more common in recent years through the use of programs such as Skype, Viber, and Hangouts. A major challenge with hand-held devices is that the power consumption increases significantly during a video call. The phone's camera is used to encode video and stream it over the internet continuously. At the same time, video from the caller is received and decoded continuously. Considering that video is streamed with a rate of 24-30 frames per second or more, the amount of simultaneous encoding and decoding is significant. One of the commonly known challenges is that the user perception of a video call can vary based on the internet connection speed, the processing limitations of the hand-held device, and battery level. An advanced codec such as H264/AVC is suitable for hand-held devices [91], and is widely deployed today. The H264/AVC codec is far more advanced than its predecessors, and introduce more dynamism, especially with varying input frame sizes. This dynamicity can be exploited using the system scenarios design methodology presented by Gheorghita et al. [39].

There exist proposals for video resizers today, such as the "Apparatus and method for resizing an image" [111], which rescales a camera's video frame size. A simple control logic could easily be added to change the frame size of a picture, such that the dynamic behavior of the internet connectivity could be exploited and used to decide the scale factor of the video before it is encoded and transmitted.

Figure D.3 shows the results of measurements we have performed of available internet bandwidth for a 500-second period on three different wireless networks. The upper graph shows the wireless local area network

(WIFI) at NTNU, while the middle and lower graphs shows measurements over mobile Long-Term Evolution (LTE) [95] and Wideband Code-Division Multiple-Access (WCDMA) [23] networks, respectively. The measurements are performed using the bandwidth measurement tool IPERF [1]. IPERF is used to measure available bandwidth between two IP-addresses. Our measurements are performed with 10 seconds intervals between a Sony Xperia Z3 android-based smartphone and a computer running Ubuntu. Figure D.3 clearly shows significant dynamicity in the available bandwidth, which we can exploit by using our FSS-framework [126]. In addition we calculate the moving average over the available bandwidth, as seen in Figure D.3.

We focus on the H264/AVC encoder control structure obtained from the Mediabench II website [60], where we adapt the platform's voltage and frequency according to the input frame size, assuming that a video resizer circuit rescales the input frames based on the moving average measurement of the available bandwidth in Figure D.3.

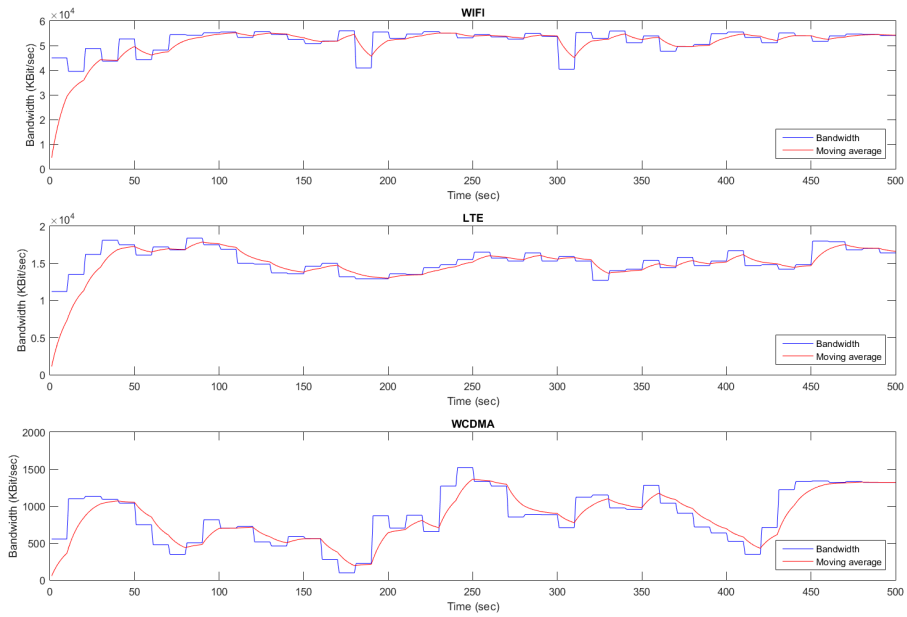


Figure D.3: Available bandwidth measurement and moving average over WIFI, LTE and WCDMA

D.5 Experimental setup

Our target domain is embedded systems such as mobile phone platforms where energy efficiency is the main constraint. Flora et al. [30] highlight the main hardware characteristics of such a system, containing relatively weak CPUs and limited memory. Their survey show that good architectures are even more important for mobile phones than for desktops. The SAM4L board from Atmel [8] is a representative example of a mobile platform because it contains an energy friendly CPU with limited on-chip SRAM. One advantage of the SAM4L-board is the application control of active voltage states, making it possible to dynamically tune the hardware for the running application.

During our experiments we measure the live current consumption and core voltage on the SAM4L board using a NI myDAQ measurement device [77]. The processor core voltage is measured directly on the vcc_mcu_core pin of the SAM4L microcontroller relative to the board ground, while the SAM4L microcontroller's total current consumption is measured through available measurement pins on the board. Time stamped data is logged with a locally developed LabVIEW program, and used together with our voltage and current measurements, to find the energy consumption of the H264/AVC encoder control structure. The single-core FSS framework is used to implement the H264/AVC encoder control structure with system scenarios.

Our version of the SAM4L board contains the ATSAM4LC4CA microcontroller. The microcontroller contains an ARM Cortex M4 processor, and supports two different run modes. RUN0 operates with a core voltage of 1.8V, while RUN1 operates at 1.2V. The microcontroller also supports various low power states, i.e., sleep modes, where the processor clock is turned off, and multiple oscillators with different frequency ranges. Table D.1 shows all sleep modes used in our experiments and their corresponding measured power consumption, overhead and wakeup times.

For the scope of our research, we have selected the internal oscillators RCFAST (12 MHz) and RC80M (80 MHz) as the main clock sources for our DVFS settings. Since the ARM Cortex M4 processor only supports frequencies up to 48 MHz, the RC80M oscillator is clock divided to 40 MHz when it is selected as the main clock source. Hence, our DVFS settings are PS0 (1.8 V and 40 MHz) and PS1 (1.2 V and 12 MHz).

Based on the SAM4L board datasheet, the DVFS is triggered by a write to configuration registers after a write to protective lock registers. In other

Table D.1: Run/Sleep modes in our SAM4L-board from Atmel

Run/Sleep mode	Measured power consumption (μW)	Wakeup time (incl. setup time)
RUN0 (heavy load)	29 865	N/A
RUN0 (idle)	25 340	N/A
sleep00	14 010	83 clk @ 40 MHz
sleep01	10 370	659 clk @ 40 MHz
sleep02	5 460	664 clk @ 40 MHz
sleep03	79	3907 clk @ 40 MHz
wait0	12	5669 clk @ 40 MHz
retention0	5.8	5997 clk @ 40 MHz
RUN1 (heavy load)	5 107	N/A
RUN1 (idle)	4 013	N/A
sleep10	2 440	88 clk @ 12 MHz
sleep11	1 830	390 clk @ 12 MHz
sleep12	1 460	395 clk @ 12 MHz
sleep13	51	1057 clk @ 12 MHz
wait1	5.9	1728 clk @ 12 MHz
retention1	4.4	6703 clk @ 40 MHz 1786 clk @ 12 MHz

words, a switch is triggered after a few clock cycles. We have measured the switching time between our two clock settings with an oscilloscope. The switch takes approximately 15 clock cycles (at 12 MHz) when both oscillators are on continuously, and approximately 50 clock cycles (at 12 MHz) if the currently unused oscillator is powered down. The added power consumption by having both oscillators enabled is negligible because unused oscillators that are idling does not contribute to any measurable power consumption. Therefore, both oscillators are kept enabled in our measurements for a significantly reduced switching time.

We have measured the total static power to be less than 16% of the total power consumption in PS0, and less than 29% in PS1. The static power consumption on the SAM4L board is approximated by measuring the total power consumption for two different frequencies in each power mode. With these four equations we estimate the effective capacitance and static energies under the assumption that the activity factor (α) is equal to 0.5 in Equation D.1.

$$P_{\text{Total}} = P_{\text{Static}} + P_{\text{Dynamic}} = P_{\text{Static}} + \alpha C V^2 f \tag{D.1}$$

When performing dynamic voltage scaling on the SAM4L board, the main clock frequency supported in the new voltage level has to be configured first. According to the datasheet, the 1.2 V (PS1) configuration does not support frequencies larger than 12 MHz. The frequency setting must therefore be switched from 40 MHz to 12 MHz before the voltage switch from 1.8 V (PS0) to 1.2 V (PS1). When changing configuration from PS1 to PS0 the frequency must be scaled after the voltage switch to maintain stability.

In addition, the output from the voltage regulator has to be stable after a voltage switch before the CPU can continue processing. After a switch from PS1 to PS0 we have to wait for a status flag, which signals that the voltage level has stabilized. The CPU is therefore stalled while waiting for this flag to be set in the SAM4L board. However, when a voltage switch is triggered from PS0 to PS1 then waiting for this flag is not necessary before continuing the CPU execution. As described by Atmel Support, when switching from a higher voltage level to a lower level, the voltage regulator is stable and will regulate fast enough to continue execution in succeeding cycles. The measured current consumption will gradually decrease after this switch until the charge on the decoupling capacitor connected to the core voltage has been consumed down to its PS1 level. The power delivered from the regulator on the other hand is equal to the PS1 level immediately after the voltage switch.

Our SAM4L board came initially with a 22 µF decoupling capacitor connected between the core voltage input and ground. The large decoupling capacitor resulted in extremely long switching times. This was confirmed by Atmel Support, and they recommended a change to 4.7 µF to guarantee that all modules would function properly. Since we are only interested in the CPU, we measured the DVFS switching times with lower decoupling capacitors. Our experiments showed that the CPU functioned correctly using a 1 µF decoupling capacitor, giving a DVFS switching time comparable to state-of-the-art processors. Table D.2 shows the switching times and energy overhead when switching from PS0 to PS1 and vice versa. The energy overhead is found by measuring the time between the last instruction before and the first instruction after the DVFS switch, before multiplying this time with the power consumption for the run mode before the switch. An oscilloscope was used for the timing measurement.

Table D.2: DVFS switching times between PS0 and PS1.

From	To	Cycles @ 12 MHz	Switching time (μS)	Overhead Energy (μJ)
PS1	PS0	197	16.40	0.0623
PS0	PS1	15	1.25	0.0281

D.6 Scenario identification methodology

The application behavior of streaming media applications must be profiled with different input data in order to detect different run-time situations (RTSs) that occur while the application is running. Our scenario identification approach takes into account available and relevant application domain parameters. This is typically the stream of input data, the application code handling this data, long input sequence which we profile in order to identify the dynamism in the workload, application run time due to while loops/conditions, data variable dependency, and storage requirements.

The H264/AVC encoder source code was obtained from the Mediabench II website [60]. This model is profiled at design-time in order to explore the dynamic range of loop iteration count variations for various input frame sizes. The H264/AVC encoder control structure is extracted from the Mediabench II source code, and modeled under the assumption that all memory accesses take one clock cycle. In [124] all instructions, i.e., calculations and memory accesses, are exchanged with ”no operation” (NOP) instructions, which makes the model highly optimistic. In the current work we exchange these NOPs with single cycle memory access to SRAM in order to make the model more realistic. These improvements are discussed in further detail in Section D.7.2.

In addition, different voltage and frequency settings are profiled in order to find optimal DVFS configurations in combination with RTH. Our profiled results show that the optimal energy consumption is achieved if the maximum available frequency setting for each voltage level is selected. This results in two scenarios, PS0 (1.8V and 40 MHz) and PS1 (1.2V and 12 MHz).

Table D.3 shows an evenly distributed selection of true 16:9 resolutions that we have selected, which our SAM4L board can process within a one second time frame using our extracted model. It also shows the corresponding scenario selections and calculated run times, based on our available power modes in the SAM4L board. In Table D.3, the Scenario* column represents

the scenario distribution for a platform with more available voltage settings, and is discussed in Section D.8.

The largest frame size the platform can handle from our selection is measured to be 816 x 459. The required run time (in clock cycles) for each frame size is estimated by counting all loop iterations and multiplying with all instructions within each loop based on the program dis-assembly of the extracted model. These numbers are compared to the measured run time for each frame size. From the measured results, we observed that the correct cycle count could be approximated by multiplying the number of all loop iterations for any input frame size with a factor of 4.22. This factor corresponds to the additional assembly instructions needed to determine the loop length and initiate for-loops before they are executed. The number of all loop iterations is calculated as shown in Listing D.1. The total number of iterations are combined together with other parameters for all the nested loops in the application, resulting in the loopcnt variable. This loopcnt variable is multiplied by the approximated factor of 4.22. The separation of the approximated factor allows for easy portability of the scenario prediction mechanism to other platforms, because other platforms might implement the loop handling instructions differently. This difference would result in a higher or lower factor. The largest frame processing time is therefore approximated to 34.7 million clock cycles in an un-optimized setting where no pipelines are utilized. We use this frame size processing time as our worst-case scenario, and we exploit it in combination with DVFS and RTH for lower frame sizes. We benefit from DVFS if the processing time of a smaller frame size with a lower frequency setting is less than the processing time of our worst-case frame size. Therefore, we have to take our available system knobs into consideration when we classify our system scenarios.

Our scenario selection will be dependent on the platform's running frequency. In PS1 the platform is processing a frame 70% slower than in PS0. Therefore, frame sizes that need more than 30% of the run time of our worst-case frame size are processed with PS0 (Scenario 0), and smaller frame sizes are processed with the PS1 configuration (Scenario 1).

Our arguments above are generalized in Equation D.2, where we set the Scenario 1 upper limit for the maximum frame processing time in PS1 to be equal to the frame processing time of our largest frame (816 x 459) in PS0. The Scenario 1 upper limit is calculated as shown in Equation D.3. In Equation D.2 and D.3, $T_{\text{WC frame}}$ is the execution time in seconds of the worst-case frame size. T_{sc_1} is the maximum allowed execution time for Scenario 1 in seconds. LC_{sc_0} and LC_{sc_1} are the total number of loop iterations

Table D.3: Scenario assignment based on run time

Input frame size	Run time (million cycles)	Scenario	Scenario*
816 x 459	34.70	0	0
752 x 423	29.54	0	0
688 x 387	24.79	0	1
624 x 351	20.46	0	2
560 x 315	16.54	0	2
496 x 279	13.04	0	3
432 x 243	9.96	1	4
368 x 207	7.29	1	4
288 x 162	4.54	1	5
224 x 126	2.80	1	5

for a specific frame in PS0 or PS1, respectively. β is the approximated scale factor used to get the total number of clock cycles, and f_{sc_0} is the platform's main clock frequency when Scenario 0 is active. In other words, if the required frame processing time is above 30% of our worst-case frame processing time, then the platform switches from Scenario 1 to 0. Based on Equation D.2, we find the Scenario 1 limit in Equation D.3 without involving the β scale factor. This simplification makes our method portable to other platforms with a different β scale factor. Equation D.3 shows the limit 2,466,807, which corresponds to 10.4 million clock cycles of actual run time (after multiplying with the 4.22 approximated scale factor β). I.e., all frame sizes with a run time above 10.4 million clock cycles are assigned to Scenario 0 in Table D.3.

$$T_{\text{WC frame}} = T_{\text{sc}_1} = \frac{\beta \cdot LC_{\text{sc}_0}}{f_{\text{sc}_0}} = \frac{\beta \cdot LC_{\text{sc}_1}}{0.3 \cdot f_{\text{sc}_0}} \tag{D.2}$$

$$LC_{\text{sc}_1} = 0.3 \cdot LC_{\text{sc}_0} = 0.3 \cdot 8,222,691 = 2,466,807 \tag{D.3}$$

D.7 Prediction and switching mechanism

Our generic framework for system scenarios (FSS) [126] is used to implement the scenario mechanisms as shown, e.g., in the left hand side of Figure D.5 on page 170.

The FSS framework separates the application software from the scenario mechanisms to make the framework portable to other platforms. This separation allows us to combine software and hardware implementations of the system scenario mechanisms with minimal modifications to the original application. I.e., the application monitoring parts of the scenario mechanism is defined by the Application Monitoring Unit (AMU). Lightweight and inline AMU functions are injected in the application at different checkpoints to collect RTS information, which will be used to predict or detect future system scenarios. The AMU then calls the Platform Adaption Manager (PAM), which performs the actual platform reconfiguration for the detected or predicted scenario.

The AMU is separated from the PAM because the PAM contains functions and non-portable configurations which are specific to the platform. If the FSS framework is ported to another platform, then the PAM functions and configurations must be configured for the new platform as well. The AMU functions would in this case stay the same and is platform independent. Alternatively, the AMU could be implemented partly in software and partly in hardware to speed up the scenario prediction, which may be needed in multi-core systems using system scenarios if the communication between off-chip modules takes too long time with a software-only approach. The hardware implementations of our FSS framework is out of the scope of this paper.

All streaming media applications where the application run time is dependent on the input frame size would benefit from this approach because the nested loop structures in such applications normally have a strong dependency on the frame size, i.e., larger frame sizes results in significantly increasing the storage requirements and the application run time.

The input data variable dependency of H264/AVC encoder control structure is modeled as part of our application monitoring unit (AMU) in the FSS framework [126]. The input frame sizes of the H264/AVC encoder is used directly in the calculations to determine succeeding loop sizes and memory accesses. This dependency and how we exploit these calculations is discussed in Section D.7.1. In Section D.7.2 and D.7.3 we present two new contributions to this paper, where our AMU mechanisms are improved by the use of loop transformations (Section D.7.2), and a sleep mode management is implemented to select the lowest power consuming sleep mode allowed from the calculated slack time after a processed frame (Section D.7.3).

D.7.1 Input data variable dependency prediction

The input data variable dependency is shown in Listing D.1, where inline application dependent equations are pre-calculated (lines 2-7), before a global lookup-table is updated with the combined RTS parameter (line 10) as part of the overall AMU. This RTS parameter is then read by the AMU_1 function, which detects the next scenario (line 13). The application dependent calculations in lines 2-6 correspond to succeeding (nested) for- and/or while-loops in the encoder control structure, which are combined together in line 7 to estimate the required frame processing time in terms of loop iterations. In our particular case it would have been enough to compare the loopcnt variable with the threshold between Scenario 0 and 1 directly, and use the result to determine the next scenario.

```
1  // Functions for monitoring RTS parameters
2  long loopcnt1 = (size_y + img_pad_size + img_pad_size) * (2 + (3*(
       size_x + img_pad_size + img_pad_size)));
3  long loopcnt2 = ((size_x + 2 * img_pad_size) * 2) * (5 + ((3*(maxy
       - 2 - 2)) + (3*(ypadded_size - maxy + 2))));
4  long loopcnt3 = ((je2 + 2)/2) * (2 + (((ie2 - 2)/2) + 2));
5  long loopcnt4 = (ie2 + 2) * (((je2 - 2)/2) + 2);
6  long loopcnt5 = (size_y/4) * (1 + (6*(size_x/4)));
7  long loopcnt = loopcnt1 + loopcnt2 + loopcnt3 + loopcnt4 +
       loopcnt5;
8
9  // Update combined RTS parameter
10 (g_rts_data)[0] = loopcnt;
11
12 // Scenario detection/prediction
13 AMU_1(scenario_old, scenario, rts_data[], g_pareto_lc[]);
```

Listing D.1: Functions combining data-dependent RTS parameters

However, we have generalized this comparison by using the AMU_1 function. This lightweight inline function compares the global RTS lookup-table values with global scenario lookup table values in order to determine which scenario the platform will adapt to. This generalization takes into account multiple scenario parameters and RTS parameters. The use of lookup tables would simplify an extension to multiple independent RTS parameters and/or scenarios. The values in the global scenario lookup table is found from our identification step, from Equation D.3. If the value of the global RTS lookup table is greater than the value in the global scenario lookup table, then the platform changes its configurations to PS0 if it is in PS1, and vice versa. The PAM function is then called by the AMU and performs

the platform re-configuration using the DVFS system knob as described in Section D.5. Figure D.4 shows the rts_data content for different input frame sizes which corresponds to Table D.3 after multiplying the values with the β-factor.

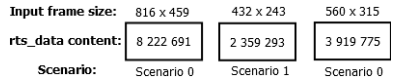


Figure D.4: Contents of rts_data for different input frame sizes.

D.7.2 Improving the scenario energy overhead

Our initial approach as described in Section D.7.1 resulted in an average energy overhead of 8.5% for the 500-second sequence [124]. These measurements were taken using an extreme conservative approach, however, where the scenario mechanism is triggered between each frame even if the scenario for the next frame is the same as the previous one. In this paper, we have improved our mechanism to trigger a DVFS switch only if the previous scenario is different from the predicted scenario. This improvement reduced the average energy overhead to 7.3%.

Further investigations to our encoder control structure reveled that our NOP instructions only delays further processing on the SAM4L board. Therefore, the measured power consumption of our encoded frames were equal to the boards idle power consumption with no read and/or writes to memory, which is very optimistic. We therefore exchanged the NOPs in our encoder control structure with one cycle read and write operations to the board's on-chip SRAM. This modification resulted in more realistic power measurements for video encoding compared to only using NOPs. We measured the energy consumption for our 500-second streams with the new encoder control structure model, which resulted in a 2.4% energy overhead using our scenario mechanisms. This overhead corresponds to the calculations performed in Listing D.1.

Finally, we observed that parts of these calculations are performed before each for-loop in the code, making our pre-calculations in Listing D.1 redundant. Therefore, simple loop transformations are applied, where the loop size calculation performed before each for-loop is performed before executing the encoder control structure. As a result, our scenario mechanisms could benefit from this improvement and eliminate redundant calculations.

The average energy overhead of our scenario mechanisms are measured to be 0.7% after the simple loop transformations, which we consider negligible. Figure D.5 illustrates the changes after the simple loop transformations.

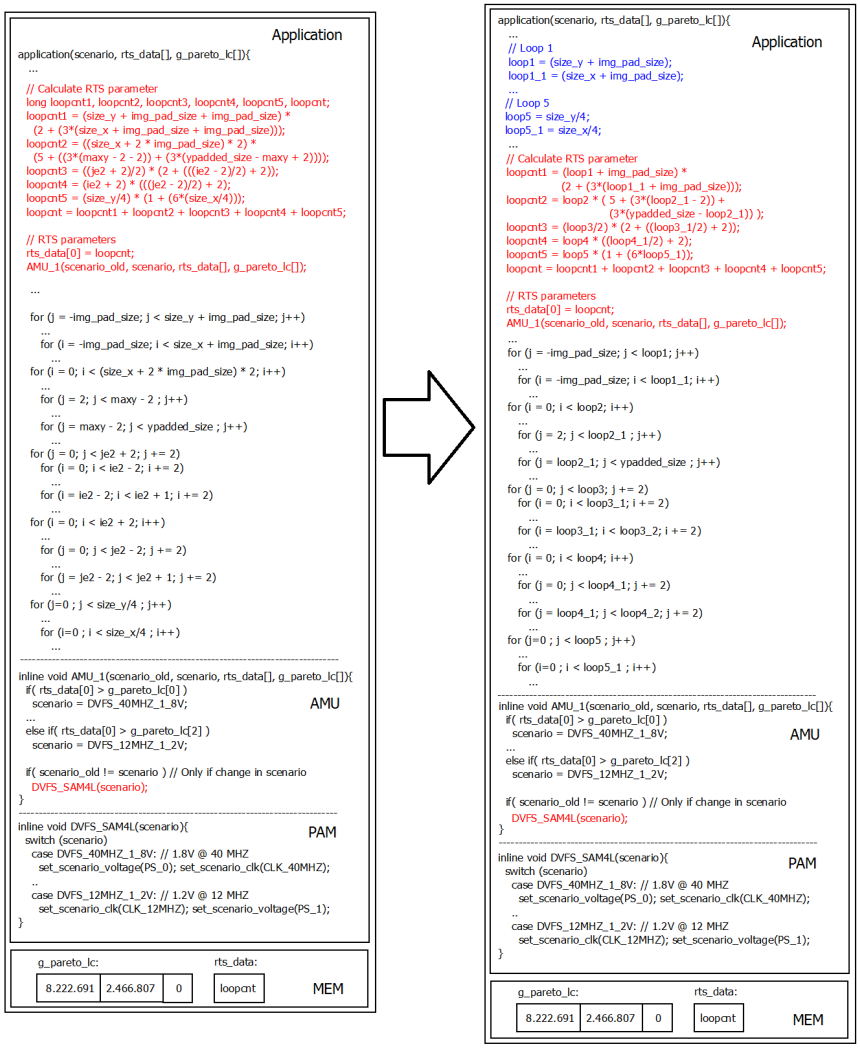


Figure D.5: Changes to the code after simple loop transforms

D.7.3 Introducing sleep mode management with system scenarios

Our SAM4L board contains different sleep modes as shown in Table D.1 on page 162. Other work, such as the RTH energy saving strategy by Awan et al. [10], have exploited available sleep modes to utilize RTH more energy efficiently. Their solution is based on an offline and online approach, where they find the most efficient sleep modes offline for a set of jobs based on the maximum time interval, t_1, for which the processor may be enforced in a sleep state without causing any task to miss its deadline under worst-case assumptions. Their online algorithms compute the sleep time by calculating the available slack present at run-time. The most efficient sleep state is then selected online based on the sleep time calculated online and the pre-calculated t_1 interval. Their approach works well for sporadic tasks where jobs can start at any time with negligible online overhead.

Our system scenario approach enables us to simplify the solution by Awan et al. significantly by exploiting our pre-calculated loop execution times from our AMU_1 mechanism. Instead of computing the slack online, we pre-compute the total available frame cycles (TOTAL_FRAME_CYCLES in Listing D.2) offline for each scenario (PS0 and PS1) based on what we know about our system from the scenario identification step. The stored wakeup time, including the setup time, are then summed with the calculated loopcnt value (g_rth_lc and g_rts_data in Listing D.2) and compared to the pre-calculated total frame cycles available online. Our solution then selects the lowest power consuming sleep mode if our estimated total execution time is less than the slack time available. The different sleep power consumptions, setup times and wakeup times are measured on our SAM4L board, and included in Table D.1. All these wakeup time values (including the setup time) from Table D.1 are stored in the g_rth_lc global lookup table. This lookup table is read by the AMU_2 mechanism shown in Listing D.2.

AMU_2 reduces the online overhead complexity from Awan et al. to an if-else comparison. Starting with the lowest power consuming sleep mode, the AMU_2 compares the slack available with the sum of loopcnt and the corresponding sleep wakeup time until it finds a solution. If no solution is found the system stays idle until the arrival of the next frame. For simplicity, the AMU_2 mechanism is not shown in Figure D.5. The AMU_2 is executed inline after the nested for-loops in Figure D.5, and the selected sleep mode is then used in the RTH mechanism directly. The AMU_2 mechanism is

thus a separate inline function added in addition to the AMU_1 function in our AMU part in Figure D.5.

```
1  static inline void AMU_2(volatile int *scenario){
2    if(*scenario == DVFS_40MHZ_1_8V){ // Current scenario = PS0
3      if( ((g_rts_data)[0] + (g_rth_lc)[5]) <
           TOTAL_FRAME_CYCLES_40MHZ )
4        (g_rts_data)[1] = (long) PS0_RET1; // loopcnt + RET1 wakeup
             < Slack
5      else if( ((g_rts_data)[0] + (g_rth_lc)[4]) <
           TOTAL_FRAME_CYCLES_40MHZ )
6        (g_rts_data)[1] = (long) PS0_RET0; // loopcnt + RET0 wakeup
             < Slack
7  //...
8      else if( ((g_rts_data)[0] + (g_rth_lc)[0]) <
           TOTAL_FRAME_CYCLES_40MHZ )
9        (g_rts_data)[1] = (long) PS0_SLEEP00; // loopcnt + SLEEP00
             wakeup < Slack
10     else
11       (g_rts_data)[1] = (long) PS0_RUN0; // No RTH
12   }else if(*scenario == DVFS_12MHZ_1_2V)\{ // Current scenario =
       PS1
13     if( ((g_rts_data)[0] + (g_rth_lc)[10]) <
           TOTAL_FRAME_CYCLES_12MHZ )
14       (g_rts_data)[1] = (long) PS1_RET1; // loopcnt + RET1 wakeup
             < Slack
15 //...
16     else if( ((g_rts_data)[0] + (g_rth_lc)[6]) <
           TOTAL_FRAME_CYCLES_12MHZ  )
17       (g_rts_data)[1] = (long) PS1_SLEEP10; // loopcnt + SLEEP10
             wakeup < Slack
18     else
19       (g_rts_data)[1] = (long) PS1_RUN1; // No RTH
20   }
21 }
```

Listing D.2: AMU_2 structure

D.8 Experimental results

In our experiments we simulate a realistic video sequence of 500 seconds using the experimental setup presented in Section D.5, assuming the video sequence can be processed with 25 ARM Cortex M4 cores in parallel in order to meet the timing requirements for a frame rate of 25 frames per second (fps). The parallel cores are assumed able to process each frame individually with negligible synchronization overhead. This solution requires buffering and software pipelining of the frames to maintain forward frame dependency

requirements. Other researchers have suggested similar approaches where their focus targets the parallelization of the H264/AVC encoder [129], [105]. We do not target general high-performance multi-core platforms. Instead, we aim at a subset of applications with non-interacting concurrently operating tasks. Moreover, we target the embedded multi-core domain, where cache coherence and complex bus protocols are avoided by limiting the functionality of the cores. In particular, we limit ourselves to totally independent cores working on tasks that do not have any interaction, where we believe this extrapolation is valid.

Our scenario implementation is independent of this parallelization, and can be used on a faster processor for a single-core solution. After each frame is processed, we force the core into a low power retention mode, where the measured energy consumption is negligible. Before entering retention mode, a configurable timer is set to wake up the system just before the next frame arrives.

Our video sequence is measured using all three network measurements in Figure D.3. To process a frame, we specify a threshold requirement, bw_th, for the available bandwidth to be equal to at least five times (5x) the required bandwidth for a frame rate of 25 fps with a particular frame size. This threshold allows other activity to continue on the network, and avoids using up all the available bandwidth. With lower thresholds, the limited number of available voltage levels in the SAM4L board would always force the system to use the most power consuming voltage setting. The 5x threshold thus enables us to demonstrate how our methodology exploits the dynamism in the available bandwidth. Our simulated 500-second video sequence is created by combining different frame sizes from Table D.3 in a sequence that fits with the available bandwidth of the measured networks in Figure D.3, and our threshold requirements.

D.8.1 Energy reductions compared to a RTH approach without system scenarios

The application code is compiled in a conventional way with compiler optimization parameter O3. When we optimize the application code with the O3 configuration, we assume it is a good enough optimization for the exploitation step in the system scenario design methodology presented by Gheorghita et al. [39]. The energy consumption is measured in real time using our experimental setup, with and without the scenario mechanism.

The overhead energy consumption of our scenario mechanism is measured by forcing the scenario mechanism to select PS0 each time a scenario switch takes place. This approach would not reduce the energy, but would trigger the scenario mechanisms in our AMU and PAM modules. The results are then compared to the energy consumption without using the scenario mechanism. The difference equals the scenario mechanism overhead.

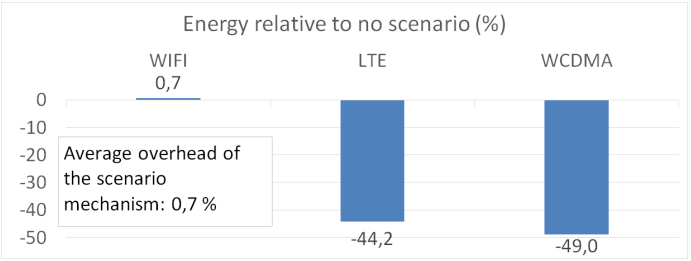


Figure D.6: Energy consumption relative to no system scenarios.

Figure D.6 shows the energy reductions relative to the energy consumed by a no scenario implementation utilizing a race to halt (RTH) setup in each of the measured networks of Figure D.3. A retention mode is used when the system is halted, which consumes 4.4 μW . The average energy overhead of our scenario mechanisms are measured to be 0.7% after our improvements discussed in Section D.7.2. We consider this overhead negligible.

The lowest energy consuming situation occurs when all frame sizes are suitable for the PS1 configurations, such as for the WCDMA network. We obtain in this situation up to 49% energy reduction as compared to a RTH solution without the use of the scenario mechanism. These results include the average scenario mechanism overhead of 0.7% energy increase for the 500-second sequence.

When the available bandwidth is higher than our threshold for our largest frame size, then the scenario mechanism is not needed. It simulates our most energy-consuming situation, where the system is running constantly in the PS0 platform configuration. In this situation the energy consumption does not increase more than the measured scenario overhead.

The LTE results are interesting because the energy reduction for this network is determined by how long the system is in the PS1 configuration. In our measurement, the platform runs with the PS1 configuration 93.8% of the time for this LTE network measurement (with bw_th = 5x), and runs in the PS0 for the remaining sequence. The energy savings would decrease if the platform runs in PS0 for longer time periods.

Our results meets Hillestad's [48] requirement for perceptual video quality, as mentioned in Section D.1, while maintaining the frame rate.

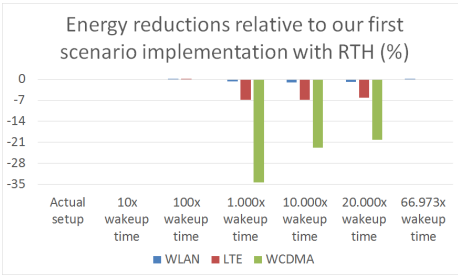
D.8.2 Exploiting available sleep modes based on remaining available slack time

The energy overhead of our scenario mechanism is measured using our streams with and without the AMU_2 and its corresponding lookup tables for the system scenario sleep mode management. The lookup tables for the scenario sleep mode management are based on Tables D.1 and D.2. The added overhead of the AMU_2 is measured to be less than 0.1% and therefore we consider the added overhead negligible. With our experimental setup all frames, even the largest one, are processed and finished with enough slack time for our AMU_2 to select the lowest power consuming sleep mode. The sleep mode scenarios consequently does not give any energy reduction. If, on the other hand, the slack time of different frame sizes would be in the same range as wakeup times, we would benefit from selecting the optimal sleep mode. To illustrate this claim, we have simulated our recorded streams assuming the wakeup time of each sleep mode were a factor N slower, where N equals 1x, 10x, 100x, 1.000x, 10.000x, 20.000x and 66.973x. If we were actively using other types of memories (e.g. off-chip memories) then the wakeup times for these memories are more likely to be at least 1.000 times slower, which makes our assumptions realistic. If the slack time would be significantly shorter than what we have in our experiment, such that our most power efficient sleep mode could not be selected, then we also expect similar energy reductions. If the wakeup time is too long, which is the case if $N > 66.973x$, the system would never be able to go into a sleep mode. Based on this we see that our application benefits from the AMU_2 if its task slack times dynamically changes between different processing intervals and between different sleep mode wakeup times.

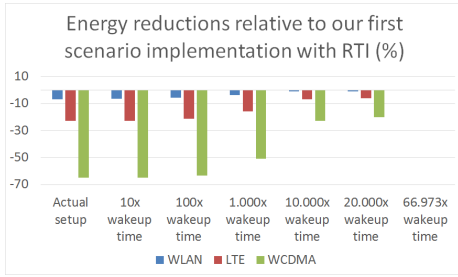
Without AMU_2 the RTH mechanism cannot adapt the sleep mode selection according to dynamically changing slack times. Instead the least power consuming sleep mode that still is able to wake up in time after the largest frame size has to be selected in all situations. This sleep mode will be increasingly more power consuming when N increases, and for $N > 66.973x$ the wakeup times of all sleep modes will be too long. The AMU_2 mechanism exploit lower power consuming sleep modes when smaller frames are processed, instead of only selecting the safest sleep mode. The estimated

energy reductions compared to not having the AMU_2 (and corresponding lookup tables) available are shown in Figure D.7(a). Figure D.7(b) shows the energy reductions compared to a Race-To-Idle (RTI) setup where the processor does not go to sleep or halt after each processed frame.

Figure D.7 clearly shows that we would start to get significant energy reductions if the wakeup times for our sleep modes were approximately 1.000 times slower. In this case, our AMU_2 approach would result in up to 34.3% energy reduction. The effect decreases when N increases since fewer sleep modes are within the range of the slack times. In our simulation, we observe that if the wakeup times were more than 66.973 times slower, then even the sleep mode with the shortest wakeup time would become too slow. In this situation our processor would stay idle until the next set of frames arrive. When the wakeup times are only 100 times slower, we observe a slight increase in energy consumption. This effect is caused by the fact that our AMU_2 mechanism selects sleep mode sleep03 over sleep02 because it has lower power consumption. On the other hand, the sleep03 wakeup time is significantly larger than the sleep02 wakeup time as shown in Table D.1, and this large difference affects the total consumed energy consumption. If the wakeup times would be more evenly scaled between the different sleep modes, then this effect would not appear.



(a) Sleep scenarios compared to scenario Race-to-Halt (RTH).



(b) Sleep scenarios compared to Race-to-Idle (RTI).

Figure D.7: Potential improvements in energy consumption with slower wakeup times.

D.8.3 Introducing more DVFS states into our scenario mechanisms

Our SAM4L board configuration capability is limited to two voltage states. If more voltage states were available, which is the case for modern Intel processors for example, the energy consumption is expected to decrease further since each frame size could correspond to a separate DVFS setting.

According to Ramalingam et al. [90], the Sakurai-Newton (SN) delay approximation is a widely used closed-form delay metric for the CMOS gates because of simplicity and reasonable accuracy. This approximation is good when the supply voltage is significantly larger than the threshold voltage. We use this delay approximation model to get the relationship between maximum voltage and frequency for a single gate as shown in Equation D.4, where V_{DD} is the supply voltage, V_t is the threshold voltage, C is the load capacitance, and $k = (\frac{W}{L})\mu_n C_{ox}$ (logic design and fabrication characteristics factors). The α-power dependence is the velocity saturation index, which is a parameter between 1 and 2. According to Gonzalez et al. [41], the velocity saturation index is likely to be 1.3-1.5 in a 0.25μm technology.

$$\frac{1}{\tau} \approx f \approx k \cdot \frac{(V_{DD} - V_t)^\alpha}{C \cdot V_{DD}} \tag{D.4}$$

To determine the necessary change in operating frequency when voltage is scaled, we observe that Equation D.4 is close to linear when $\alpha \approx 1.5$. Measured voltage-frequency curves from a 32nm bulk process [55] and a 28nm FD-SOI process [130] also show a roughly linear relationship between voltage and frequency (the frequency scale in [55] is logarithmic). Therefore, we will assume a linear frequency-voltage dependence when the supply voltage is significantly higher than the threshold voltage.

To demonstrate the added gain of more DVFS settings, we have extrapolated our available DVFS settings with six evenly spaced voltage and frequency levels based on our assumption of linear frequency-voltage dependence, ranging from 1.05 V to 1.8 V and from 6 MHz to 40 MHz, respectively.

The corresponding scenario assignment for the different frame sizes are shown in the scenario* column in Table D.3. Figure D.8 shows all scenario configurations from scenario 0 (1.8V/40MHz) to scenario 5 (1.05V/6MHz). We simulated our video sequence by manually applying the new scenario selection based on the LTE bandwidth measurement shown in Figure D.3, and calculating the dynamic energy. A lower threshold value, equal to four times the required bandwidth (bw_th = 4x instead of 5x), is selected to uti-

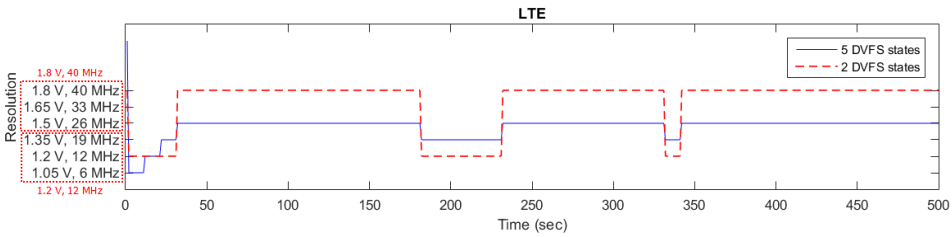


Figure D.8: DVFS state transitions during the 500-second video clip.

lize better our extrapolated DVFS settings. The increased number of DVFS settings allows the system to use a lower threshold for the bandwidth requirement to process a particular frame. Our simulated result based on extrapolated DVFS settings is shown in Figure D.8 for the LTE network. From this figure, we observe that the system switches to a higher frequency five times, and it scales down three times. Our existing solution with two DVFS settings would switch back and forth between two settings 6 times in total. Taking Equation D.1 and the energy overhead we measured from Table D.2 into consideration, our extrapolated results indicate that the dynamic energy is now reduced 59% compared to not having scenarios. A multi-core extension of our scenario mechanisms needs to address cache coherence and on-chip communication challenges in addition to the scenario selection at run-time. This extension is out of the scope of this paper.

D.9 Conclusion

Our framework for systems scenario based designs (FSS) is combined with race to halt (RTH) and dynamic voltage and frequency scaling (DVFS) in order to achieve significant energy reductions for an H264/AVC encoder control structure. We measure our results on a modified SAM4L board from Atmel, which enables online DVFS tuning with switching times comparable to state-of-the-art switching times.

System scenarios are identified based on data-dependent control variables, varying with the input frame size. These scenarios are predicted by using inline estimation functions for the expected execution time. These functions are executed at run time, and are compared to a predefined set of ranges for the scenario prediction with minimum overhead. Our switching mechanism then re-configures the platforms voltage and frequency accord-

ingly.

We measure the available bandwidth in WIFI, LTE, and WCDMA networks for 500 seconds. A video stream consisting of different frame sizes is then composed based on the measured network data and a threshold requirement. Compared to a no system scenario solution with RTH we achieve up to 49% energy reduction for the encoded streams, including a 0.7% average energy overhead, which is negligible. This average energy overhead is a result of optimal tuning and simple loop transformations, compared to our initial approach using conservative tuning having 8.5% average energy overhead.

The perceptual video quality and frame rate is maintained, and we show how we could reduce the energy consumption even further if a few more voltage levels are available as compared to only two available voltage levels.

We also show how existing sleep modes can be exploited in order to gain up to 34.3% energy reduction if the sleep wakeup times were 1000 times slower than our actual board. Our assumption for 1000 times slower wakeup times are representative for solutions dependent on off-chip memories or other dependent modules that require long wakeup times. We expect this assumption also to hold if the slack time of the processed task is in the range of the sleep wakeup times.

Furthermore, we expect similar results to be obtained by other applications with similar characteristics. The techniques for scenario identification and switching presented here enables efficient use of our framework for system scenarios, especially for applications where the dynamic behavior is determined by many-valued data variables.

APPENDIX

E

FAST AND ACCURATE DVFS IMPLEMENTATION IN GEM5 USING SYSTEM CALL EMULATION MODE

Yahya H. Yassin, Magnus Jahre, Snorre Aunet,
Per Gunnar Kjeldsberg and Francky Catthoor
IEEE Transactions on Very Large Scale Integration (VLSI) Systems
Submitted in 2017
(Currently under review)

Abstract

Stringent power budgets in battery powered mobile platforms have led to the development of energy saving techniques such as Dynamic Voltage and Frequency scaling (DVFS). In this paper, we propose a novel non-intrusive application controlled DVFS management implementation in the GEM5 simulator, used with GEM5's system call emulation mode. We also propose a novel architecture independent energy model based on classification of different measurable workload classes and power measurements from real HW. Our energy model is calibrated with power measurements from a SAM4L microcontroller board, containing an ARM Cortex M4 processor. Together with the GEM5 output statistics, this model estimates the total energy consumption of our simulated system accurately. The results from our modified GEM5 simulator are validated with two different, representative media applications. After correction of systematic offset errors, our results deviate with less than 4% compared to measurements from the SAM4L board. Our contributions in this paper can easily be tailored to other in-order processor models in GEM5, and will therefore enable system architects to explore new techniques and compare the improvements relative to existing architectures.

E.1 Introduction

Significant technology advancements in embedded systems have enabled the growth of features in advanced applications used in mobile computing systems [72]. These trends have resulted in a power management challenge on battery powered mobile platforms. Today's mobile platforms have stringent power budgets, such as 1-2 W, while wearable ultra low power embedded systems have even more stringent power budgets (in the order of a few milli watts) [36].

Different energy saving approaches have been proposed by researchers. A survey made by Mittal et. al [72] categorizes energy saving approaches into four categories; 1) Dynamic Voltage and Frequency (DVFS) and power-aware scheduling techniques, 2) Using low power modes, called Power Mode Management (PMM), or dynamic power management, 3) Micro-architectural techniques, i.e., by leveraging the application properties or variation in workload to dynamically reconfigure a component of the system to save energy, and 4) Usage of accelerator cores, such as DSPs, GPUs and FPGAs.

In order to further improve these energy saving techniques, custom circuitry is often required, e.g., for platform re-configuration. Still, only a few of the available low energy general purpose embedded processors or microcontrollers are able to dynamically tune their power configurations at run-time while the processor core is active. For example, the SAM4L microcontroller board [8] and STM32F4 microcontrollers [103] support switching between two voltage states while the processor is active. At the same time, the dynamic behavior in advanced applications is increasing [65], and when used in mobile platforms, they enable more opportunities to implement energy saving approaches with finer granularity. Finer granularity leads to the need of more run-time re-configuration settings, and many computer architects are using computer architecture (CA) simulators, such as GEM5 [12], GEM5-GPU [84], and Multi2Sim [110], to model and analyze new processor designs and energy saving techniques [4]. A CA simulator incorporates detailed performance models and uses them to approximate the behavior of real hardware. They enable computer designers to experiment with a variety of different configurations early in the design stage of a system and investigate interesting tradeoffs before the prototype stage of the design process [99].

According to Spiliopoulos et. al [99], the use of DVFS and PMM are two of the most important techniques for managing the dynamic power consumption of a system. Therefore, CA simulators need configurable fre-

quency and voltage scaling functionality in order to further develop energy saving techniques. GEM5 [12] is a widely used CA simulator in academia, and researchers have proposed different solutions for DVFS management in full system mode [99], [46]. The McPAT energy modeling framework [63] is often used together with GEM5 output statistics to estimate the overall energy consumption. However, the GEM5 full system mode simulates a complete system in an operating system based simulation environment. This is not representable for embedded systems, which often run without an operating system (OS). The application itself then has to be in control of the DVFS management.

Furthermore, for many realistic contexts the simulation time of a full system is unacceptably long. GEM5 also provides a System Call emulation (SE) mode, which is an application-level simulation, where one only needs to specify the statically linked binary file to be simulated. FS mode on the other hand requires a full fledged OS simulation.

To our knowledge, no published DVFS implementations exist in GEM5 SE mode. A few implementations are available in full system mode, based on the Alpha processor [46], or implemented as a component responsible for setting clock frequencies according to OS policies [99].

In this paper, we propose 1) a non-intrusive application controlled DVFS management implementation in GEM5 SE mode, and 2) an architecture independent energy model based on classification of different measurable workload classes.

To our knowledge, there does not exist any published solutions for application controlled DVFS implementations in GEM5. Our DVFS mechanism is controlled by the application through custom pseudo instructions implemented in the GEM5 simulation kernel based on the ARM processor model. These custom pseudo instructions communicate with Python configuration scripts at the GEM5 user-level, where the DVFS controller is implemented.

Our architecture independent energy modeling framework uses known energy and power formulas available in literature [87], [72]. In our model, we separate different types of workloads into measurable classes, which are parametrized in our power formulas. These parameters are calibrated using measurements from real HW in order to estimate the power consumption of different workloads accurately, resulting in a fast and accurate energy model. These calibrated parameters are stored in a simple XML interface which is read by our energy model script. In this paper, we calibrate our parameters with power measurements from the SAM4L board [124]. Other architectures can be modeled similarly by exchanging our calibrated values

with new ones in our XML interface. The only information needed by the energy model script from the GEM5 simulator are the workload related statistics. Together this makes the setup fully reusable in a user friendly way for other in-order processor platforms.

Section E.2 presents an overview of CA simulators and existing DVFS implementations in GEM5. Our DVFS implementation in GEM5 is described in Section E.3, and our novel energy model is presented in Section E.4. In Section E.5 we present our experimental setup before introducing our results in Section E.6. We discuss the results in Section E.7. Finally, our conclusions are presented in Section E.8.

E.2 Related work

A number of CA simulators exist today in addition to the GEM5 simulator [12], such as SimpleScalar [9], Sniper [15], and Multi2Sim [110]. Multi2Sim [110] supports only out-of order execution and is mainly intended for CPU-GPU computing. SimpleScalar [9] and Sniper [15] are application-level simulators as opposed to GEM5, which is a full system simulator. The benefit of an application-level simulator is the ability to run only target applications instead of a full fledged target OS. The GEM5 simulator support application-level simulation, using the SE mode.

Li et al. [63] introduce an integrated power, area and timing modeling framework, called McPAT, that supports comprehensive design space exploration for multi-core and many-core processor configurations. This framework includes models for the fundamental components of different types of processor cores. With a flexible XML interface, McPAT can be interfaced with many performance simulators. McPAT focuses mainly on high performance processors with high complexity, and all parts of the system architecture must be modeled properly in order to extract the correct energy result from McPAT.

Similar to McPAT, a framework for analyzing and optimizing microprocessor power dissipation at the architectural level, called WATTCH, is presented by Brooks et al. [14]. It is a fast and high accuracy framework which is tailored to work together with SimpleScalar [9]. Similar to McPAT, WATTCH requires the internal components of the system architecture to be modeled properly in order to benefit from their energy model.

Spiliopoulos et al. [99] extends the GEM5 simulator to support full-system DVFS modeling. They extend the GEM5 clock and voltage domains,

and use them with a kernel-level DVFS controller containing configurable memory-mapped registers that interacts with the software. In addition to the DVFS extensions in GEM5, Spiliopoulos et. al [99] rely on McPAT [63] for the power models and they extend it with a set of coefficients that describes the GEM5 modeled system with DVFS.

Haririan et al. [46] presents non-intrusive full system DVFS emulation in GEM5 based on the DEC Alpha processor model. They model the performance aspects of DVFS in GEM5, where their configuration script at the user-level controls the DVFS based on custom instructions provided by the simulation kernel in full system mode. In other words, the configuration script directs the simulator to switch between two simulation sets at run-time, and scales the frequency at run-time based on the status of the application. Their solution does not allow the application to have direct DVFS control of the platform. The status of the main application is monitored by a concurrent utility application running periodically. This utility application communicates with the configuration scripts, and compares its performance counters with previously measured values. The comparison results are used to trigger a DVFS switch accordingly.

To our knowledge, DVFS controllers are only implemented in full system simulators, and without giving the application direct control over the DVFS switching. In this paper, we present our implementation of a DVFS controller in GEM5 SE mode, using a generic technique which can be applied to other simulators as well. Our application controlled DVFS implementation in GEM5 focuses on simulating low power embedded systems with less complex architecture than high performance processors. Our DVFS mechanism is application controlled through custom DVFS pseudo-instructions, and our energy model is calibrated with actual performance measurements from the SAM4L microcontroller board [8], containing one ARM Cortex M4 processor.

E.3 Contribution I: GEM5 DVFS in SE mode

In this section, we present our first contribution in this paper; SE mode DVFS implementation in GEM5. The GEM5 simulator is comprised of C++ processor models at the simulator kernel level which interacts with Python, Ruby, and Swig configuration scripts at the user level. Similar to Haririan et al. [46], we take advantage of GEM5's pseudo instruction functionality in the simulator kernel. However, instead of implementing

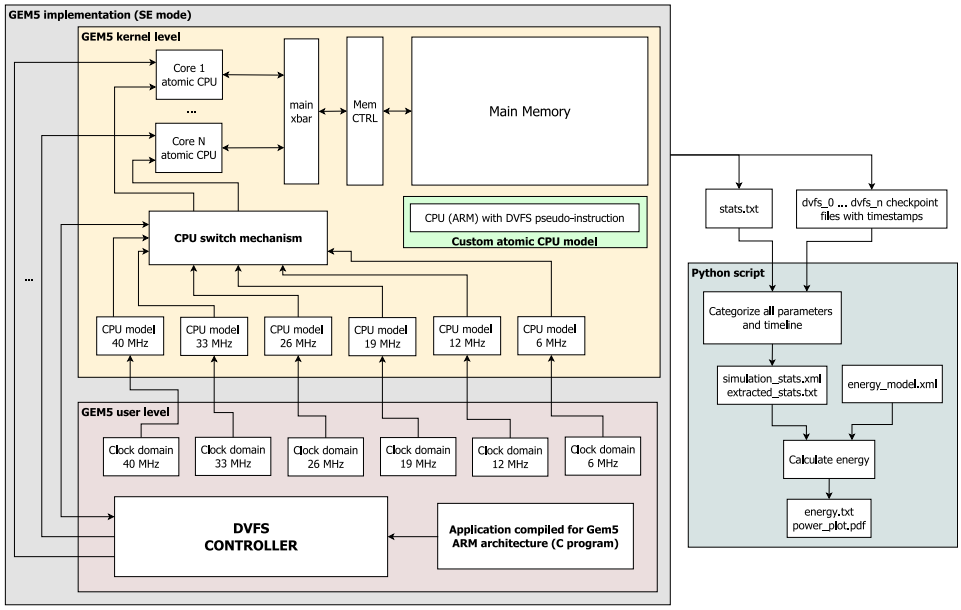


Figure E.1: GEM5 DVFS simulator flow

performance monitors controlling the DVFS configuration like Haririan et al. [46], we let the application control the DVFS switch through a custom pseudo instruction we implemented in the ARM processor model. This instruction interacts with the user level configuration scripts and switches the CPU model at run-time. Figure E.1 shows our GEM5 implementation, and different levels of our implementation are described in the following subsections. A summary of our main modifications are listed in Table E.1. Table E.2 compares our generalized modifications to state-of-the-art DVFS implementations in GEM5. In the following sub-sections we will describe our GEM5 DVFS simulator flow as depicted in Figure E.1. As shown in the figure, our DVFS implementation consist of user level and kernel level modifications to the GEM5 simulator, which outputs statistics and checkpoint files. These output files are then read by our energy model which is modelled using a Python script.

File path	Type of modification
Kernel level modifications	
In src/cpu/: scpu1800mv40mhz scpu1650mv33mhz scpu1500mv26mhz scpu1350mv19mhz scpu1200mv12mhz scpu1050mv06mhz	Copies of src/cpu/simple with modified variable and function names
In src/sim/: pseudo_inst.hh pseudo_inst.cc	Implementation of the DVFS instruction function
In src/arch/arm/isa/: operands.isa insts/m5ops.isa formats/m5ops.isa	Extension of the ARM instruction set with one DVFS instruction
In util/m5: m5op_arm.S m5op.h m5ops.h	Connecting the pseudo DVFS instruction to the configuration scripts
In src/python/m5/: simulate.py	DVFS checkpoint function used by the DVFS instruction
User level modifications	
In config/examples/: se.py	Declaration of each clock domain for DVFS
In config/common/: Simulation.py Options.py CpuConfig.py	Connecting CPU models to the system, adding the DVFS options flag and controller functionality

Table E.1: GEM5 modifications

State-of-the-art	Our implementation
Works only in full system mode	Implemented for SE mode
Custom versions of simple, detailed and timing CPU models used with DVFS	Copies of the simple atomic CPU model with modified variable and function names
Implementation of the DVFS pseudo-instructions used with performance monitors	Implementation of application controlled DVFS pseudo-instruction used together with GEM5's checkpoint mechanism
OS controlled or system monitor based DVFS policies through either kernel level mechanisms or by GEM5's *switchCpus* mechanism	DVFS policy triggered by a DVFS instruction from the application together with GEM5's *switchCpus* mechanism

Table E.2: Comparison to state-of-the-art DVFS implementations in GEM5

E.3.1 GEM5 kernel level modifications

In this paper we have implemented our DVFS mechanisms in a generic way so that it can be tailored to work with any in-order CPU model. Only CPU specific mechanisms, such as our custom DVFS instruction, must be ported to the target instruction set architecture, provided there exist available space for additional instructions. In this paper, we apply our changes to the ARM processor model.

Compilation of the GEM5 kernel is only required if new frequencies or CPU's are added or modified. Our GEM5 version does not have native support for multiple clock domains per CPU model in SE mode. Therefore, we have implemented one custom CPU model for each performance level. These custom CPU models are copies of GEM5's simple atomic in-order CPU model with modified file and variable names as shown in Table E.1. Our CPU models are controlled by the GEM5 *switchCpus* function used in the Python configuration scripts. The *switchCpus* function simply switches the currently active CPU model to another available model at run-time.

Our DVFS instruction, *gem5Dvfs*, is implemented in an unused location within the GEM5 model of the ARM instruction set. This instruction combines available registers for storage of the DVFS state, delay and period

variables of our DVFS function. The functionality of the DVFS instruction is modeled in the pseudo_inst files shown in Table E.1. The *gem5Dvfs* function is shown in Listing E.1.

At the end of *gem5Dvfs* in Listing E.1 the simulation exits its loop with "gem5_DVFS" as the cause of exit. This message is then handled by the user level configuration scripts to switch the current CPU model with another model based on the value of the *dvfs_state* variable from the application, before the simulation continues. The *when* and *repeat* variables are optional configurations that can delay a DVFS switch with a number of ns, or trigger a periodic DVFS switch depending on what is required by the application. In this paper, we set these parameters to zero because the energy delay is handled by our energy model described in Section E.4. This instruction is connected to the *gem5_dvfs_inst* function in the GEM5 utilities, which are included when compiling the application for the target platform in GEM5. The utility function makes the DVFS instruction available to the application.

```
1 void gem5Dvfs(ThreadContext *tc, uint64_t dvfs_state, Tick delay,
      Tick period){
2  if (!tc->getCpuPtr()->params()->do_checkpoint_insts)
3   return;
4
5  // 1 Tick = 1 ps. Delay is in ns
6  Tick when = curTick() + delay * SimClock::Int::ns;
7  Tick repeat = period * SimClock::Int::ns;
8  exitSimLoop("gem5_DVFS", dvfs_state, when, repeat);
9 }
```

Listing E.1: DVFS pseudo instruction

In order to control the DVFS switches occurring while the application is running we implemented a copy of the GEM5 checkpoint function to create DVFS checkpoints with time stamps each time the DVFS instruction is used. In addition we force the simulator to start a new collection of performance statistics after each DVFS instruction.

These statistics are stored in the stats.txt output from GEM5 where each DVFS performance level is collected in separate simulation rounds. Each simulation round in the stats.txt file in GEM5 lists the statistics for all CPU models instantiated in the configuration script. In order to know which CPU was active in each simulation round, we also introduced a new statistics variable, *dvfsCPUActive*, in each of our custom CPU models. This variable is used in a Python script in a post-processing step to extract the performance history from the correct CPU.

E.3.2 GEM5 user level modifications

Our DVFS controller is implemented in the GEM5 Python configuration scripts at the user level. The advantage of having the DVFS controller in the configuration script is that the designer can change and experiment with different DVFS switching policies without recompiling the simulation kernel.

In GEM5's SE mode user level script, se.py, we instantiated one CPU clock domain for each of our custom CPU models as shown in Figure E.1, and assigned them their corresponding frequency. These domains are then connected with their corresponding CPU models in the *switchCpus* mechanism in the common user level Simulation.py script. Our working version of the GEM5 simulator did not permit a frequency change after the processor was initialized. However, it allowed changing the processor model through the GEM5 *switchCpus* function, which worked when we assigned separate clock domains for each CPU model. The DVFS functionality is only activated when our "--gem5-dvfs" option is enabled at the start of the simulation. We modified the GEM5 run function used by se.py in Simulation.py to include our DVFS controller as shown in Listing E.2. Our controller is implemented in the *gem5_dvfs_switch* function as shown in Listing E.3.

```
1 if options.repeat_switch and maxtick > options.repeat_switch:
2  exit_event = repeatSwitch(testsys, repeat_switch_cpu_list,
       maxtick, options.repeat_switch)
3 else:
4  if options.gem5_dvfs:
5   exit_event = gem5_dvfs_switch(options, maxtick, cptdir, testsys)
6  else:
7   exit_event = benchCheckpoints(options, maxtick, cptdir)
```

Listing E.2: Modification in the run function

Our current DVFS controller handles homogeneous multi-core switching, i.e., all active and parallel processor cores of the same kind change their models simultaneously when the application executes the DVFS instruction. As shown in Listing E.4 and E.6, our *dfs* function generates a CPU switch list. This switch list is used by the GEM5 *switchCpus* function to switch the active CPU with another model. An extension of our *dfs* function to support different models to be simultaneously active is considered future work, and requires heterogeneous multi-core support in GEM5.

The functionality of our DVFS controller is implemented in our Platform Adaption Manager (PAM) function called *PAM_select*. This function selects which CPU model to choose from based on the application request given

by the *dvfs_state* variable, and triggers a frequency change from our *dfs* function. The *dfs* and *PAM_select* functions are shown in Listing E.4 and E.5.

```
def gem5_dvfs_switch(options, maxtick, dvfsdir, testsys):
 switch_cpu_list = []
 np = options.num_cpus

 exit_event = m5.simulate(maxtick - m5.curTick())
 exit_cause = exit_event.getCause()

 num_checkpoints = 1
 max_checkpoints = options.max_checkpoints

 # Initial event needs to be handled
 # before the while loop
 if "gem5_DVFS" in exit_cause:
  exit_cause_base = exit_cause
  dvfs_state = exit_event.getCode()

  switch_cpu_list = PAM_select(dvfs_state, testsys,\
                               np)
  if len(switch_cpu_list) > 0:
   m5.switchCpus(testsys, switch_cpu_list)
 else:
  exit_cause_base = exit_cause

 while exit_cause_base == "gem5_DVFS":
  m5.dvfs_checkpoint(joinpath(dvfsdir, "dvfs_"+\
                     str(num_checkpoints)+".%d"))
  num_checkpoints += 1

  exit_event = m5.simulate(maxtick - m5.curTick())
  exit_cause = exit_event.getCause()
  if "gem5_DVFS" in exit_cause:
   exit_cause_base = exit_cause
   dvfs_state = exit_event.getCode()

   switch_cpu_list = PAM_select(dvfs_state, testsys,\
                                np)
   if len(switch_cpu_list) > 0:
    m5.switchCpus(testsys, switch_cpu_list)
  else:
    exit_cause_base = exit_cause

 return exit_event
```

Listing E.3: DVFS Controller

```
def dfs(testsys, np, new_frequency):
 switch_cpu_list = []

 for j in xrange(np):
  # Select the first (and only) active CPU type
  if not testsys.cpu[j].switchedOut():
   old_cpu = "cpu"
   switch_cpu_list = generate_switch_list(testsys, old_cpu,
       new_frequency, j, np)
  elif not testsys.switch_cpus_40mhz[0].switchedOut():
   if new_frequency is "40":
    switch_cpu_list = []
   else:
    old_cpu = "switch_cpus_40mhz"
    switch_cpu_list = generate_switch_list(testsys, old_cpu,
        new_frequency, j, np)
  # ...
  elif not testsys.switch_cpus_06mhz[j].switchedOut():
   if new_frequency is "06":
    switch_cpu_list = []
   else:
    old_cpu = "switch_cpus_06mhz"
    switch_cpu_list = generate_switch_list(testsys, old_cpu,
        new_frequency, j, np)
  else:
   fatal('All CPUs are switched out!')
 return switch_cpu_list
```

Listing E.4: dfs function

```
def PAM_select(state, testsys, np):
 if state == DVFS_STATES.MHz_40_V_1_80:
  return dfs(testsys,np,"40")
 # ...
 elif state == DVFS_STATES.MHz_06_V_1_05:
  return dfs(testsys,np,"06")
 else:
  return dfs(testsys,np,"40")
```

Listing E.5: PAM_select function

```
def generate_switch_list(testsys, old_cpu, new_frequency, j, np):
 cmd = "testsys.switch_cpus_" + new_frequency + "mhz[j].workload"
 exec("%s = %s" % (cmd,"testsys." + old_cpu + "[j].workload"))
 cmd = "[(testsys." + old_cpu + "[i], testsys.switch_cpus_" +
     new_frequency + "mhz[i]) for i in xrange(np)]"
 exec("%s = %s" % ("switch_cpu_list",cmd))
 return switch_cpu_list
```

Listing E.6: generate_switch_list function

E.3.3 Usage of DVFS in the application

Listing E.7 shows the DVFS function, *dvfs_scenario*, which must be included in the application for DVFS control. In our tests we also trigger a DVFS switch at the beginning and at the end of the application in order to isolate the power estimation of the application. *dvfs_scenario* initially dumps the performance statistics of the current performance level, before switching the CPU model through our DVFS function. Immediately after the switch all performance statistics are reset, and the performance statistics for the new CPU model are then collected in a separate simulation round in stats.txt.

Compared to state-of-the-art solutions, we introduce direct application control of the voltage and frequency mechanisms in the simulated platform. Our implemented mechanisms are generic because we mainly add extensions to architecture independent parts of the simulator. Only the implementation of the *gem5Dvfs* instruction needs to be ported to an available location in the instruction set of other architectures of interest.

```
1 inline void* dvfs_scenario( int state ){
2   m5_dump_stats(0,0); // Split the statistics
3   gem5_dvfs_inst(state,0,0); // DVFS checkpoint
4   m5_reset_stats(0,0); // Start new statistics
5 }
```

Listing E.7: Inline DVFS instruction implementation

E.3.4 Limitations of our DVFS mechanisms

The current implementation of our DVFS mechanisms is limited to the simple atomic in-order CPU model in GEM5. Our mechanisms focus on CPU voltage and frequency scaling, and do not support memory hierarchy voltage and frequency scaling. For our focus, this is not a limitation. However, doing the following minor modifications will enable using our technique for other domains as well. To support DVFS in the memory hierarchy, mechanisms similar to the *switchCpus* mechanism is required for the memory hierarchy in GEM5 to enable memory level DVFS. Similarly, a heterogeneous multi-core support in GEM5 is required before our DVFS mechanisms can be extended to support heterogeneous multi-core DVFS. These extensions are considered out of the scope of this paper.

E.4 Contribution II: Energy model

The second contribution in this paper is our GEM5 energy model, based on power measurements from the SAM4L board with different workloads. The theory and assumptions behind our energy model and power consumption formulas are presented in Section E.4.1. How we combine the GEM5 statistics with SAM4L power measurements is presented in Section E.4.2, before we present our energy modeling script in Section E.4.3. Section E.4.4 describes how we applied offset error corrections in GEM5.

E.4.1 Estimation of the power consumption

Our model for power consumption is based on Equation E.1 and E.2, [87].

$$P_{\text{Total}} = P_{\text{Static}} + P_{\text{Dynamic}} \tag{E.1}$$

$$P_{\text{Dynamic}} = C_E f V_{\text{dd}}^2, \text{ where } C_E = \alpha C \tag{E.2}$$

In Equation E.2, C_E is the effective load capacitance, V_{dd} is the supply voltage, f is the clock frequency, α is the activity factor (between 0 and 1), and C is the load capacitance. According to [19], the static power can be modeled as shown in Equation E.3, where $P_{\text{Short-circuit}}$ is the short-circuit (direct path) power and I_{Leak} is the sub-threshold current (leakage current). These are given by Equation E.4 [113] and E.5 [7]. In Equation E.4, β is the gain factor of a MOS transistor ($\mu A/V^2$), τ is the rise or fall time of a signal, f is the device frequency, V_{Th} is the threshold voltage, and V_{dd} is the supply voltage. According to Alioto [5], the overall energy per clock cycle of a VLSI system consists mainly of the dynamic and the leakage energy. Alioto [5] argues that the short-circuit current energy contribution is usually negligible, due to the exponential MOS I-V characteristics. Rabaey et al. [87] also explains that the short-circuit power in Equation E.4 can be neglected for well designed circuits, because it is normally less than 10% of the dynamic power dissipation, except for slow input signals (large τ). We therefore assume that $P_{\text{Leak}} >> P_{\text{Short-circuit}}$, and hence neglect the short circuit power consumption.

$$P_{\text{Static}} = P_{\text{Short-circuit}} + I_{\text{Leak}} \cdot V_{\text{dd}} \tag{E.3}$$

$$P_{\text{Short-circuit}} = \frac{\beta}{12}\tau f(V_{\text{dd}} - 2V_{\text{Th}})^3 \tag{E.4}$$

$$I_{\text{Leak}} = I_{\text{ds}} = I_0 e^{(1-k)\frac{V_{\text{bs}}}{V_T}} e^{k\frac{V_{\text{gs}}}{V_T}} (1 - e^{-\frac{V_{\text{ds}}}{V_T}} + \frac{V_{\text{ds}}}{V_0}) \tag{E.5}$$

In Equation E.5, V_{gs} is the gate-to-source potential, V_{ds} is the drain-to-source potential, V_{bs} is the substrate-to-source potential, $V_T = K \cdot T/q$ is the thermal voltage (26 mV at room temperature [7]), I_0 is the zero-bias current, V_0 is the early voltage (proportional to channel length), and k is the effectiveness of the gate potential in controlling the channel current. Assuming that our target device operates in saturation, which is the case for $V_{\text{ds}} > 3 \cdot V_T$ ([7]), we can ignore the body effect. Equation E.6 assumes $V_{gs} = 0$, and results in a simplified formula for the leak current which can be used when $V_{\text{ds}} >> V_{\text{Th}}$, which is the case for our target platform architecture.

$$I_{\text{Leak}} = I_0 + \frac{I_0}{V_0} V_{\text{ds}} \tag{E.6}$$

In our model we assume that the drain-to-source potential (V_{ds}) is equivalent to the supply voltage (V_{dd}). Hence, our model for total power consumption valid within our target domain, is as shown in Equation E.7.

$$P_{\text{Total}} = I_0 V_{\text{dd}} + \frac{I_0}{V_0} V_{\text{dd}}^2 + C_E f V_{\text{dd}}^2 \tag{E.7}$$

In order to estimate the values of C_E, I_0 and V_0, we measure the total power consumption on our SAM4L microcontroller at 40 MHz and 20 MHz with $V_{\text{dd}} = 1.8$ V, and similarly at 12 MHz with $V_{\text{dd}} = 1.2$ V. With these three measurements we use Equation E.7 to find the values of C_E, I_0 and V_0. We also observe that these values differ when the SAM4L board is executing different types of workloads. I.e., when the application is reading and writing to memory, the measured C_E value is, as expected, higher than when the SAM4L board is only executing arithmetic instructions (a memory access has higher load capacitance than the ALU). We made two separate measurements on our SAM4L board; one measurement with a computational workload, and one measurement with a memory access workload. From these measurements, we calculate the C_E, I_0 and V_0 values shown in Table E.3.

Table E.3: Variables related to the total power based on measurements

Type of workload	C_E (nF)	V_0 (V)	I_0 (mA)
Computation	0.18673	-0.55610	-0.84444
Memory access	0.20833	-0.49091	-0.75000

E.4.2 Combining GEM5 statistics with SAM4L power measurements

The GEM5 simulator outputs a stat.txt file containing statistics such as the number of instructions executed, the type of instructions executed, the number of memory accesses (reads and writes), and the number of simulated cycles at a given frequency. In our implementation of the DVFS mechanism, the simulator also outputs checkpoint files for each executed DVFS instruction. These checkpoint files contain an ID number and a time stamp, which is used together with the stats.txt file to map a time line of simulation events when using DVFS. Our DVFS mechanism splits the statistics in the stat.txt files into different chunks in order to separate the activity occurring at different frequencies into simulation sets. Together with our power model and parameters from Table E.3, we calculate the total energy consumption for the GEM5 simulation as shown in Equation E.8.

$$E_{\text{Total}} = E_{S_U} \cdot S_U + E_{S_D} \cdot S_D + \sum_{i=E0}^{En} E_{\text{Sim}_i} \tag{E.8}$$

In Equation E.8, $E0$, $E1$, ..., En are the energy consumptions of all the simulation sets derived from our model of the power consumption, Equation E.7, and the output stat.txt file from GEM5. E_{S_D} and E_{S_U} are the energy scale down and up overhead for DVFS, respectively. S_U and S_D are the number of scale up and scale down events (i.e., DVFS switches) occurring throughout the simulation. Our power model assumes that the C_E, V_0 and I_0 parameters are known for different workload classes, i.e., the groups of instructions with similar activity factors (α). As described in Section E.4.1 we define two workload classes in this paper; one for memory access instructions and one for the computational (arithmetic) instructions. Estimated values for C_E, V_0 and I_0 for each workload class based on measurements from our SAM4L board are found in Table E.3. Deep-sleep power consumption and the energy consumption of the DVFS switching overhead (E_{S_D} and E_{S_U}) are measured directly on the SAM4L board (containing an ARM Cortex

M4 processor) with an oscilloscope and an ampere meter. The oscilloscope was used to find the time between two pin pull-up signals on the SAM4L board; one before and one after a DVFS switch. The ampere meter was used to measure the current consumption of the chip during this time, and to measure the deep-sleep power consumption.

E.4.3 GEM5 energy modeling script

We calculate the application's total energy consumption with a Python script. Our script can calculate the energy consumption in two ways, defined by a NO_INTERVAL input parameter. If NO_INTERVAL is zero the energy model assumes our application processes a set of workloads arriving in uniform time intervals (e.g., every seconds). The application is in this case assumed to complete its workload, go to sleep and wake up just before the next workload arrives. Otherwise, when NO_INTERVAL is one, our model assumes that the application executes one single non-periodic workload and goes to sleep immediately when it is finished.

For CPU bound workloads, active execution at high frequency alternating with deep sleep typically leads to a larger overall energy reduction than operating at lower frequency and V_{dd} without deep sleep. This technique is known as Race-To-Halt (RTH) [10]. As explained in [10], DVFS techniques are on the other hand more suitable for memory bound workloads, and for real time systems with deadlines and long idle times. In deep sub-micron technology nodes the RTH technique is useful because the leakage and short-circuit current are increasing in magnitude, and in the end dominate. For scaled, bulk CMOS, this could typically be the case. For simplicity, only the active execution is modeled in our GEM5 simulator, and the sleep energy consumption is added for each simulation set. Equation E.9 shows how we calculate the total energy per simulated frequency.

$$E_{\text{Sim i}} = P_{\text{active}_i} \cdot t_{\text{active}_i} + P_{\text{sleep}_i} \cdot t_{\text{sleep}_i} \tag{E.9}$$

P_{active} in Equation E.9 is calculated using the total power model of Equation E.7. One of the benefits of the GEM5 simulator output statistics is the modeling of the number and types of instructions executed, including memory accesses. The separation between different workload classes allows us to model the effective capacitance C_E more accurately since the activity factor α in Equation E.2 varies with different types of workload classes.

The total effective capacitance $C_E = \alpha_{\text{Total}} C$ is split up as shown in Equation E.10. Our measured values for C_E of each workload class are combined in Equation E.10 to estimate the effective activity factor α_{Total} for our specific experiment. The overall methodology presented in this paper can this way handle any number of workload classes.

$$\alpha_{\text{Total}} C = K_{\text{C}} \alpha_{\text{C}} C + K_{\text{M}} \alpha_{\text{M}} C \tag{E.10}$$

$$K_x = \frac{\#\text{Instructions of type x}}{\#\text{Sum of all executed instructions}} \tag{E.11}$$

In Equation E.10, K_{C} and K_{M} are the fraction of the execution that are computational and memory access instructions respectively. They are calculated using Equation E.11, where x is the type of instruction.

E.4.4 Offset error corrections

Additional circuitry in the SAM4L board are hard to incorporate into our model because it requires vendor specific information which is not publicly available, e.g., the clock circuitry. We assume that the power consumption of the additional circuitry can be compensated with systematic offset corrections based on separate measurements with different types of workloads. In order to fine-tune the energy results of our GEM5 simulator, we added offset correction to the measured parameters in Equation E.7, resulting in Equation E.12. The measured parameters are based on our calibration applications with two different workload classes defined earlier (computational and memory access). The offset correction parameters are obtained by running the calibration application in the SAM4L board and on GEM5, and comparing the power consumption results. The offset corrections are then approximated experimentally until the deviation in the simulated GEM5 results is minimized.

$$P_{\text{Total}} = I_0 X_{I_0} V_{\text{dd}} + \frac{I_0 X_{I_0}}{V_0 X_{V_0}} V_{\text{dd}}^2 + C_E X_{C_E} f V_{\text{dd}}^2 \tag{E.12}$$

E.5 Experimental setup

Our DVFS extension is implemented on the stable_2014_12_14 release[1] of GEM5. Our modular approach makes it easy to port to newer GEM5 releases through adaption of new or modified system components.

The DVFS GEM5 simulator is tested using two applications obtained from the Mediabench II website [60]; H264 and JPEG. From H264, we extracted the encoder control structure from the Mediabench II source code, and modeled it under the assumption that all memory accesses take one clock cycle. We implemented the extracted control structure using our framework for system scenarios (FSS) [126]. Test data for three video streams simulated using different wireless networks (WLAN, LTE and WCDMA) was ported to GEM5 from our previous work [124]. The overhead of the scenario mechanisms in GEM5 were estimated using a separate program where all other code is removed. The energy consumption of the scenario mechanisms were averaged over 1 million frames in GEM5, resulting in an overhead of 1.54 μJ per frame simulated at 1.8 V and 40 MHz. E.g., in a stream of 500 frames the energy overhead of our scenario mechanisms (1.54 μJ per frame) equals less than 0.01% of the total energy consumed while processing the stream, which we consider to be negligible.

Like our H264 encoder control structure application, we extracted the JPEG compression control structure from the Mediabench II JPEG application, and the energy consumption of the JPEG compression application is measured on the SAM4L board with three different configurations. In the first configuration, we compress 500 consecutive 600x400 frames at 1.8 V and 40 MHz (JPEG FAST). In the second configuration we compress the same frames at 1.2 V and 12 MHz (JPEG SLOW). In the final configuration, we compress the frames using DVFS (JPEG DVFS), where we switch between our two voltage and frequency settings after every 10th compressed frame (i.e., 50 DVFS switches for 500 frames) in order to simulate a sequence with multiple DVFS switches.

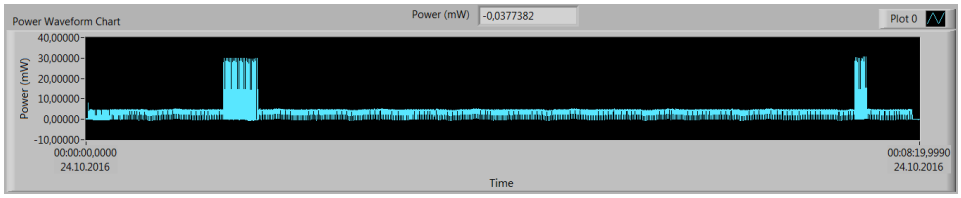


Figure E.2: SAM4L simulation

E.6 Results

E.6.1 Correction of systematic offset errors

The energy consumption of our applications are measured on the SAM4L microcontroller and calculated using our modified GEM5 simulator. The deviation of the GEM5 simulator relative to the SAM4L measurements before and after correction of systematic offset errors are shown in Table E.4.

Table E.4: GEM5 energy measurement deviation from SAM4L measurements

Application	Diff in value (%) without correction	Diff in value (%) with correction
H264 WLAN Scenario	14.5	1.2
H264 LTE Scenario	11.3	-0.6
H264 WCDMA Scenario	7.9	-4.0
H264 WLAN NO Scenario	13.5	-0.01
H264 LTE NO Scenario	13.0	-0.6
H264 WCDMA NO Scenario	11.8	-1.9
JPEG FAST	13.5	0.06
JPEG SLOW	10.7	-0.9
JPEG DVFS	12.7	-0.1

We used different error correction factors for different workloads as shown in Table E.5. These error correction factors are obtained by comparing the results of our calibration applications on the SAM4L board and

[1] https://github.com/gem5/gem5/releases

GEM5. These factors correct offsets in the result caused by systematic errors due to additional processor and clock circuitry in the SAM4L board which are hard to incorporate into our model. From Table E.5 we can observe that no error corrections were needed for the static power, which implies that our assumptions in the static power model have good enough accuracy to directly model the SAM4L board.

Table E.5: GEM5 offset correction parameters

Workload	X_{C_E}	X_{I_0}	X_{V_0}
Arithmetic	1.18	1.0	1.0
Memory access	1.17	1.0	1.0

E.6.2 Comparing energy reduction techniques in GEM5

Our GEM5 DVFS simulator can be used by designers to efficiently evaluate alternative strategies for energy efficient hardware and software implementations. As an example, Table E.6 compares the relative energy savings of using the system scenario based design methodology over a brute-force RTH technique. We presented these experiments in our previous work [124] based on time consuming measurements on the SAM4L board. The same experiments have now been performed using the GEM5 DVFS simulator. The relative energy consumption based on the SAM4L measurements are listed in Table E.6 together with the energy consumption obtained from the GEM5 simulator (with and without systematic error corrections). In Table E.6, negative values means reduction in the energy consumption with the system scenario technique compared to the RTH technique. From Table E.6 we can observe that our improvements are not affected much by the systematic offset errors.

In our previous work [124], we motivated that more available DVFS modes could result in further energy improvements. Figure E.3 shows the relative energy improvements with 6 compared to only 2 DVFS modes, as found using our GEM5 simulator. Note that a stricter bandwidth sensitivity requirement of the LTE network was used for the 6 DVFS mode experiment, as discussed in [124]. The energy improvements are also compared to a RTH solution with one constant voltage and frequency.

Figure E.2 and Figure E.4 show the GEM5 simulation plot and the simulation view of our locally developed Labview program, showing the

Table E.6: Energy consumption of our H264 application

Application	Compared to no scenario in SAM4L	Compared to no scenario in GEM5	Compared to no scenario in GEM5 with correction
H264 WLAN	0.7 %	-0.5 %	-0.5 %
H264 LTE	-44.2 %	-43.1 %	-44.2 %
H264 WCDMA	-49 %	-46.7 %	-47.9 %

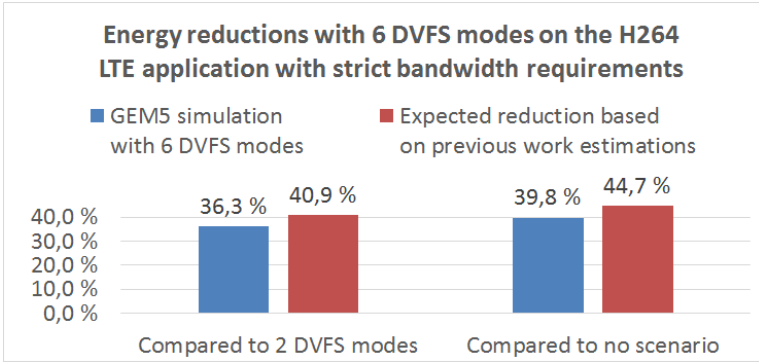


Figure E.3: Energy reduction with 6 DVFS modes

measured power consumption over time.

E.7 Discussion of results

Compared to measurements done with the same applications on the SAM4L board the results of our H264 and JPEG applications simulated in GEM5 show at most 4% deviation in values after offset error corrections. Deviations occur for a number of reasons, and our measured deviation varied with the supply voltage. The main cause for these deviations are systematic offset errors, originating from additional processor and clock circuitry in the SAM4L board which are hard to incorporate into our model. Our offset correction parameters in Table E.5 shows at most 18% offset error in the C_E estimation. This offset scales well with different V_{dd} values compared to our SAM4L measurements. The I_0 and V_0 variables are not offset corrected, and implies that our assumptions related to the static power model have

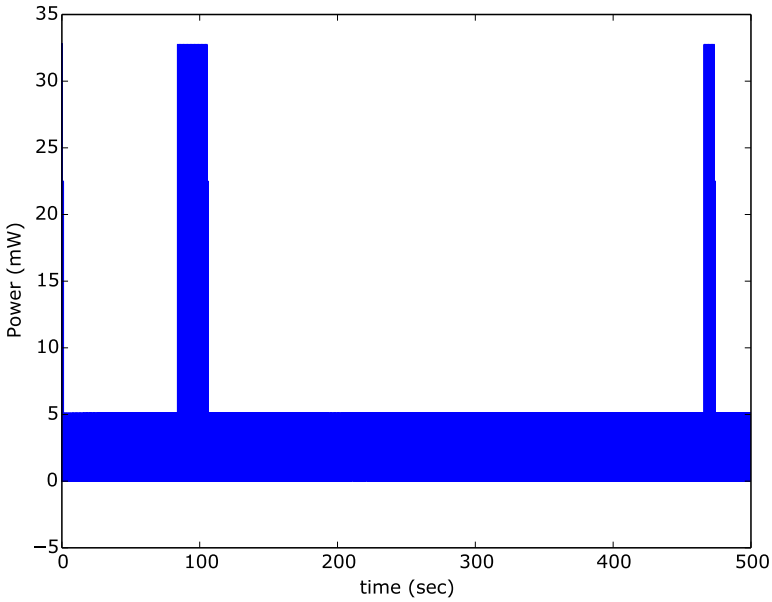


Figure E.4: GEM5 simulation

good enough accuracy to match the SAM4L board.

Other reasons for the deviations are the platform and model differences between the SAM4L microcontroller and the ARM model in GEM5. The SAM4L board contains an in-order ARM Cortex M4 processor, while the ARM processor model in GEM5 is based on a simplified ARM Cortex A processor. Even if we use the simple atomic in-order CPU model in GEM5, these architectural differences will result in a small difference in the number of executed instructions. These architectural differences results in different settings being used in the ARM compiler on the SAM4L board compared to the ARM compiler used with GEM5 (such as platform specific optimization flags). Slight measurement errors and noise from the wires and probes on the SAM4L board also affect the measured power consumption, which would deviate slightly from expected values. After our offset corrections, the deviation is reduced significantly, to at most 4%, which we find acceptable. In general, our offset corrected energy reduction results from the GEM5 simulator coincides well with the measured improvements from the SAM4L board for the H264 application.

In our previous work [124], we expected a significant improvement when using more than 2 DVFS modes based on intra- and extrapolations of our DVFS modes on the SAM4L board. There, we calculated the expected dy-

namic power consumption when 6 DVFS modes are available. With stricter bandwidth requirements and $C_E = 0.165nF$, the system scenario technique (without using RTH) was roughly estimated to improve the energy consumption by 59% compared to not using scenarios [124]. In this paper we use the same parameters, and extend this estimation with the combination of system scenario and RTH techniques, which results in reduced energy estimates for both our solutions. We predicted 40.9% energy reduction when combining RTH with the system scenario technique with 6 DVFS modes compared to only having 2 DVFS modes available. The GEM5 simulation results in 36.3% improvement, which is slightly less than our roughly estimated value of 40.9%. Our GEM5 energy model takes into account the static and sleep energy consumption, which is not taken into account by our estimates. The finer granularity of our power model, with different workload classes (and C_E values) also contributes to the difference from the estimated values.

Our DVFS implementation and energy modeling approach differs from previous techniques presented in Section E.2, e.g., through introduction of workload classes and through calibration using measurements from real hardware. The energy model can easily be calibrated to other types of architectures, by taking new measurements of the power consumption for different workload classes, and measuring the DVFS switching energy. These measurements can then be used to re-calculate the C_E, I_0 and V_0 parameters for the new target architecture. Our changes to the GEM5 simulator kernel and configuration scripts are for the most part architecture independent and hence fully reusable, except from the DVFS instruction. GEM5 supports different CPU models, which can be used with our GEM5 extension by applying the modifications mentioned in Section E.3. This enables the system designer to experiment with new techniques on non-ARM architectures, and add more DVFS modes or different switching policies.

E.8 Conclusion

We present a novel non-intrusive application controlled DVFS management in GEM5 SE mode, and a novel Python script-based energy model used together with GEM5. The application triggers DVFS through one custom pseudo-instruction, implemented in GEM5's ARM instruction-set model. This instruction takes advantage of GEM5's utility functions, making it easily portable to other processor architectures. The rest of our DVFS mech-

anisms are architecture independent. Our novel energy model accurately parametrizes different workload classes, which are calibrated with power measurements from real HW. The results from our modified GEM5 simulator are validated with two different, representative media applications. The energy results from our GEM5 simulation are compared to and calibrated with simulations from a SAM4L microcontroller from Atmel, resulting in less than 4% deviation after systematic offset error correction. Our changes to the GEM5 simulator kernel and configuration scripts are for the most part architecture independent and hence fully reusable, except from the DVFS instruction. We believe our modifications to GEM5 mentioned in this paper can be applied to other in-order CPU models supported by GEM5.

APPENDIX

F

ALGORITHM/ARCHITECTURE CO-OPTIMISATION TECHNIQUE FOR AUTOMATIC DATA REDUCTION OF WIRELESS READ-OUT IN HIGH-DENSITY ELECTRODE ARRAYS

Yahya H. Yassin, Francky Catthoor, Fabian Kloosterman, João Couto, Jyh-Jang Sun, Per Gunnar Kjeldsberg and Nick Van Helleputte
ACM Transactions on Embedded Computing Systems (TECS)
2018
In press.

Abstract

High-density electrode arrays used to read out neural activity will soon surpass the limits of the amount of data that can be transferred within reasonable energy budgets. This is true for wired brain implants when the required bandwidth becomes very high, and even more so for untethered brain implants that require wireless transmission of data. We propose an energy efficient spike data extraction solution for high-density electrode arrays, capable of reducing the data to be transferred by over 85%. We combine temporal and spatial spike data analysis with low implementation complexity, where amplitude thresholds are used to detect spikes and the spatial location of the electrodes is used to extract potentially useful sub-threshold data on neighboring electrodes. We tested our method against a state-of-the-art spike detection algorithm, with prohibitively high implementation complexity, and found that the majority of spikes are extracted reliably. We obtain further improved quality results when ignoring very small spikes below 30% of the voltage thresholds, resulting in 91% accuracy. Our approach uses digital logic and is therefore scalable with increasing number of electrodes.

F.1 Introduction

The behavior of neural cells and networks are studied through extracellular monitoring of the action potentials in animals' brains in vivo. The activity is normally recorded using very small electrodes on implantable neural probes. The digitized signals are then transmitted to a remote receiver where detailed analysis is performed using advanced signal processing algorithms.

The electrodes detect extracellular voltage deflections that are a mixture of action potentials (spike band: 300 Hz - 6 kHz) emitted by nearby neurons and slower fluctuations that reflect population activity (local field potential band: 0.1 Hz - 300 Hz). In this paper we focus on a data reduction method applied to the spike band, based on the detection of action-potentials (also known as "spike detection") [86].

At least 20 kHz sampling rate is required to capture the spike waveform necessary for spike detection and further analysis. Figure F.1 illustrates a typical probe and its electrode arrangement. The figure also shows the bandpass filtered signal (0.250 - 5.938 kHz) from each electrode at 25 kHz sampling rate, which are analyzed together in order to extract relevant spike data for post-processing. Due to the high electrode density, a spike detected in one electrode is also detected in neighboring electrodes. One of the main challenges of using such electrodes is to capture temporal and spatial spike data in noisy environments. Temporal spike data is captured by inspecting a single channel over time, while spatial spike data is captured by inspecting multiple channels with different spatial placement on the probe.

The number of electrode channels will soon reach the limits of how much data we can transfer within reasonable energy budgets, even while transmitting the data through electrical wiring [88]. The reason is the fast growing number of concurrent sensor nodes (electrodes) on a neural probe. The state-of-the-art neural probe produced with silicon technology contains 1356 electrodes [88]. Raducanu et al. [88] show that the transmission circuitry contributes significantly to the power consumption. Other research projects aim towards thousands of electrodes on a single neural probe [100], [20]. This will exacerbate the transmission problem even more.

Researchers also aim at reducing the number of wires connected to the probe, or eliminate them all together, in order to have more freedom of movement while recording. A wireless transmitter embedded on a battery powered probe will limit the amount of data that can be transmitted in "raw form" even further compared to wired probes. Such wireless probes introduce a long term energy constraint, i.e., the total available energy

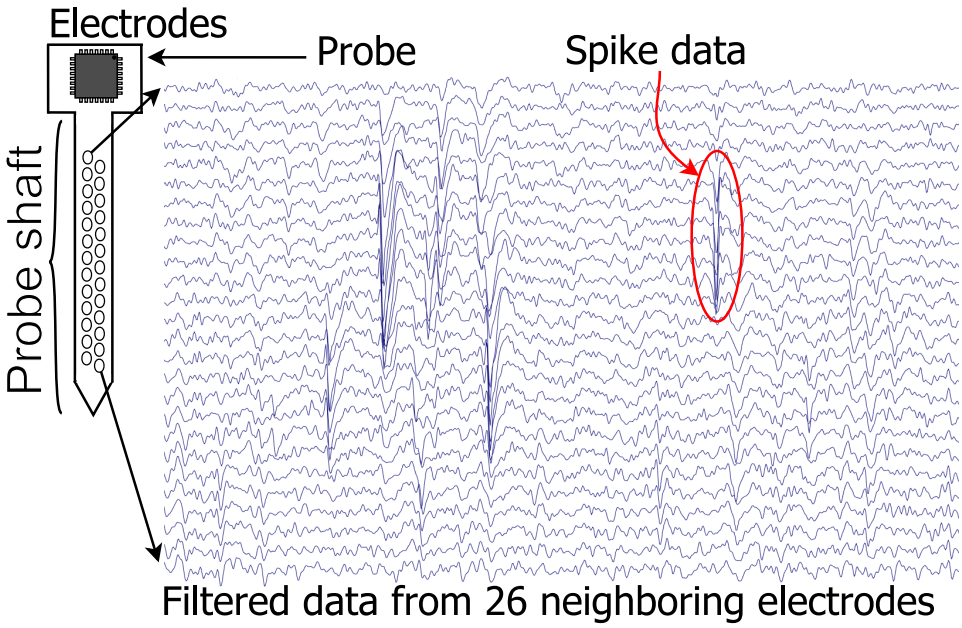


Figure F.1: The filtered recorded signal of a probe with multiple electrodes, clearly showing the spikes and spatial signal dependency.

based on the battery capacity. Even if the electrodes are implemented with optimal energy efficient circuitry, it is still a challenge to meet the energy constraints related to wired and wireless data transmission for future high-density electrodes. The only way to meet the requirements of the long term energy constraint is to perform data reduction to be able to transmit less data. For all data reduction approaches it is therefore important to capture sufficiently detailed information about the spike waveforms, such that these can be assigned to distinct neurons remotely (e.g., through a process called spike sorting [86]).

Mainly three different methods for data reduction exist as indicated by the three root nodes in Figure F.2. The first method uses compression techniques to compress the samples of the recorded data, thus transmitting less bits. The second method is to select recording sites along slender probe shafts independently for multiple channels, such as the Electronic Depth Control (EDC) technique [107]. The third method for data reduction is to discard irrelevant data. Compression and EDC techniques are currently used to transmit continuous recordings in ”raw form”, but may not be enough to meet the energy constraints of future high electrode count

probes. Hence, a need is present for data elimination in order to meet the stringent requirements for data transfer.

The signal analysis circuitry should be integrated on the probe closer to the electrodes in order to reduce the data to be transmitted as early as possible. Neural function is affected by temperature so any solution must not increase the brain temperature by more than 1-2 °C [120]. This limits how much short term processing is allowed on an implant. In high-density electrode arrays the heat generation and energy consumption related to the signal filtering, digitization, and especially data transfer, increases with the number of electrodes. In addition, the space available for signal analysis circuitry is limited, which poses a significant challenge. To avoid overheating of brain tissue, state-of-the-art solutions like the one presented by Raducanu et al. [88] place the signal processing and transmission circuitry on top of the probe outside the brain.

Different approaches have been proposed to meet the challenges presented in this section, such as online spike detection [108] and compression [128] solutions. Common for all approaches is their attempt to reduce the data to be transmitted by extracting the most valuable information. The definition of "valuable" is mainly dictated by what will be done with the data in further analyses. This means that the chosen data reduction technique should match the data analysis goals. These solutions will be discussed in Section F.2. In this paper we focus on spike data extraction for remote post-processing, such as spike sorting and clustering. We propose a novel and effective data reduction technique, which keeps the run-time and storage overhead limited. The technique is mainly comprised of a digital online temporal spike detector implemented on the probe (the part outside the brain), based on voltage amplitude thresholding in combination with a spatial data extraction mechanism. This technique solves the energy problem by performing data selection, to reduce the data to be transmitted significantly. This is, to our knowledge, the first spike detection method that performs spatial and temporal analysis locally on the probe. This algorithm/architecture co-optimization technique identifies and extracts essential spike data in a way that combines accuracy and area/energy efficiency. The spatial mechanism extracts all samples around neighboring electrodes when a threshold crossing is detected. This data extraction allows remote post-processing algorithms to get sufficient information for every detected spike.

Related work and alternative solutions are presented in Section F.2. In Section F.3, we give a brief overview of the general goal of the data reduction and post-processing steps in order to better understand our proposed tech-

nique. Section F.4 discusses wireless requirements needed to transfer large amounts of data. The temporal and spatial spike data extraction technique is presented in Section F.5, and the estimated hardware (HW) requirements for our technique is discussed in Section F.6. Sections F.7 and F.8 presents our experimental setup and quality metrics, respectively. We present and discuss our results in Section F.9. Finally, we present our conclusions in Section F.10.

F.2 Related work

Different methods for data reduction exist today, and one common aim for all solutions is to reduce the loss of potentially wanted data (false negatives). Some of these methods are implemented with analog circuitry and other solutions use digital logic for the spike detection. Alternative solutions to spike detection focus on data compression [128], or using EDC [107]. These alternative solutions can also be used complimentary to a spike detection solution. All solutions presented in this section are summarized in Figure F.2.

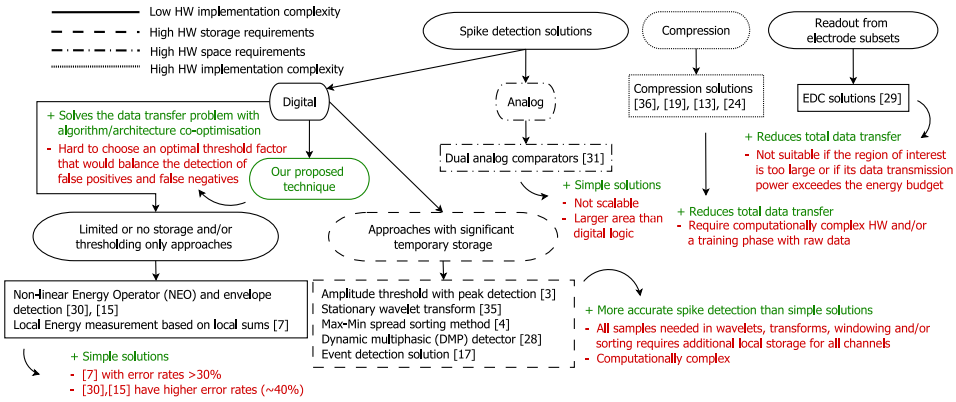


Figure F.2: Summary of current spike detection solutions.

Solutions using analog circuitry, such as detection with dual analog comparators [115], work well when the number of electrodes is small. For high-density electrode arrays, the cost of the analog comparators in terms of area limits the number of electrodes that can be supported, compared to simple digital logic. Solutions using digital circuitry can be divided into two categories: those with low complexity and minimal local storage re-

quirements, versus solutions that have higher complexity and/or large local storage requirements.

Advanced digital solutions need temporary storage for history, which is used for sorting, filtering and/or wavelet transforms. An example is spike detection using a stationary wavelet transform for the adaptive threshold [122]. Another example, such as a FPGA solution combining amplitude thresholding and peak detection on +/- 1 ms samples [11], requires data buffering. Other spike detection solutions using adaptive thresholds based on a max-min spread sorting method [17], analyze every 2 ms of recorded data with 50% overlap. This solution also requires buffering and sorting. Solutions using a dynamic multiphasic (DMP) detector [106] or solutions with event detection based on median and baseline estimation [75] require storage for data sorting.

Simple digital solutions on the other hand use limited or no temporary storage for spike data history, and/or use simple thresholds. Examples are spike detection with adaptive thresholding, based on Non-linear Energy Operator (NEO) and envelope detection [108], [59]. These solutions are simple to implement and only require storage for the previous and next sample for all sensor nodes. Other solutions with limited or no temporary storage use a local energy measurement based on local sums as thresholds for spike detection [26]. However, these simple digital solutions typically have error rates higher than 30%.

In general, advanced digital solutions require additional temporal storage per sample for all electrodes using a solution based on wavelets, transforms, windowing and/or sorting. They are therefore taking too much area, or produce too much energy overhead. Analog solutions usually do not scale well compared to digital solutions, and they are also more complex to implement. Simple digital solutions have error rates above 30%. To our knowledge, only temporal spike detection is performed locally on the probe in existing solutions, where the spatial spike data analysis or selection is performed remotely.

Alternative approaches focus on data compression, combined with spike detection. These are considered as a separate root node in Figure F.2. Zhang et al. [128] proposes a signal dependent compressed sensing (CS) approach used to compress detected spikes before they are transmitted off-chip. Their solution outperforms previous compression works in terms of compression rate and reconstruction quality. However, this solution is dependent on a dictionary matrix (D-matrix) created offline before the CS-circuit is activated for compression. The D-matrix is based on recorded raw signals

that are transmitted off-chip. A dictionary matrix requires significant storage per electrode if different channels are compressed independently. Other compression solutions exist based on wavelet transforms ([81], [57]), and threshold crossing and windowing ([93]). Compression solutions based on wavelet transforms are able to achieve good compression rates while maintaining excellent signal reconstruction quality. On the other hand, these solutions require complex hardware (HW). Simpler compression solutions based on threshold crossing and windowing require less complex HW. For very high-density electrode arrays, any solution that requires buffering, such as windowing approaches, would increase the amount of required storage significantly. Reducing the amount of required temporal storage per sample is therefore a necessity in high-density electrode arrays.

Another approach is to record from only a subset of available electrodes with EDC [107]. This technique is already applied in recent probe designs (i.e. "electronic depth control" probes) that use a switching matrix to select electrodes of interest. This is a valid approach, since it is likely that some electrodes are not positioned close to neurons, not in brain regions of interest, or some electrodes may be noisy. This approach is good when the specific region of interest is small enough. In cases where the region of interest is too large for continuous recording, a simple and more energy efficient solution is needed to meet the energy constraints. Such a solution must be scalable with increasing number of electrodes, and use a minimal amount of local storage per processed sample.

As mentioned, simple temporal spike detection solutions have high error rates. These error rates include both false positive and false negatives. From a data reduction perspective, it is desirable to reduce the false positives, but this usually comes at the cost of an increased number of false negatives. In contrast, if detecting as many spikes as possible is important, then it is desirable to reduce the false negatives, at the expense of higher false positive rates. The trade-off is usually controlled by the value of a threshold (or configurable scaling factor). The threshold selection depends on how the experimenter wants to weight the false positives and negatives. Hence, no single optimal threshold exists. Our focus is to select enough spike data locally for a successful remote post-processing, where the desired result is fine-tuned. Our proposed solution falls between the simple and advanced digital solution category. Hence, we optimize the energy consumption by adding only the required storage needed to preserve enough spike samples for successful remote post-processing.

F.3 General goal of data reduction approaches

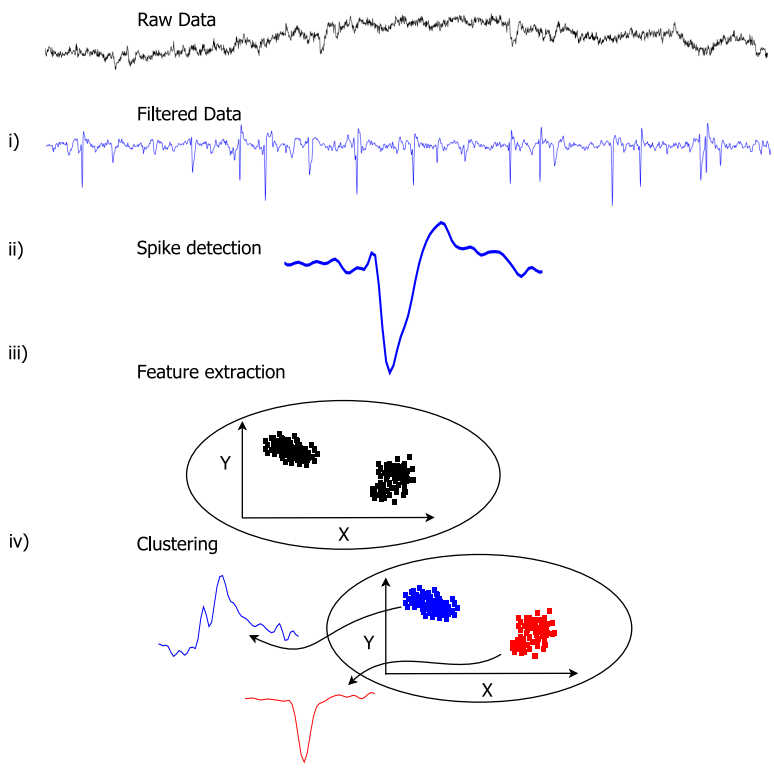


Figure F.3: Typical spike sorting overview as illustrated in [86].

The goal of data reduction approaches is to reduce the transmitted data while preserving all relevant information. What constitutes relevant information is application dependent. Spikes generated by individual neurons are the result of integration of input signals and transmitted to other neurons. For some applications it is sufficient to know the time of a spike regardless of the neuronal source, for example neural decoding approaches based on activities from multiple neurons (multi-unit activity) [101]. However, for most applications additional information about the wave shape is needed, as this relates to the specific neuron that emitted the spike (different neurons emit spikes with different wave shapes). Wave shape features can be used to improve neural decoding approaches [58]. More often, waveform features are used to identify groups of spikes that originate from the same neuronal source ("spike sorting"; [86]). The spike sorting process relies on differences

in intrinsic properties of neurons and their location relative to the electrodes, which are reflected in the spike amplitude and wave shape. After detection and extraction of spike waveforms from high-pass filtered (>300 Hz) raw signals, the sorting procedure is usually performed by deriving informative features from the waveforms (e.g. peak amplitude or components from principal component analysis, PCA[2]) across a set of electrodes, followed by a feature-based classification algorithm, as summarized in Figure F.3. Step iii in Figure F.3 shows an illustration where the x- and y-axis represents the first and second PCA components, respectively. In the final spike sorting step, similar features are clustered together with a label representing its cluster. Conventional approaches transmit the filtered data from the neural probe to a remote receiver, and process steps ii to iv remotely. The goal of our approach is to detect relevant spike waves (temporally and spatially) with minimal amount of resources on the neural probe. We then reduce the amount of data by only transmitting these waveforms to a remote receiver, where steps iii to iv are processed remotely.

F.4 Wired and wireless data transfer limitations

Existing wired high-density electrode arrays with close to 1000 electrodes are able to transmit data in its raw form through wires connected to the probe shaft. As motivated in Section 1, the transmission circuitry in wired solutions contributes significantly to the power consumption, which will be even higher for wireless transfer. As an example, a probe with 120 electrodes is estimated to require between 50 and 58 Mbps to send all data in its raw form. This estimate is based on a 25 kHz sampling rate, using 16-bit samples. In our estimate we have assumed the coding overhead to be between 5% and 20% of the required bandwidth [80], i.e., the processing overhead required to prepare and transmit the data to the remote wireless receiver. Today's wired solutions have 2-5 meters cables running from the head stage that sits on the probe to the readout system (like a PC). The distance between the transmitter and the receiver is therefore assumed to be up to 5 m in our power consumption estimation for wireless data transmission.

Wireless low-power networks such as Personal Area Networks (PAN) and Body Area Networks (BAN) cannot handle the throughput requirements for 120 electrodes, because these networks are designed for lower bandwidths. To meet our bandwidth requirements we need to use networks similar to the wifi 802.11n or 802.11ac standards.

Halperin et al. [44] cites 1.28 W for the 802.11n transmitter in single-input and single-output (SISO) mode (which is barely sufficient for our 120 channel example). This power consumption is nearly independent of the exact data rate, unless data is buffered up and transmitted in chunks. Chunking can reduce the power consumption, but requires additional bandwidth overhead which is simply not possible in SISO mode with our bandwidth requirements.

The 802.11n multiple-input and multiple-output 3 (MIMO3) mode allows up to 405 Mbps for 2.1 W at the cost of multiple transmitters on-chip. In the most optimistic scenario (leading to a lower bound), the required power then becomes a fraction as shown in Equation F.1, because the power consumption will depend on the data rate when it is buffered in chunks.

$$\text{Power consumption} = \frac{\text{data rate}}{405 \text{ Mbps}} \cdot 2.1 \text{ W} \tag{F.1}$$

Without any data reduction the estimated 50 to 58 Mbps transfer requires a power consumption of more than 1 W in SISO mode and 0.30 W in MIMO3 mode in addition to the buffering cost. If the amount of data is reduced 8 times, the power consumption for data transfer would become less than 40 mW.

Also more advanced BAN oriented wireless transceivers are being developed but in the end their power efficiency will still not remove the bandwidth-energy bottleneck fully and similar equations as Equation F.1 will be valid, with other constants. We believe our general approach will hold also for those.

If the data is streamed with lower speed and bandwidth by means of buffering, we would have latency violations. Nevertheless, many applications have feed-forward processing so that added latency would be fine. However, a more severe limitation in our scope is that the probe does have very strictly limited space. This prevents us from implementing a reasonable sized buffer.

Figure F.4 shows our envisioned platform where the electrodes are filtered through filters on the probe, before all samples are time-multiplexed into our custom ASIC. Other researchers have successfully implemented analog filters on the probe [64]. Data to be transmitted are then buffered up and transmitted through the radio circuitry. It should be noted though that the actual layout may change for future probe designs.

The size of the probe is too small to carry anything but a very compact ASIC, which is custom made as illustrated in Figure F.4. To transmit the full data set out to an external implant will therefore not work for high

density electrode arrays. Data elimination is hence essential as an initial stage for any other processing on external devices.

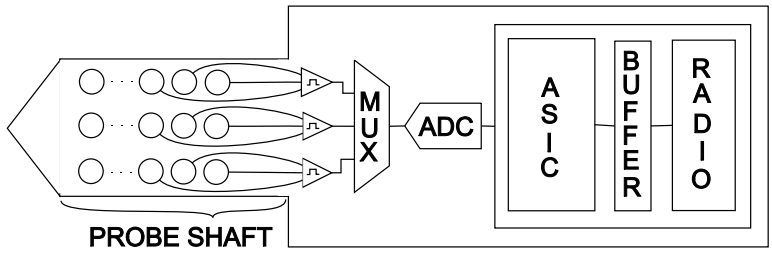


Figure F.4: Block scheme of our envisioned platform.

F.5 Temporal and spatial spike data extraction

We focus on spike data extraction, using low power and simple online digital circuitry. Samples around detected spikes are selected locally on the probe, and all other data are discarded before the selected samples are transmitted to a remote receiver for post-processing. This solution is scalable with increasing number of electrodes, and uses a minimal amount of local storage. Pruning away unneeded data results in large reductions in transmit energy, because we transmit significantly less data with minimal increase in processing/memory access energy only.

We have divided the spike data extraction in three consecutive phases as illustrated in Figure F.5. All these phases are performed for each sample per electrode. In the initial phase, a temporal spike detection is performed using a crude amplitude thresholding mechanism. The second phase performs a spatial analysis of all detected spikes and marks neighboring samples for extraction. In the final phase, all samples marked for extraction are transmitted. All three phases are described in more detail in Sections F.5.1, F.5.2 and F.5.3.

F.5.1 Phase 1: Temporal spike detection

As motivated in Section F.2, we use a threshold crossing method to detect spikes. Our adaptive threshold calculation is based on a simple amplitude

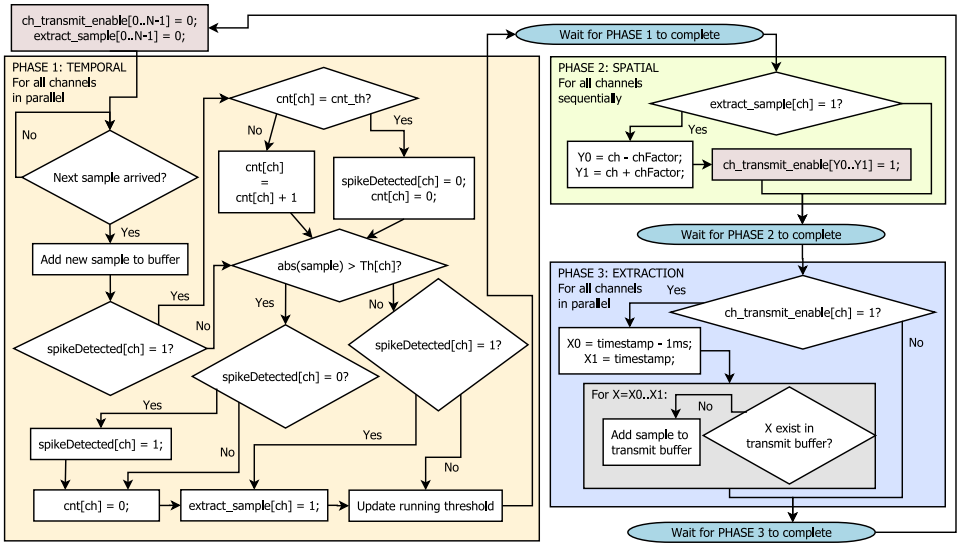


Figure F.5: 3-phase spike data extraction

threshold solution used by Jadhav et al. [53], [52], where they use five standard deviations above the mean as threshold.

In our approach, we use the mean (M), together with a factor (ThFactor) times the standard deviation (STD) of samples as a crude threshold for spike detection. Based on the stringent HW and energy requirements for wireless data transmission, the algorithm cannot afford to store more than a few temporary samples when the threshold is calculated locally on the probe. Other amplitude threshold spike detection solutions using a median calculation, such as the one surveyed by Rey et al. [92] and the absolute value method validated by Dragas et al. [26], requires additional storage to calculate the median value. Therefore, we implement a running mean and sample variance calculation algorithm, based on Welford's method for calculating the sum of squares [117], as our crude thresholding algorithm.

The sample variance algorithm is chosen simply because the underlying distribution is not known at run-time. The standard deviation is estimated using the square root of the bias corrected sample variance [116], which is shown in Equation F.2. Equation F.3 and F.4 shows the calculation of the running mean and sum of squares, respectively. In the following equations, x is the absolute value of the sample, k is the sample number, V is the variance, S is the sum of squares, STD is the standard deviation, and M is the mean value.

$$STD_k = \sqrt{V_k},\ V_k = \frac{S_k}{k-1} \tag{F.2}$$

$$M_k = M_{k-1} + \frac{1}{k}(x_k - M_{k-1}) \tag{F.3}$$

$$S_k = S_{k-1} + \frac{k-1}{k}(x_k - M_{k-1})(x_k - M_{k-1}) \tag{F.4}$$

Our method for calculating the sample variance is based on a one-pass updating algorithm with comparable accuracy to the two-pass method derived form the definition of variance. One-pass updating algorithms are also recommended by Chan et al. [18] when two-pass algorithms cannot be chosen due to storage limitations. Equation F.3 can be re-structured as shown in Equation F.5, and we use it to replace the $\frac{k-1}{k}(x_k - M_{k-1})$ part of Equation F.4 with $(x_k - M_k)$, which is equivalent. We therefore use Equation F.6 for S_k to simplify the HW implementation.

$$M_k = M_{k-1} + \frac{1}{k}(x_k - M_{k-1}) = \frac{1}{k}x_k + \frac{k-1}{k}M_{k-1} \tag{F.5}$$

$$S_k = S_{k-1} + (x_k - M_k)(x_k - M_{k-1}) \tag{F.6}$$

Our adaptive threshold mechanism calculates the running mean and standard deviation (STD), and uses the result to calculate the running threshold according to Equation F.7. Compared to Jadhav et al. [53], [52], we replace the factor five with the configurable ThFactor. In addition, we use the running mean and STD formulas presented in this section.

$$Th_k = M_k + (ThFactor \cdot STD_k) \tag{F.7}$$

In Equation F.7, Th is the running threshold, M is the running mean, STD is the standard deviation, and ThFactor is a configurable factor that is used to tune the sensitivity of the threshold. All samples with an absolute value above the threshold are marked for extraction by setting the variable extract_sample to 1 for this channel (Figure F.5). At the same time the variable spikeDetected is set to 1. The calculation of Th_k and marking of samples is performed for all electrodes individually and simultaneously in parallel. When a threshold crossing is detected on the absolute value of a sample, the last millisecond of previously buffered samples are marked for extraction (if they are not marked from before), and a counter (cnt) starts counting for the inclusion of the next millisecond of samples. With a frame

rate of 25 kHz, extracting 1 ms of samples (approx. 25 samples) before a threshold crossing with a buffer and after a threshold crossing (using a counter without additional buffering) is assumed to include sufficient samples around the spikes (i.e., peri-spike samples) for post-processing. This extraction of data is also a requirement in related work for post-processing. In Figure F.5, the variable cnt_th is used to determine the length of this extraction period. spikeDetected is set to 0 as soon as the 1 ms period has passed.

From Equation F.2 and F.3, we can observe that new sample values have smaller impact on the running mean and standard deviation when k increases. Depending on the goal of the spike data analysis, the designer or neuroscience expert may want to reset the variance history periodically. We therefore include another tunable parameter called the Variance History Update (VHU), which resets the previous history of M and STD values (i.e., variance history). The Variance History Reset Interval (VHRI) is determined by the VHU parameter and the sampling frequency (FS) according to Equation F.8, where FS is equivalent to 1 second of data samples (25k samples for 25 kHz sampling frequency). In other words, when $k = \text{VHRI}$ then the variance history is reset. The VHU variable must be stored in a register, and the size of this register may vary depending on how often the designer needs to reset the variance history.

$$\text{VHRI} = \text{VHU} \cdot \text{FS} \tag{F.8}$$

Using up to 18-bits for VHU is sufficient to have variance history reset periods of more than 47 hours, given a k-register of 32-bits (the k-register wraps around to zero after 2^{32} samples at a rate of 25k samples/second, i.e., after 47.7 hours). If longer recording time is needed, then the VHU- and k-register must be adjusted accordingly. The VHU can be removed completely if no reset of the variance history is needed (the k-register must still be large enough).

F.5.2 Phase 2: Spatial spike data analysis

Phase 2 is a sequential phase that iterates through all electrodes one by one. If extract_sample is marked on an electrode channel, then it marks this sample for transmission by setting the variable ch_transmit_enable to 1 (Figure F.5). A specific number (chFactor) of closely placed electrode channels surrounding the marked channel is also marked for transmission. Phase

1 has to be completed for all channels before Phase 2 can start, because the analysis of each electrode sample is dependent on the situation of all other electrode channels in the same time stamp. In other words, all channels are processed sequentially in Phase 2, where ch_transmit_enable is set for all channels that are close (in distance) to a channel with extract_sample marked (determined by the chFactor tuning parameter).

How the neighboring channels are selected is determined by a geometrical matrix unique for each probe design based on the placement of each electrode. It is up to the probe designer and neuroscience expert to determine which electrodes are neighbors to each other in the geometric matrix. The HW mechanism which selects the electrode neighbors should be as simple as possible because the sequential Phase 2 can become the bottleneck of the application if it takes too much time. One simple solution is to assign a unique channel id for each electrode, such that the neighboring electrodes can be marked by incrementing and decrementing the channel id according to the chFactor. In our experiments, we use electrode channel ids starting from 0 for the electrode at top of the probe and using ascending channel ids on probes placed below. The spatial extraction is then done according to PHASE 2 in Figure F.5.

F.5.3 Phase 3: Spike data extraction

When all the samples to be transmitted are marked, our algorithm enters Phase 3. Phase 3 is a parallel phase where for all electrodes simultaneously it is checked if the corresponding ch_transmit_enable is marked. If it is marked, the previous 1 ms of samples (including the current sample) are added to a transmit buffer, unless they have already been added. After Phase 3 completes, the ch_transmit_enable and extract_sample variables in Figure F.5 are reset for all channels.

F.6 Estimated HW requirements

The HW requirements are estimated assuming there is one time-multiplexed arithmetic logic unit (ALU) available for processing, and that output buffers and other necessary logic are available. Our local storage requirement only considers the storage required to process the data from all electrodes, which

is illustrated in Figure F.6.

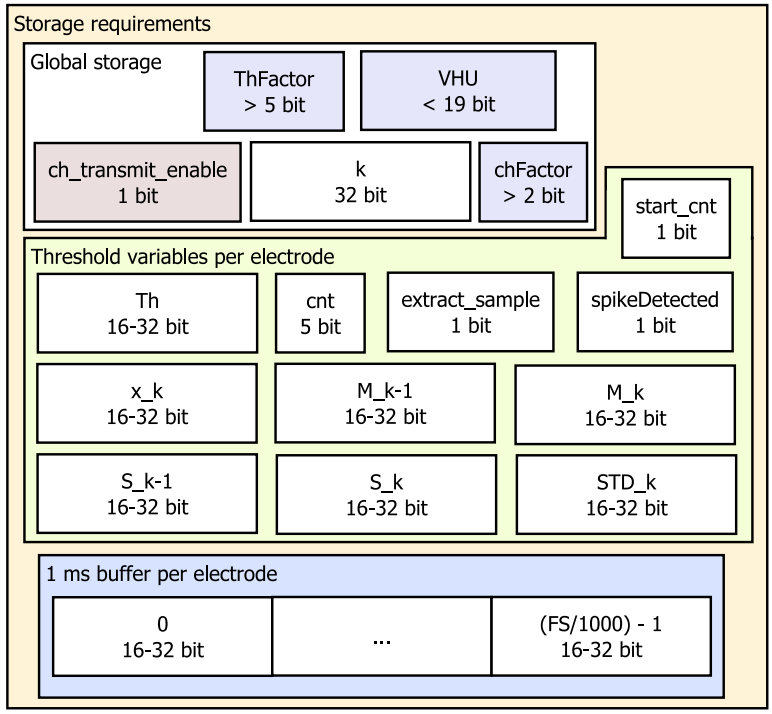


Figure F.6: Storage estimate

Three control bits are required per electrode to mark samples for extraction, detected spikes and to start a counter to include the future 1 ms samples. One global control bit is needed to transmit the samples marked for extraction. A 5-bit counter (supports up to 32 kHz sampling rate to count 1 ms of data) and a 1 ms buffer (25 samples for 25 kHz sampling rate) of 16- to 32-bit samples is required per electrode to include enough peri-spike samples before a spike is detected. A 16- to 32-bit threshold register is needed with six 16- to 32-bit variables for intermediate mean, STD and absolute value (ABS) variables, which are used to calculate the running means and STD values. All these registers are needed per electrode. 32-bit variables are only needed if high signal accuracy is required, but can be anywhere between 16 and 32 bit. More bits per sample will require more storage. In our estimate we consider either 16- or 32-bit samples. A global 32-bit register is needed for the k-register in order to support up to approximately 47 hours of neural recordings (if FS = 25 kHz) without

variance history reset. This register can be smaller or larger depending on how long recordings the probe is designed for. Finally, we need to store the configurable tuning variables ThFactor (at least 6 bits), chFactor (at least 3 bits) and VHU (up to 18 bits) globally. If 16-bit variables are used for each sample, then 520 bits are needed per electrode, and 1032 bits per electrode if 32-bit variables are used. In addition, 60 bits are needed for the global transmit bit, ThFactor, chFactor, VHU and k parameters. The storage for each electrode requires the memory to be accessed concurrently, i.e., one port for each electrode if one bank of SRAM is used, which is not feasible unless one bank of SRAM is used for each electrode. 6T SRAMs requires less area than standard cell-based memories (i.e., memories consisting of standard cells only) [35]. However, Gemmeke et al. [35] show that although near threshold voltage (NTV) SRAMs have an area benefit, the use of cell-based memories have a better power benefit. Andersson et al. [6] makes the same conclusion, i.e., cell-based memories are up to 4x faster than SRAMs with orders of magnitude lower energy dissipation (at the cost of larger area per bit for large memory sizes only). Andersson et al. [6] also mention that standard cell memories scale with less effort than custom SRAMs.

In Table F.1, we have estimated how many electrodes are supported for 16- and 32-bit samples per 5 mm^2 digital chip using a 40 nm process from Taiwan Semiconductor Manufacturing Company (TSMC) with and without 3D stacking. For simplicity we assume 6 bits for ThFactor, 3 bits for chFactor, 18 bits for VHU, and 32 bits for k in our estimation. The area requirement assumes one D edge-triggered flip-flop for 1-bit register, and 6 nand gates per flip-flop. 40 nm TSMC supports up to 2000K gates/mm^2.

Table F.1: Supported number of electrodes using 40 nm cell-based memories

	16-bit samples	32-bit samples
5 mm^2 (TSMC)	3200	1610
3D stacking (TSMC)	6400	3220

The required area for ALU, output buffers and other logic are not included in the estimation, but are expected to be relatively small. For instance, in each electrode Phase 1 performs approximately 4-5 additions, 4 subtractions, 2 divisions and 3 multiplications per sample in addition to calculating a square root and absolute value. Therefore, a basic time-multiplexed ALU can be configured for a group of electrodes instead of having one ALU per electrode. The 1 ms buffer is intended to be a part of the transmit buffer, where we have assumed that up to 20% coding over-

head is required for the transmit system (based on the discussion in Section F.4), depending on what kind of wireless technology is used. Phase 2 and 3 perform significantly less calculations because these phases mostly enables control bits and transmits samples.

F.7 Experimental setup

Our 3-phase approach is tested on data obtained with a 128-channel silicon probe in the visual cortex of an awake mouse during the presentation of visual stimuli. Signals were digitized and stored using the Janelia WHISPER recording system (https://www.janelia.org/node/46162). Data were acquired at NERF using a 128-channel passive probe developed at IMEC. More recently, a newer passive probe with 966 electrodes and 384 configurable channels has been developed at IMEC [64]. These recordings are processed with our 3-phase approach in Matlab, where the size of the extracted data is compared relative to transmitting all samples, i.e., the reduction in transmitted data.

Our extracted spike data are also compared to the results from unsorted spike data after a flood fill algorithm [94] performed remotely offline on the data set. I.e., when spikes are detected the algorithm finds the mask for each spike using upper and lower threshold bounds. The method using the flood fill algorithm is a much more advanced method compared to our simpler method. The flood fill method is also not feasible for local low power implementation on the probe because of its complexity. Therefore we use the results from the flood fill algorithm method as our golden reference. It should be noted that even the flood fill algorithm results on our data set contain false positives and false negatives which are not annotated. Publicly available annotated data sets for spike detection only contain annotated data for one channel. Therefore, we are unable to validate our method with such data sets because our method performs spatial analysis across channels. To overcome this literature limitation, we use the flood fill algorithm results as our golden reference, and validate the quality of our algorithm relative to the flood fill algorithm results. We present our quality metrics in Section F.8, before introducing the data reduction and quality results in Section F.9.

The signal loss of our method (i.e., spike amplitudes below the threshold) will depend on what the focus of the experiment is. E.g., if detecting large spikes is a priority, then it could be considered that spikes below the

threshold is not of any interest. This will in other words strongly depend on the neuroscientist analysing the data and the intended post processing steps. This aspect is thus very domain specific.

F.8 Quality metrics

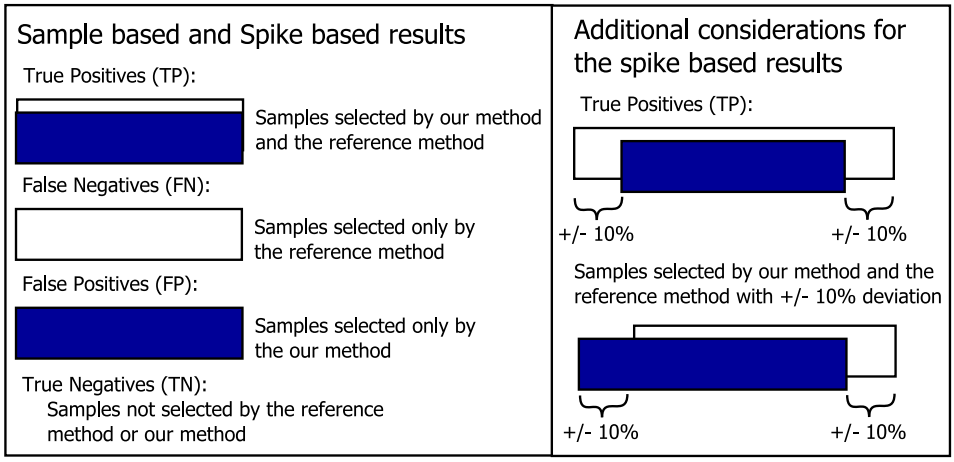


Figure F.7: Definitions for quality measurement.

We measure the quality of our method by collecting the amount of false positives (FP), false negatives (FN), true positives (TP) and true negatives (TN) compared to our reference algorithm. We use rate metrics based on receiver operation characteristics (ROC) analysis [28], [85]. How we define these parameters is illustrated in Figure F.7. TPs are samples extracted by our method and the reference algorithm. FPs are samples extracted by our method and not the reference algorithm. TNs are samples that are not extracted by any of the methods. The FNs are the samples extracted by the reference method and not our method.

Lower FNs means higher sensitivity for our method compared to our reference algorithm. Equation F.9 shows the sensitivity, i.e., the TP rate (TPR) as a function of TPs and FNs. TPR is a value between 0 and 1, where 1 means that there are no FNs, i.e., our method extracts all the samples that are extracted by the reference algorithm. The higher the TPR value, the better is the quality. Equation F.9 also shows the FP rate (FPR). The FN rate (FNR) of our method and the precision is shown in Equation

F.10. The FPR and FNR metrics indicate the errors compared to our reference algorithm, having values between 0 and 1, where 0 is optimal. The precision of our method, i.e., the positive predictive value (PPV) indicates how much of the detected spikes are TPs (between 0 and 1, where 1 is optimal). A low PPV value therefore indicates that additional samples (FPs) are transmitted.

$$TPR = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}, \; FPR = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}} \tag{F.9}$$

$$FNR = \frac{\mathrm{FN}}{\mathrm{TP} + \mathrm{FN}}, \; PPV = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} \tag{F.10}$$

The accuracy of our method, i.e., how well our method is able to find the TP and TN compared to the reference algorithm, is calculated according to Equation F.11.

$$ACC = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{FP} + \mathrm{FN} + \mathrm{TN}} \tag{F.11}$$

F.9 Data reduction and quality results

In Section F.9.1, we present our data reduction results. The quality results of our method is presented in Section F.9.2. All our results are discussed in Section F.9.3.

F.9.1 Reduction of transmitted samples

The amount of extracted data is compared to the amount of raw data for two different threshold sensitivities. Figure F.8 shows the data reduction in percent by using our methodology on 0.1 second (2500 samples) of the recorded data, averaged over different recording lengths up to 10 minutes.

The data reduction varies for short recording lengths, but converges towards approximately 78-80 % and 83-84% for our two ThFactor settings over longer recording lengths. This is confirmed by the results in Figure F.9, which shows the total data reduction in percent of our complete 55 minute recording.

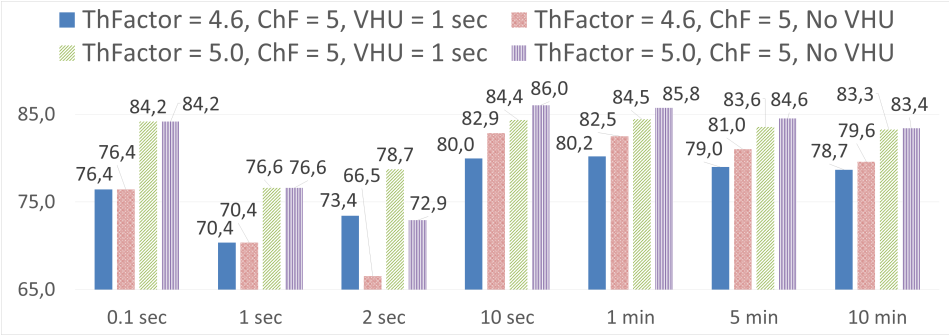


Figure F.8: Data reduction (%) of 0.1 second neural data averaged over various recording lengths up to 10 minutes.

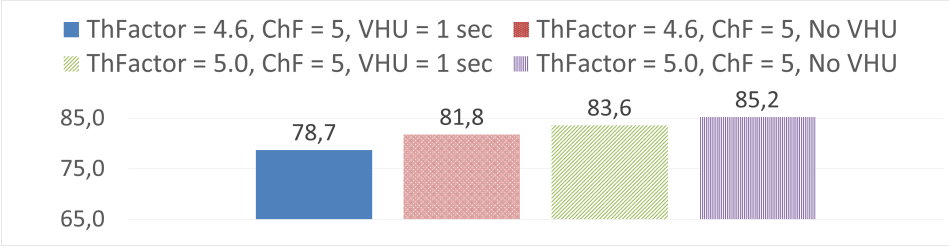


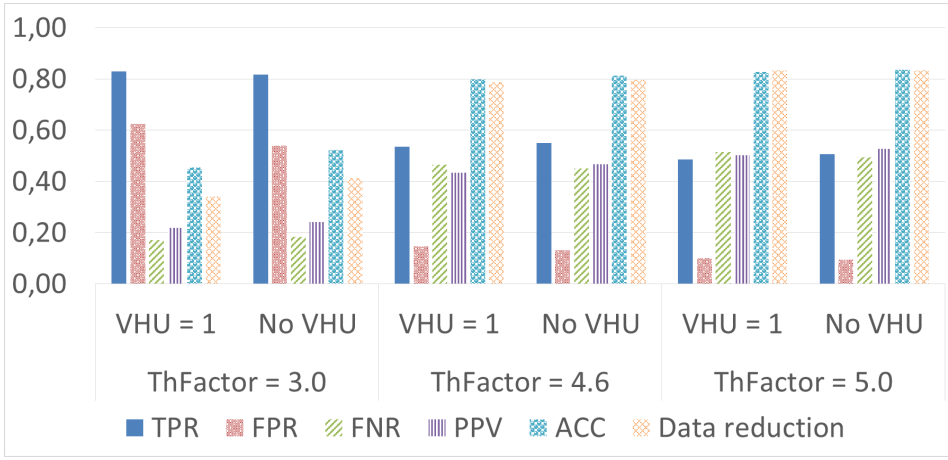
Figure F.9: Total data reduction (%) of a 55 minute recording.

The VHU parameter does have an impact in reduced data when the variance history is reset for every recorded second compared to not resetting the variance history. The impact is however only about 3%.

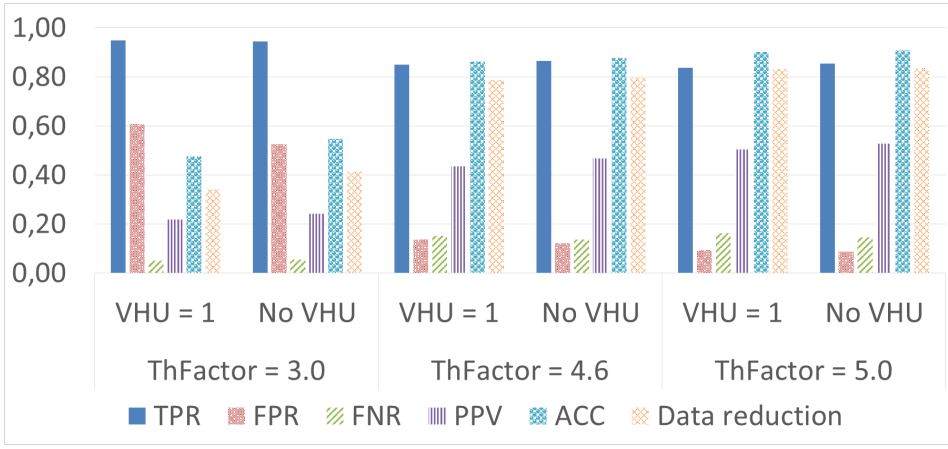
We have estimated the BW requirement based on our results from the ten minute recording averaged over 0.1 second chuncks. If we assume 5-20% coding overhead to transmit the data, the required bandwith is between 10.3 Mbps and 11.8 Mbps (with ThFactor = 4.6, no VHU) or between 8.4 Mbps and 9.6 Mbps (with ThFactor = 5.0, no VHU). If a compression algorithm is used on top of our data reduction method, the bandwidth requirement would most likely decrease even further.

F.9.2 Quality results

We have measured the quality of our method by calculating the metrics described in Section F.8 for averaged 0.1 second segments of the recorded data, over different recording lengths. The quality results of our method



(a) Sample-based results of our method on averaged 0.1 sec segments over a 10 minute recording.

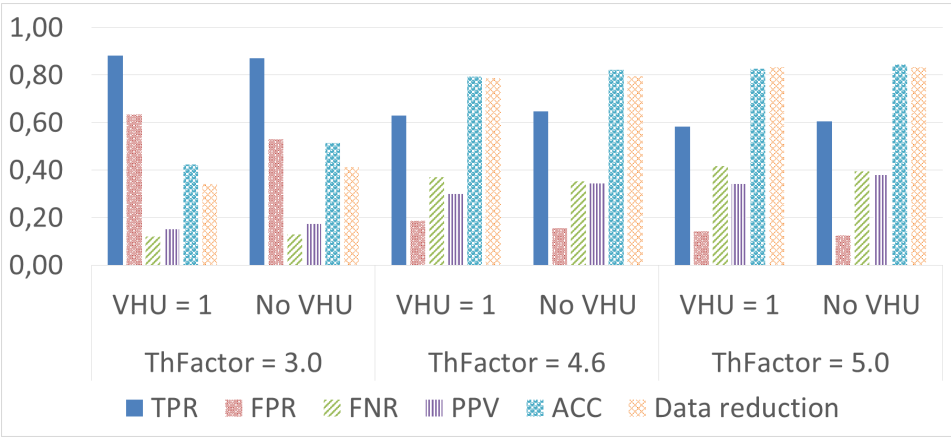


(b) Same results as Figure F.10(a) when ignoring reference spikes with amplitudes lower than 30% of Th.

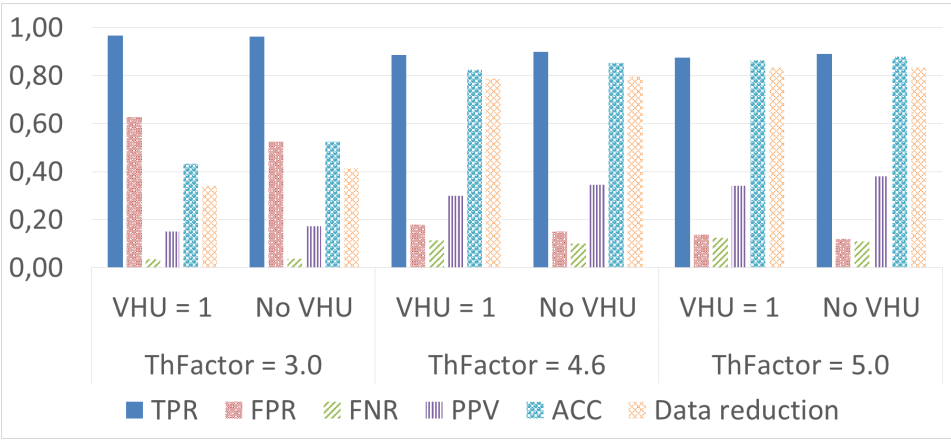
Figure F.10: Sample-based quality results.

converges in a similar way as our data reduction results. We therefore present our quality results for averaged 0.1 second segments over a 10 minute recording in Figures F.10 and F.11.

These results have been calculated by counting samples (sample-based) and spikes (spike-based), where multiple spike time stamps within a 1 ms (25 samples) window (e.g., such as overlapping spikes) are considered as the same spike in the spike-based results. The results show that our method



(a) Spike-based results of our method on averaged 0.1 sec segments over a 10 minute recording.

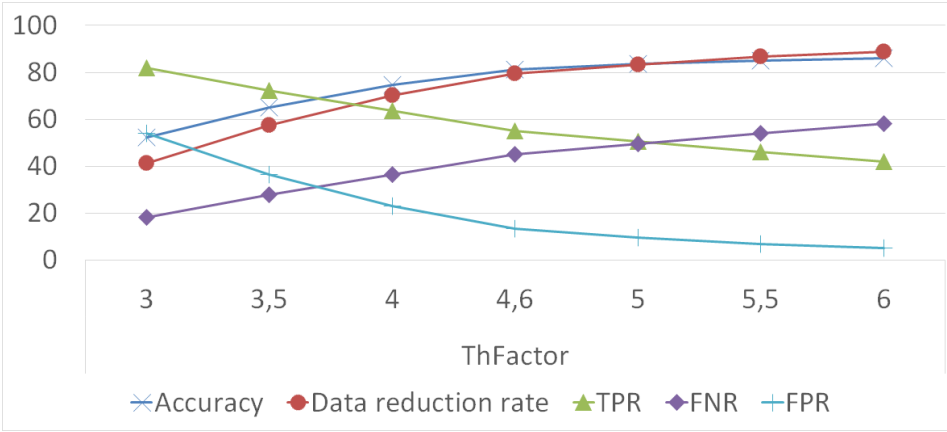


(b) Same results as Figure F.11(a) when ignoring reference spikes with amplitudes lower than 30% of Th.

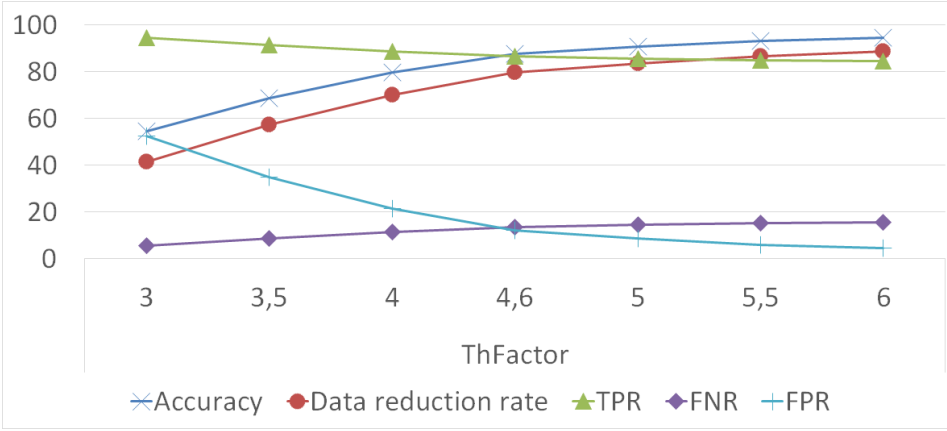
Figure F.11: Spike-based quality results.

reduces the amount of data to be transmitted significantly at the expense of discarding spike data with amplitudes significantly lower than the threshold crossings. In other words, our method is able to detect all spikes that crosses the threshold, and the majority of smaller amplitude spikes that are not detected have amplitudes lower than 30% of the threshold. We confirm this observation by calculating the metrics presented in Section F.8 where undetected spike samples (FNs) with amplitudes below 30% of our thresholds are ignored. These results are shown in Figure F.10(b) and

F.11(b), while Figure F.10(a) and F.11(a) show the results including all reference spikes. From Figures F.10 and F.11, we can observe minimal difference with or without the VHU factor. These figures also show that the FPR and TPR values are higher with lower ThFactor at the cost of lower accuracy and data reduction. For ThFactor around 4.6 and 5.0, we achieve a significantly improved accuracy and data reduction rate. However, if the ThFactor is too high, then the TPR decreases and the FNR increases as shown in Figures F.12 and F.13.

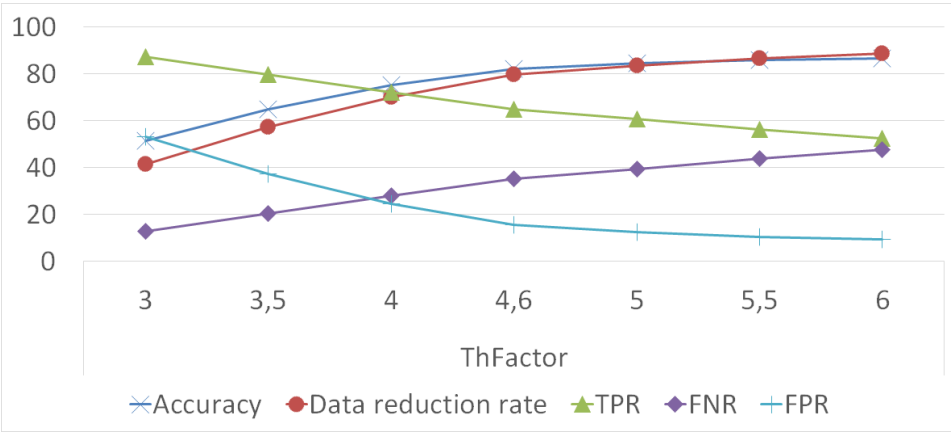


(a) Sample-based results of our method on averaged 0.1 sec segments over a 10 minute recording.

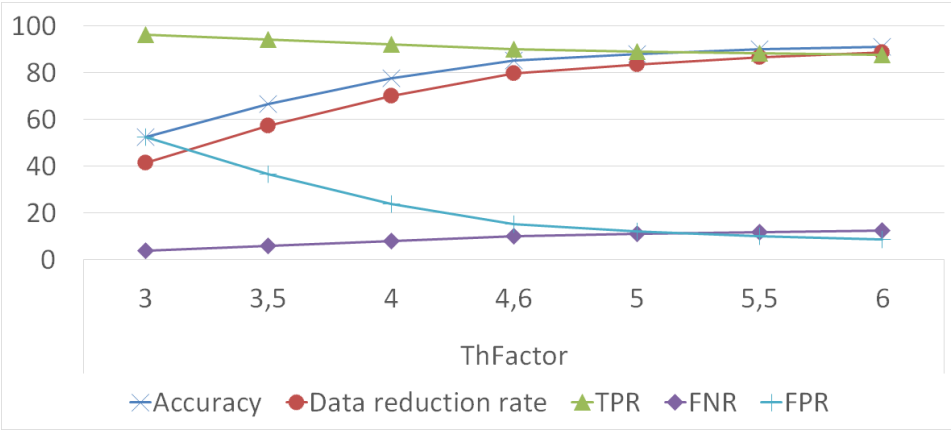


(b) Same results as Figure F.10(a) when ignoring reference spikes with amplitudes lower than 30% of Th.

Figure F.12: Sample-based quality results with different ThFactor values.



(a) Spike-based results of our method on averaged 0.1 sec segments over a 10 minute recording.



(b) Same results as Figure F.11(a) when ignoring reference spikes with amplitudes lower than 30% of Th.

Figure F.13: Spike-based quality results with different ThFactor values.

F.9.3 Discussion

Our methodology detects all the large spikes which the reference algorithm detects. Samples are discarded if they do not cross the threshold, and no nearby samples in time or on neighboring electrodes cross the threshold. The majority of all FNs have an amplitude below 30% of our threshold crossings, as shown in Figure F.10(b) and F.11(b). The spike data marked by the reference program uses a more complex thresholding algorithm to

detect spikes. Our FNs, which are a result of not having high enough amplitudes to cross our statistical threshold, are therefore assumed to be out of range for our coarse grained spike data extraction mechanism.

In Figures F.10 and F.11, the sensitivity (TPR) is between 0.55 and 0.65 when considering all FNs for ThFactor 4.6 and 5.0. Our ThFactor has to be decreased in order to increase the TPR, but at the expense of less data reduction (only up to 41%). Lowering ThFactor to 3.0 will also increase the FPR, and as a result reduce the precision (PPV) significantly. On the other hand, we obtain good TPR results when ignoring FNs below 30% of Th (TPR between 0.84 and 0.9 for ThFactor 4.6 and 5.0 in the sample-based and spike-based results). When ignoring FNs below 30% of Th, we also obtain improved results in FPR (0.09-0.18), FNR (0.10-0.16), PPV (0.30-0.53) and ACC (0.82-0.91).

The total reduction in data to be transmitted for our recording, relative to transmitting all the raw data, is significant for ThFactor 4.6 and 5.0 (as shown in Figures F.8 and F.9). Compared to compression techniques combined with spike detection, these data reduction results are similar [128]. However, our solution is scalable with increasing number of electrodes and does not depend on training phases with all the raw data available. We expect that solutions requiring the raw data to be recordable for all electrodes simultaneously will not be feasible to implement (see specifications and constraints in Section F.4 and F.6) in future neuronal probes due to the stringent energy constraints. A compression solution can be added as a complimentary solution to our approach where our reduced data-set is compressed before it is transmitted.

If the very low amplitude spikes are critical for the post-processing analysis, then the threshold must be lowered accordingly until all the important spike samples are detected, at the expense of more FPs and less data reduction. Since the majority of our FNs have amplitudes below 30% of the thresholds, we conclude that the spike samples our method does not detect simply come from neurons that cannot be well isolated with a simple thresholding mechanism. A clear tradeoff exists between the desired reduction required vs discarding the FNs which are below the threshold. When energy efficiency is the bottleneck of the system (i.e., in wireless probes) then significantly high data reduction rates are important in order to reduce the energy consumption significantly, as discussed in Section F.4 and shown in Equation F.1.

The highest bandwidth requirement is 11.8 Mbps in our measured case with ThFactor equal to 4.8. This means that our method would (in average)

buffer up less than 30 16-bit words for each iteration of our method. The analog frontend bandwidth used to collect all the raw data is in our case 48 Mbps (with 25 KHz sampling rate). Therefore, we assume that the on-chip communication bandwidth between the ASIC and the radio interface would not be a problem at 11.8 Mbps.

In general, our method reduces the amount of data needed to be transmitted significantly (up to 86%), while maintaining a high sensitivity when ignoring false negatives with very small amplitudes (below 30% of the threshold). Thus we conclude that the data transmit energy will be reduced significantly for high-density electrode probes when using our method. Since the HW required per electrode is independent of other electrodes, the required digital logic is scalable with increasing number of electrodes (at least up to 3200 for 40 nm TSMC).

F.10 Conclusion

We have proposed an energy efficient spike data extraction solution for high-density electrodes capable of reducing the data to be transferred by over 85%. Our proposal extracts spike data temporally and spatially locally on the probe, with minimal digital logic per electrode. Only the reduced data is transmitted to a remote receiver, making our proposal also applicable for wireless high-density electrode arrays. None of the spike detection solutions discussed in related work are able to achieve this significant reduction, while exhibiting an acceptable hardware overhead to enable local processing on the neural probe itself.

Spike samples for all the major spikes are extracted with our proposal, because the major spikes exceed our thresholds with good margin, and our spatial solution includes samples from all sub-threshold spikes in neighboring electrodes. Spikes in electrodes with no threshold crossings in their channels, nor any threshold crossings in nearby channels are considered irrelevant. These small spikes can be acquired by reducing the spike detection threshold at the cost of low reduction of the amount of data transferred. These usually refer to neurons that cannot be well isolated and are not necessarily relevant depending on the experimental conditions.

Our method is able to obtain an accuracy rate of up to 91% compared to the results of an offline flood fill algorithm when our threshold is five standard deviations above the mean value of the signal amplitude (ThFactor = 5.0). We obtain in general good quality when ignoring spike data

with amplitudes below 30% of the temporal thresholds (sensitivity = 0.85, precision = 0.53, false positive rate = 0.09 and false negative rate = 0.15 with ThFactor = 5.0).

The use of digital logic makes our proposal scalable with increasing number of electrodes. As an example, a 40nm TSMC process would support up to 3200 electrodes per 5 mm^2 digital chip and twice as much if 3D-stacking technology is used.