UNIVERSITY OF CALIFORNIA, SAN DIEGO

Adaptation Techniques to Mitigate Impact of Process Variations and Dynamic Thermal Management

A dissertation submitted in partial satisfaction of the

requirements for the degree Doctor of Philosophy

in

Electrical Engineering (Computer Engineering)

by

Ali Mirtar

Committee in charge:

Professor Sujit Dey, Chair
Professor Chung-Kuan Cheng
Professor Bill Lin
Professor Truong Nguyen
Professor Michael B. Taylor
Professor Anand Raghunathan

2015

The Dissertation of Ali Mirtar is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

Chair

University of California, San Diego

2015

DEDICATION

To my absolutely lovely, smart, hardworking, wonderful, and gorgeous wife, Dr. Sara Zarei who not only is my soul mate and my best friend but also is my role model. Her hard work and enthusiasm for achievements have encouraged me to try more and more until I accomplish the best. To my loving parents Mojtaba Mirtar and Khatoon Zarei who have taught me how to walk, talk, dream, and achieve. It would have not been possible for me to get to this point without their great sacrifices and decades of patience.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

This dissertation would not have been completed without the guidance, and support of many people. I would like to use this opportunity to pay my sincere appreciation to these great people who have made it possible for me to get to this point.

First and foremost, I would like to convey my sincere gratitude to Dr. Sujit Dey, my PhD advisor, for giving me the opportunity to work on this research. He has been a life mentor and role model for me throughout my PhD program. His invaluable support, aspiring guidance, helpful constructive criticism and friendly advices have provided me with the ability to complete this dissertation.

My sincere gratitude also goes to Dr. Anand Raghunathan, my PhD co-advisor, who has been a great support and guidance during my PhD program. Dr. Raghunathan and Dr. Dey were deeply involved in my research and they spent many hours discussing the technical parts of our research as well as shaping my writings structure to efficiently convey our idea. Their efforts and guidance had a great role on preparation of this dissertation.

I extend my love and appreciation to my loving wife, Dr. Sara Zarei. It would have not been possible to write this dissertation without her endless support, encouragement, and belief in my ability to pursue my goals. Especially I am grateful for her support and caring during an episode of my life that I went through health crisis and surgery during last years of my PhD program. She has been there in my weak moments to give me strength and courage to comeback and continue fighting. The discussions we had on the algorithms and mathematical formulation and her insights and feedbacks improved the quality of this dissertation, especially parts of Chapter 3.

I use this opportunity to also acknowledge the comments and recommendations made by my committee members, Dr. Chung-Kuan Cheng, Dr. Bill Lin, Dr. Truong Nguyen, and

Michael B. Taylor. Also the feedback and suggestions by anonymous reviewers of conferences and journals have helped to improve the quality of this dissertation.

I am very grateful for the financial aid provided to me by the National Science Foundation. Also the research environment and great colleagues at MESDAT Lab helped to improve the quality of research and life during these past few years. I extend my gratitude to Dr. Shaoxuan Wang who has been my senior colleague at MESDAT Lab and have always been there for me whenever I had questions or needed advise even after he left our lab.

Finally the emotional support and encouragement that I received from my parents and family members as well as my great friends have helped me to succeed in this path. My parents have always encouraged me to continue my education and taught me to dream big and reach for the stars. Completing this dissertation is like grabbing one of the stars. I am also thankful to my dearest friend Dr. Mohammad Abouali for his continued support throughout my PhD program. Finally, I appreciate my friends and colleagues at Broadcom who helped me to continue to balance my work and research and always supported me to continue this path.

Chapter 3 is based on the material that is co-authored with Dr. Sujit Dey and Dr. Anand Raghunathan and is accepted for publication in ACM Transactions on Design Automation of Electronic Systems (TODAES) for publication. Parts of this chapter are also appeared on the proceeding of 2012 International Green Computing Conference (IGCC'12) titled "Adaptation of video encoding to address dynamic thermal management effects".

Chapter 4 is based on the material that is co-authored with Dr Sujit Dey and Dr Anand Raghunathan and is accepted for publication by IEEE Transactions on Very Large Scale Integration (TVLSI) titled "Joint Work and Voltage/Frequency Scaling for Quality-Optimized Dynamic Thermal Management".

VITA

2001 – 2006	B.S., Electrical Engineering, Sharif University of Technology, Tehran, Iran
2002 – 2003	Teaching Assistant, Computer Engineering Department, Sharif University of Technology
2003 – 2006	Teaching Assistant, Electrical Engineering Department, Sharif University of Technology
2006 – 2008	M.S. Electrical Engineering (in Digital Electronics), Sharif University of Technology, Tehran, Iran
2008 – 2009	Electrical Engineering and Computer Science Department Scholarship for New PhD Student, University of California, Irvine
2009	Summer Intern, MindSpeed Corp., Newport Beach, California
2009 – 2012	Research Assistant, Dept. of Electrical and Computer Engineering University of California, San Diego
2010	Summer Intern, Qualcomm Inc., San Diego, California
2011	Summer Intern, Broadcom Corp., San Diego, California
2012 – 2014	Staff Engineer – Systems Designer, Broadcom Corp., San Diego, California
2015	Ph.D., Electrical Engineering (Computer Engineering), University of California, San Diego
2015	Adjunct Faculty, National University, San Diego, California
2015	Adjunct Faculty, Polytechnic University of Puerto Rico, San Juan, Puerto Rico

PUBLICATIONS

Mirtar, Ali, Sujit Dey, and Anand Raghunathan. " An Application Adaptation Approach to Mitigate Impact of Dynamic Thermal Management on Video Encoding." *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, Accepted.

Zarei, Sara, Ali Mirtar, Forest Rohwer, and Peter Salamon. "Stochastic Tracking of Infection in a CF Lung." *PloS one* 9, no. 10 (2014): e111245.

Mirtar, Ali, Sujit Dey, and Anand Raghunathan. "Joint Work and Voltage/Frequency Scaling for Quality-Optimized Dynamic Thermal Management." *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on,* Early access 2014.

Zarei, Sara, Ali Mirtar, Bjarne Andresen, and Peter Salamon. "Modeling the airflow in a lung with cystic fibrosis." *Journal of Non-Equilibrium Thermodynamics* 38, no. 2 (2013): 119-140.

Zarei, Sara, Ali Mirtar, Forest Rohwer, Douglas J. Conrad, Rebecca J. Theilmann, and Peter Salamon. "Mucus distribution model in a lung with cystic fibrosis." *Computational and mathematical methods in medicine* 2012.

Zarei, Sara, Ali Mirtar, Mohammad Abouali, Peter Salamon. “Spirometric Trends in Patients with Cystic Fibrosis.” *BIOCOMP’12*, Las Vegas, July 2012,pp. 650-653.

Mirtar, Ali, Sujit Dey, and Anand Raghunathan. "Adaptation of video encoding to address dynamic thermal management effects." In *Green Computing Conference (IGCC), 2012 International*, pp. 1-10. IEEE, 2012.

Mirtar, Ali. "Power Reduction in NoCs Considering Data Contents." *Master Thesis, Sharif University of Technology, 2008.*

Mirtar, Ali. "Examination and Commission of Nios II Softcore Microprocessor and Its Peripherals." *Bachelor Thesis, Sharif University of Technology, 2006.*

ABSTRACT OF THE DISSERTATION

Adaptation Techniques to Mitigate Impact of Process Variations and Dynamic Thermal Management

by

Ali Mirtar

Doctor of Philosophy in Electrical Engineering (Computer Engineering)

University of California, San Diego, 2015

Professor Sujit Dey, Chair

The goal of this dissertation is to introduce application adaptation techniques to maximize the quality of complex applications when certain hardware flaws exist in a platform. Two types of hardware flaws have been studied in this research: static hardware flaw, which is due to process variations, and dynamic hardware flaw, which is due to dynamic thermal management.

The adaptation idea is based on the premise that complex and real-time applications provide a set of parameters that can be used to tune their complexity and quality. In this dissertation, we use this application layer opportunity to adapt applications to the effects of hardware drawbacks by tuning their parameters to achieve maximum quality. We then extend

our solution and show how our proposed application layer solution can be combined with conventional hardware solutions to provide a hybrid adaptation method to tackle hardware drawbacks.

The experimental results on practical platforms such as MacBook Air show that using our application adaptation can significantly improve the quality of real-time applications, such as video encoder and turbo decoder, while not requiring any modification to the hardware. In addition, the results show these adaptation methods can be used to reduce the cooling efforts such as processor fan speed.

Chapter 1

Introduction

Digital devices have become part of our daily lives. Some devices such as video recorders, communication devices, or gaming consoles are designed for a specific function and purpose but some of them are generic devices that can do multiple jobs such as personal computers and laptops. For all of these devices, the design of a product starts with the functionality it should perform, such as recording a video; and then the decision is made for the choice and design of hardware and software based on that functionality [WHB06]. In this process, system designers think of what is expected from the hardware to achieve its objectives and what is expected from software to perform the functionality on the platform. This analysis leads to the design of hardware specifications in case of application specific digital devices. On the other hand, a general-purpose hardware is evaluated by a set of specifications as well and the user should check if the hardware specifications satisfy the requirements of an application that is going to be executed on that platform.

Usually, there is a division between hardware and software development meaning the software designers assume that the hardware would guarantee its specifications during the run time of the application. To support this requirement, hardware designers should consider a large margin in their design to make sure all of the specifications hold at all times, which will result in additional cost and overdesign [PGS12]; otherwise, if it fails to guarantee its specifications from time to time [SSS04] the quality of the application might suffer significantly.

In this research a different philosophy is chosen for the design of an application. Instead of keeping hardware and software separate, it is considered that hardware may have deviated from its nominal specifications and an application layer solution is proposed to deal with this problem. In the proposed solution, part of application monitors hardware conditions and adapt its complexity in a way that it can produce maximal quality results even when the hardware can not guarantee its performance. It is very important to distinguish the difference between hardware performance and application quality in the above-mentioned solution. The discussion about this difference is provided after a brief introduction about causes of hardware performance deviations from specifications.

1.1 Performance Deviation from Specification

Hardware design goes through many steps such as functional specification, RTL design, place and route, and physical design [WHB06]. Eventually, when the design is completed it is sent to manufacturing to be built on silicon. The manufacturing process is the main source of static deviation from specification and this phenomenon is called process variation [BKN03]. It means after manufacturing, there would be some variations among produced chips performance and power consumption. It is important to recognize that the performance deviation from specifications that is caused by process variations is a static issue. Once a chip is produced, it may have for example 10% frequency below the specified nominal frequency, but this value would not change due to process variation any more and remains the same for the life of hardware.

The first response to variation-affected chips (low performance chips) was discarding them or selling them at lower prices. But as process technology became deep sub-micron, the number of discarded chips became larger, which eventually increased the cost and diminished

other benefits of using sub-micron technology. Therefore, hardware solutions started to emerge to prevent the effects of process variations such as adaptive voltage scaling [ES07] and adaptive body biasing [GM08]. While these techniques help to decrease the number of discarded chips, they lead to system overdesign by adding certain guard bands and result in area and complexity increase of chips. More details about process variations and related methods are provided in Chapter 2.

In addition to the static performance deviation from specifications, there can be dynamic performance deviation from specification due to thermal issues as well. The cooling methods, such as heat sinks and fans [XYP13; ZXL12], are designed to deal with silicon thermal issues. But these solutions are not enough and the silicon temperature can rise beyond hundred degrees Celsius very often. To address the thermal issues, multiple dynamic thermal management methods have been developed [JP07; PGP10; SCC10]. One common behavior between all of these methods is the fact that they eventually throttle some of the system resources to reduce the amount of power consumption and thereby reducing platform temperature. More details about DTM methods and this phenomenon are provided in Chapter 3 and Chapter 4.

Since DTM methods throttle system resources to reduce platform temperature, they reduce platform performance as well and prevent the hardware from achieving its nominal performance defined in its specification. It is very important to recognize that these throttling events happen during run time of an application and their duration and behavior can not easily be predicted. Therefore, a real time solution is required to address dynamic performance deviations from specifications caused by DTM.

As mentioned above the problem is the deviation of hardware performance from its specification that causes application quality drop. There are two terms in this problem

statement that are closely related and may cause confusion: hardware performance and application quality. A discussion about the difference between these two factors is provided next.

1.2 Application Quality vs. Hardware Performance

The operational frequency of hardware is one of the simplest ways to quantify its performance and it is usually one of the key parameters in defining hardware specifications. However, hardware frequency can not fully describe the performance of a hardware. For example, a dual core processor is expected to have twice the performance of a single core processor when they have the same operational frequency.

Therefore, other metrics have been introduced to compare processor performance such as total number of instructions a processor can execute per second (million instructions per second – MIPS) [MN04]. As different architectures become popular, it became evident that the measure of MIPS would not be able to compare the performance of two processors with different architectures. Therefore, other metrics are introduced based on the run time of an application such as Dhrystone benchmark [RW84] or based on the run time of a set of applications such as SPEC2006 benchmark [PZH09]. This way, by comparing the runtime of the same application on platforms with different architectures one could compare their performance.

As we mentioned before, process variations and DTM may cause the platform to deviate from its specification and drop it performance. This means the total number of instructions it can execute per second would reduce, or the runtime of applications may increase when executing on the affected platform. But when it comes to real time applications these effects are not that simple any more. Real time applications are made up smaller

building blocks which have deadlines that should be met in order to work properly. If the runtimes of the building blocks get longer to the extent that they miss their deadlines, the behavior of the application might be affected. In the case of hard real time applications such as medical devices or automotive safety systems, failure to achieve the application deadline can be catastrophic and absolutely unacceptable. In the case of soft real time applications such as video play back, video recording, gaming, or communication applications, failure to meet the deadline would affect the quality of the application. This is where the quality of an application becomes important and would be different from runtime of its building blocks.

The quality of an application is the end result that we expect from an application. For example, the quality of video encoding application is the quality of recorded video. For a gaming application, the quality can be defined as responsiveness of the game or visual details of the gaming graphics.

Traditionally, the notion of desired application quality gets translated into timing requirements of its building blocks and then translates into hardware specification requirements. Then the job of software developer and hardware designer gets separated. Hardware designer tries to achieve those required specifications and application designer assumes hardware specification is guaranteed and designs his/her own software. Subsequently if the hardware performance deviates from its specification, the effect on application quality is not predictable in general and in many cases the application quality may drops significantly.

This research changes this design paradigm and rather than separating application quality from hardware specification, we focus on application quality and propose an application layer solution to maximize the application quality while hardware performance varies, either statically after manufacturing, or dynamically during the operation. This paradigm is possible due to the design of new complex real time applications that provide a set

of parameters to the user to tune their complexity and quality. By use of these parameters, we are able to not only improve the quality of applications but also reduce margins and guard bands relative to actual required specifications that results in area reduction and hardware design simplification.

This research is not the first attempt to use application level solutions for hardware-induced problems such as hardware unreliability or hardware performance variations. A short summary of related works in this field is provided next and more detailed comparisons are provided in Chapter 2, Chapter 3, and Chapter 4.

1.3 Application Level Approaches

Many researches have worked on ideas to address hardware-induced problems through software solutions. Operating system (OS) and scheduling based solutions consist a big category of these researches. For example, the task schedulers in [TT08] and [MSG10] are designed for a multi processor system and they consider effects of process variations and optimize these platforms for power. The OS and scheduling based solutions does not limit to only static problems like process variations. For example [HCQ13], [WB08], and [RV07] propose different schedulers to address thermal managements with different boundary conditions and objectives. In this research application adaptation approach is used which is a different method from scheduling based solutions. Scheduling based solutions would work in a multi-tasking or multi application situation, which is not the topic of this research.

In [DMK10], researchers not only consider that the hardware may deviate from its specification; they go one step further and consider stochastic processors. In this design, the hardware would notify software by sending an error message if it is not able to perform its task as needed. With this additional information sent from hardware, software can make

decisions to work properly. This assumption and design have some benefits that are beyond the scope of this research. Our proposed solution has a fundamental difference with this research. Rather than changing hardware and asking it to report errors, software monitors the hardware conditions and makes decision accordingly. This approach makes our proposed solution suitable for existing hardware and avoids redesigning the hardware.

In addition to the above approaches that are general and can be adapted to be used with different applications, there are many other studies that look at specific applications and use their characteristics and the opportunities available within an application and use it against hardware-induced problems. The researchers in [WC06] and [IB07; IB10] focus on resource-constrained platforms and target video encoding application to perform well. The work in [PGS12] also uses application adaptation to improve the yield of process variation affected video encoders. In addition to these static solutions for constrained hardware, there are many research targeting thermal issues while focusing on specific characteristics of the applications. [GQ11], [YLK07], and [YK08] target different multimedia applications and use different methods such as machine learning, stochastic analysis or hybrid methods to propose different solutions to deal with thermal issues. The proposed solutions in this research are generic and do not use any specific characteristics of an application. Therefore, they can be applied on a broad range of applications such as video encoding and communication protocols.

A more comprehensive list of related research is provided in chapter 2, chapter 3, and chapter 4. The proposed solutions in each of these chapters have been compared with the related work separately.

1.4 Contributions and Overview

As it was mentioned earlier, many complex and real time applications provide a set of parameters that can be used to tune their complexity and their quality. Real time video encoding is the main application that has been studied in this research. We have also shown some of the proposed solutions on turbo decoding application, which is widely used in communication systems. The details about application parameters and application quality are discussed in the next chapters. Furthermore, three adaptation techniques are proposed in this research that are briefly introduced below.

1.4.1 Static Application Adaptation

The use of application adaptation to reduce the effects of process variations on application quality is discussed in Chapter 2. Process variation is a static problem by nature, meaning its effect may vary from chip to chip but its effects remain the same for the lifetime of the chip. A video encoder based on H.264 standard is discussed in this chapter and it is shown that by using application adaptation, the quality of affected chips would increase which results in yield increase and thereby reduction in cost per chip.

1.4.2 Dynamic Application Adaptation

Thermal issues and dynamic thermal management are the root causes of dynamic performance deviation from specifications. In chapter 3, a new method is proposed that uses application adaptation to deal with these effects for a video encoder application. The proposed solution in chapter 3 works in real time along with the video encoding application while the solution provided in chapter 2 is a static solution that is executed once before delivering the product. In addition, the role of cooling efforts on application quality also has been discussed

in chapter 3 and the results show the adaptation algorithm can help to reduce the cost of cooling as well.

1.4.3 Hybrid Application Adaptation

One of the interesting results from the study in chapter 3 is that application adaptation not only helps with the quality of video encoding application but it also helps to reduce platform temperature as well. This observation became the motivation to look at application adaptation as a new method to perform dynamic thermal management. Chapter 4 starts by introducing a new DTM method based on application adaptation called Dynamic Work Scaling (DWS). The results show that the effect of this DTM on application quality can vary from one application to another. For example it is shown that the quality of turbo decoder application is worse when using DWS compared to hardware based DTM.

Based on this observation, a new hybrid method is provided that uses both DWS and hardware based DTM to introduce a new hybrid adaptation based DTM. It is shown that this method will produce higher application quality results compared to either of the DTMs working separately.

Chapter 2

Application Adaptation for Variation-Tolerant Video Encoding

2.1 Introduction

The scaling of integrated circuits (ICs) into the nanometer regime has thrown up new challenges for designers, foremost among which are variations in the characteristics of IC components. Variations threaten to diminish the fundamental benefits of technology scaling, such as improvements in cost-per-transistor, performance and power consumption. Variation-aware design techniques that have been proposed thus far at the mask, circuit, and logic levels, are being stretched to their limits, and cannot contain the incessant increase in variations. Therefore, it is important to develop new approaches to design systems that are inherently resilient to variations in the underlying components.

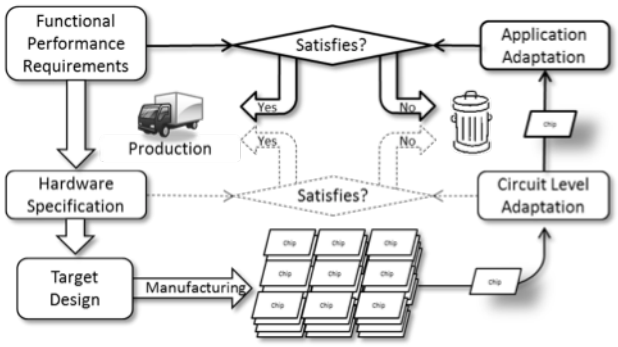


Figure 2.1 Design process: from functional performance requirements to product delivery. Dashed-line sections show the common selection method for variation affected hardware. Solid-line sections show our proposed method based on application adaptation for variation affected hardware.

Application adaptation is a fundamentally different approach for making systems variation tolerant compared to traditional hardware-based approaches (Figure 2.1). Process variations lead to significant unpredictability in the speed of transistors and gates, causing the operational frequencies of a set of manufactured instances to follow a statistical distribution [BKN03]. Hardware-based approaches focus on making chips variation resistant by costly over design or variability-aware methods (such as adaptive voltage scaling [ES07] and adaptive body biasing [GM08]) and reject chips with variation from nominal hardware specification (clock frequency). Application adaptation, instead of relying only on hardware performance, accepts chips with hardware variability and adapts the application accordingly to satisfy the functional performance requirements. Consequently, many chips that could be rejected due to hardware variability, would be accepted based on their functional performance competency even with less hardware over-design, and thereby, reduced cost per chip.

Recent market research shows video consumption will dominate both Internet and mobile traffic, and hence video applications, including video encoding/decoding, will dominate the workload of servers, personal computers and mobile platforms. Thereby, in this

chapter, we focus on real-time video encoding; an application which is not only compute intensive, but also its real time requirement makes it the most vulnerable in terms of variation effects. For real-time video encoding, we define functional performance based on the metrics used to evaluate encoding, namely visual quality and bit-rate. By designing variation aware application, we increase the tolerance toward frequency variation of chip instances (leading to yield improvements), and avoid overdesign (reducing the cost of dealing with variations). We demonstrate that, in the presence of variations, application adaptation for video encoding can significantly increase the number of chip instances with tolerable functional performance.

We observe two key properties of video encoding that enable our approach. First, there is a correlation between the amounts of computation needed, the bit-rate and the quality of the encoded video stream. In other words, there exists a multi-dimensional tradeoff between these metrics. Secondly, there is an inherent tolerance towards a nominal deviation from the specification of both the desired bit-rate as well as the quality of the encoded video. For example, the resulting quality of encoded video is often subjective, with some visual artifacts less perceivable than others. In other scenarios, slight increases in bit-rate may be acceptable. The above two properties allow us to design a video encoding system such that it would be able to adapt to frequency degradation in hardware without causing any perceivable degradation in user experience (functional performance).

In this chapter, we develop a systematic approach to application adaptation and apply it to the design of a variation-tolerant real-time video encoder using the H.264 standard [WSB03]. To achieve this goal, we first study the parameters used in the application, and characterize their effects on the different metrics of the application, namely computation time, video quality and bit rate. Subsequently, we develop an application adaptation algorithm, which uses the identified parameters and their computation time-quality-bitrate

characterizations to adapt the encoding application to improve tolerance towards variation induced frequency degraded hardware. Using software implementation of the H264 encoder application on an ARM Cortex A8 processor and Intel Core i7, we demonstrate that while frequency degradation of the processor can lead to significant degradation in the encoder performance, the proposed adaptation approach can provide tolerance of up to 30% frequency degradation, without any noticeable loss in perceptual quality, and with nominal and acceptable overhead in bit rate.

The rest of this chapter is organized as follows. In Section 2.2, we briefly introduce related works and highlight the differences and significance of our approach to others. In Section 2.3, we formulate the problem of application adaptation to tolerate hardware variation. In Section 2.4, we identify and characterize the parameters for adaptation of H.264 video encoder. Section 2.5 describes the adaptation algorithm. In Section 2.6, we introduce the experimental framework, and present results of applying the proposed approach on an embedded and server platform. Finally we present our research's conclusions in Section 2.7.

2.2 Related Work

We discuss related work along two directions – techniques for variation tolerant system design, and techniques that optimize video encoding for resource-constrained or real-time systems. There has been a significant body of work that has addressed variations at various levels of design abstraction. We focus on techniques at the architecture and system levels in Section 2.2.1. Similarly, there is a large body of literature on optimizing video encoding algorithms and implementations. We describe techniques that have been proposed for H.264 encoding in resource-constrained or real-time settings in Section 2.2.2. We also place our contributions in context of these bodies of related work.

2.2.1 Variation Tolerant Design

Efforts toward variation-tolerant design at the architecture and system levels have increased in recent years. At the micro-architecture level, the impact of variations on the performance of microprocessors has been extensively analyzed [UTB06; BAS09]. Various architectural design techniques to mitigate the impact of variations have also been investigated [TST07; LB06]. At the system level, performance and power analysis techniques that consider variations have been proposed [CLR06; MG06; GM13]. In addition, various system-level design decisions, including HW/SW partitioning, scheduling, multi-island design and power management have been adapted to address variations [CLR10; OMM08; WY08; WNW07]. Most of these techniques require hardware modifications and need to be incorporated into the design process. The proposed approach of application adaptation is complementary to these techniques, and can be employed in conjunction with them when they are unable to fully hide the impact of variations.

A few efforts have been made to address process variations in the software layer. System-level task scheduling to mitigate the impact of variations has been proposed [TT08; MSG10]. For example, in a multi-core system, more computationally demanding tasks may be scheduled on faster processor cores to minimize the overall performance impact. These techniques are limited in their scope, since they consider the application workload to be fixed and exploit whatever flexibility is available in scheduling. In contrast, our approach actively modifies the application workload to better suit the capabilities of the variation-impacted hardware platform.

The concept of application adaptation in the context of variation tolerance was first proposed in [PGS12], which also focused on hardware-signatures to be used in the binning process of variation affected hardware. They designed an algorithm to find the boundaries of

each bin that maximizes the average of all the chip instances quality. However, when it comes to selecting the optimum parameters for each chip instance, they rely on exhaustive search to generate a two dimensional tradeoff curve (between visual quality and run time). With the increase of parameter space and additional metrics (such as bit rate), exhaustive search is neither efficient nor practical. Our contribution is the formulation of application adaptation as a multi-dimensional optimization problem, a systematic methodology and adaptation algorithm to realize a variation-aware adaptive video encoder.

Significance-driven computation [MKR09] also targets quality-sensitive applications by recognizing that all the computations in an algorithm do not have the same effect on output quality. In the presence of variations, selective multi-cycle operations are used to ensure correct execution of significant computations without degradation in clock frequency. This technique is especially suitable for application-specific HW implementations, while our approach is targeting application level which is useful even when the hardware modification is not an option (such as software encoder on a processor).

Stochastic processors, which allow variations and imperfections in hardware to become visible to software in the form of errors, have been proposed in [DMK10]. They also suggested the design of stochastic applications, which have been modified so that they are robust to errors in the results of computations performed by the underlying hardware. In this process, the computational complexity of the application is often increased. In contrast, our approach assumes correct execution of computations on the underlying platform, and reduces the computational complexity of the workload to match the degraded performance of the hardware in the presence of variations.

2.2.2 Real Time and Optimized H.264

There have been several efforts in the field of adaptive video encoding that do not explicitly target variations. While this field is too vast to describe exhaustively, efforts therein can be divided into three categories, namely complexity reduction techniques, optimized encoder implementations on specific platforms, and adaptive real-time encoding. We present representative techniques in each of these categories, and restrict ourselves to the context of H.264 encoding.

Several techniques have been proposed improved encoding algorithms that decrease the computational complexity of the encoding process [SMH10; KRB06; KK05; KCC06; LCL05], thereby reducing the total required time for encoding. Such approaches are mostly implementation independent and make the encoder as fast as possible but do not guarantee real-time encoding, which is the main objective of the proposed work. Moreover, our approach can be applied to the optimized algorithms that result from any of these techniques. In this paper, we have used x264 encoder which is a very efficient implementation of H.264 that is 50 times faster than Joint Video Team Reference Model [JMR10] of H.264 [MV06].

The second category of techniques deals with optimized implementation of H.264 encoders to utilize the specific capabilities of a hardware platform, including multiple processing cores, special instructions, etc. [RPC06][WC06]. These approaches do not consider the effect of variations. In other words, while such optimizations may enable the realization of a real-time encoder on an ideal hardware platform, process variations may result in an inability to meet real-time constraints, which is the focus of our work.

The third category of techniques utilizes adaptive video encoding to meet real-time constraints on resource-constrained platforms [IB07; IB10; SBH10]. The main idea is to dynamically regulate computational complexity by using faster but less optimal versions of

various encoding steps including motion estimation. These techniques attempt to minimize the resulting degradation in video quality. The main difference between our work and these techniques is that we recognize and exploit the multi-dimensional nature of the tradeoff between encoding time, video quality, and bit rate. Therefore, our adaptation methodology can not only maintain real-time behavior of the encoder, but also work under specified quality and bit rate constraints. In addition, the nature of adaptation used in our work is much more fine-grained compared to the above techniques, since variations result in a population of manufactured chip instances with a fairly continuous range of operating frequencies.

Finally our approach does not require any internal modifications to either the hardware or the video encoding algorithms. We use the parameters that are exposed by video encoders to adapt them and cope with variations. Therefore, it is broadly applicable and may be combined with changes to the algorithm, such as new techniques for motion estimation.

2.3 Problem Formulation

Process variations lead to significant variability in the speed and leakage power consumption of transistors and gates. Since the speed of a processor is dictated by its slowest paths, variations will cause the operational frequencies of a set of manufactured instances to follow a statistical distribution [BKN03]. Considering this well-known fact, we abstract the effects of process variations on applications as follows. A design with process variations can be abstracted as a processor, or in general an instance of a hardware design, whose operating frequency is different from the nominal or target frequency (i.e., we have different chip instances with different operating frequencies). In our work, we consider hardware platforms whose operating frequency is different from each other in a certain range. Therefore the computational capacity of the hardware varies and depends on the chip instance. From an

application perspective, the run time, and therefore its functional performance/results can be different for each chip instance.

In a video encoding application, the encoder receives raw video sequence and compresses the data by utilizing redundant information in the video sequence and the human's limited visual perceptual. The compression can help to significantly reduce the bit rate of the encoded video, helping to reduce storage and/or network bandwidth needed to transmit the video. The functional performance of a video encoder can be measured by three metrics. The first metric, video quality, is measured by peak signal to noise ratio (PSNR)[HG08]. Almost all common video encoding applications are lossy; meaning the part of information which human's eyes are not sensitive to is omitted from original data. Moreover, throughout the entire process of encoding we change the information content of video to some extent; therefore we need an end-to-end measure of video quality through the video encoding process. For an encoded video sequence v we represent its video quality as $VQ(v)$. The second metric of each encoded video sequence is the required bit rate for sending it through the network measured by kilobits per second (kbps). For each encoded video sequence v, $BR(v)$ is the required bit rate to send it through the network. Finally for a given encoder, the time needed to encode a raw video sequence is denoted as $RT(v)$. The above three metrics of video encoding, the encoded video quality, encoded bit rate, and encoding run time along with different parameters of encoder (parameters are discussed in 1.4.1) form a multidimensional space, whose tradeoff will be exploited later to achieve variation tolerance.

Assuming that the encoder is built on a hardware platform with no process variation, we call the resulting video sequence of such an encoder as v_o. In the presence of process variations, since the hardware frequency may degrade, the encoder may not be able to process frames as fast as it should. In a real-time encoder, this will result in "dropping" some of the

input frames that it could not process in the given time. Consequently, an encoder may produce a lower quality video output. The video encoded in a variation affected hardware is called v_d. Finally, we propose a multidimensional adaptive approach to video encoding in the presence of process variations. This adaptive encoder uses different parameters than the original encoder to decrease its computational requirement and run in real-time. The decrease in encoding complexity may come at the expense of reduction in video quality, and increase in the bit rate. We denote the video sequence generated by an adaptive encoder as v_a.

Our goal for real-time variation tolerant H264 video encoding application adaptation is to achieve V_a with the following criteria:

$$\begin{aligned} VQ(v_d) \leq \boldsymbol{VQ}(\boldsymbol{v_a}) \leq VQ(v_o), \\ \boldsymbol{VQ}(\boldsymbol{v_a}) \geq VQ(v_o) - \epsilon, \\ \boldsymbol{BR}(\boldsymbol{v_a}) \leq BR(v_o) + \delta, \end{aligned} \tag{2.1}$$

Where ϵ is an acceptable degradation in video quality such that there is no perceptual difference, and δ is an accepptable increase in bit rate such that there is no unsatisfying increase in cost of video transmission. This leads to the following constrained maximization problem - to find parameter for the encoder to:

$$\text{Maximize: } VQ(v_a), \qquad \text{Subject to: } \begin{cases} RT(v_a) - RT(v_o) \leq 0 \\ VQ(v_o) - VQ(v_a) \leq \epsilon \\ BR(v_a) - BR(v_0) \leq \delta \end{cases} \tag{2.2}$$

As described in the above equations, we are looking for an adaptation algorithm that, even in the presence of process variations in the underlying hardware, still helps the video

encoder to generate video sequences whose quality is as close as possible to an encoder running on hardware with no process variations. To achieve this goal, we use the inherent tolerance in video appilications in terms of ϵ and δ. Selection of ϵ and δ has an important role on the significance of the solution. Too big value for ϵ and δ removes any benefit from solving the optimization problem meaning that, in the extreme, we can encode with very poor quality or can simply not compress the input sequence. On the other hand,very tight values for ϵ and δ would eliminate the opportunity for tolerance toward variations. In theory, having constraints on all three encoder metrics may result in no feasible solutions. If the optimization problem returns no solution for a hardware instance, it means that the instance does not meet the required functional performance and should be discarded. Note that, this adaptation method is independent of the platform (Software/FPGA/ASIC) and implementation method of encoder.

To come up with an adaptation algorithm, it is necessary to first identify parameters that significantly impact the three metrics of encoder, namely encoder computation time, output video quality, and bit rate. It is also necessary to analyze and model the impact of these parameters on the three metrics so that these models may be used for adaptation in the presence of variations.

2.4 Identification and Characterization of Application Adaptation Parameters

As mentioned before, our approach to application adaptation will be to adapt a set of application parameters such that the computation complexity of the application can be adjusted to match the variation induced frequency degradation of the hardware, while maintaining the required quality of the application results. In this section, we first identify a

set of parameters of the H.264 encoding application which may be used for application adaptation. Next, we characterize the effects of the selected parameters on the H.264 encoder computation time (hence computation complexity), as well as its functional performance in terms of video quality and bit rate.

2.4.1 Identification of Adaptive Video Encoding Parameters

In H.264 codec, as in any other image/audio/video compression application, numerous parameters and tools can be used with varying effects on computation complexity, and the quality of the resulting image/audio/video. Since ability to reduce computation complexity in response to frequency degradation of the underlying hardware will be the main objective of our application adaptation approach, we investigated the H.264 encoding parameters available, and selected three that will have the most significant impact on computation complexity, while making varying compromises on the encoding quality (video quality and bit rate). Although we focused on the most effective parameters of H.264, this approach is not limited to 3 parameters and other parameters can be applied with the same method if required. Next we briefly explain each of the selected parameters.

2.4.1.1 Group of Picture Size

In H.264, there are 3 different frame types. An I-frame is encoded using only information in the same frame, and is least expensive to compute. However, an I-frame is also most expensive in terms of bits needed to store. In contrast, a P-frame is encoded using the information in the previous frame(s). Hence, when consecutive frames with similarities are encoded as P-frames, the resulting frames have significantly less bits. However, computing P-frames can be significantly more compute intensive than computing I-frames. Finally, the third frame type is B-frame. B-frames are generated using information in frames before and after them. Hence, they can be encoded with fewer bits than I-frames and P-frames, but takes

more computation time compared to them. Group of Picture (GoP) size is the total number of frames starting from an I-frame and a sequence of P-frames and/or B-frames until the next I-frame. For instance, GoP size of 1 means all frames are I-frames. Computation time for GoP=1 will be low, but the resulting video will have significantly higher bit rate, and thereby significantly more expensive to transmit in terms of network bit rate needed. On the other hand, using a GOP size of 96 will lead to very efficient video encoding in terms of bit rate, but will require significantly more computation time.

2.4.1.2 Number of Reference Frames

As mentioned above, P-frames are encoded using information from the previous frames. Number of reference frames is the number of frames that are used in the process of encoding a P-frame. Using lower number of reference frames to encode a P-frame will lower the computation time needed, but may increase the bit rate needed to encode the P-frame.

2.4.1.3 Quantization Level

Quantization level is an integer constant that is used by the encoder to leave out details of the video that are unlikely to be perceived by human eyes, thereby reducing the amount of encoding that needs to be done, and thereby the computation time needed. Using higher quantization level also reduces significantly the bit rate and the file size, thereby reducing the application time needed for memory access as well. However, this comes at the expense of losing video quality. As we will see in the next section, this is the most sensitive parameter toward video quality.

Note that quantization level is also used by the encoder to perform rate control when asked to perform Constant Bit Rate (CBR) encoding. Hence, if we use quantization level in our adaptation algorithm, the encoder can be used to perform Variable Bit Rate (VBR)

encoding only, which is also a common encoding for real-time use case. To enable CBR encoding, we may use other encoding parameters for adaptation, besides quantization level.

2.4.2 Characterization of Video Encoding Parameters

Having described the video encoding parameters we use in adaptation, in this section, we study and characterize them in terms of their effects on computation time, video quality and bit rate. In the following discussion, we use a 3-tuple (g, n, q) to denote an encoding setting, where g referes to GoP size, n referes to number of reference frames, and q refers to quantization level. We conduct the characterization experiments using a few sample values from a possible range of values allowed for each of the parameters. For instance, according to the H.264 standard, quantization level can be any integer number between 0 to 51, so we use 5 numbers 18, 23, 28, 33, and 38 as our samples (18 to 33 is the practical range to use for quantization). Table 2.1 shows the sample values for each parameter used for characterization, producing a multi-dimensional sample space for parameters.

Table 2.1 Parameters and their values used for characterization

Parameter	Values
Group of pictures	1, 6, 12, 24, 48, 96
Number of reference frames	1, 2, 3, 4, 5
Quantization level	18, 23, 28, 33, 38

Next, we use x264 encoder (one of the most efficient H.264 encoder implementations) to encode representative video sequences using all parameter combinations from Table 2.1, and record the encode computation time, resulting video quality, and video bit rate for each

parameter sample. The experiments are conducted on a 600 MHz ARM Cortex-A8 processor present in a Beagle Board [BGL15].

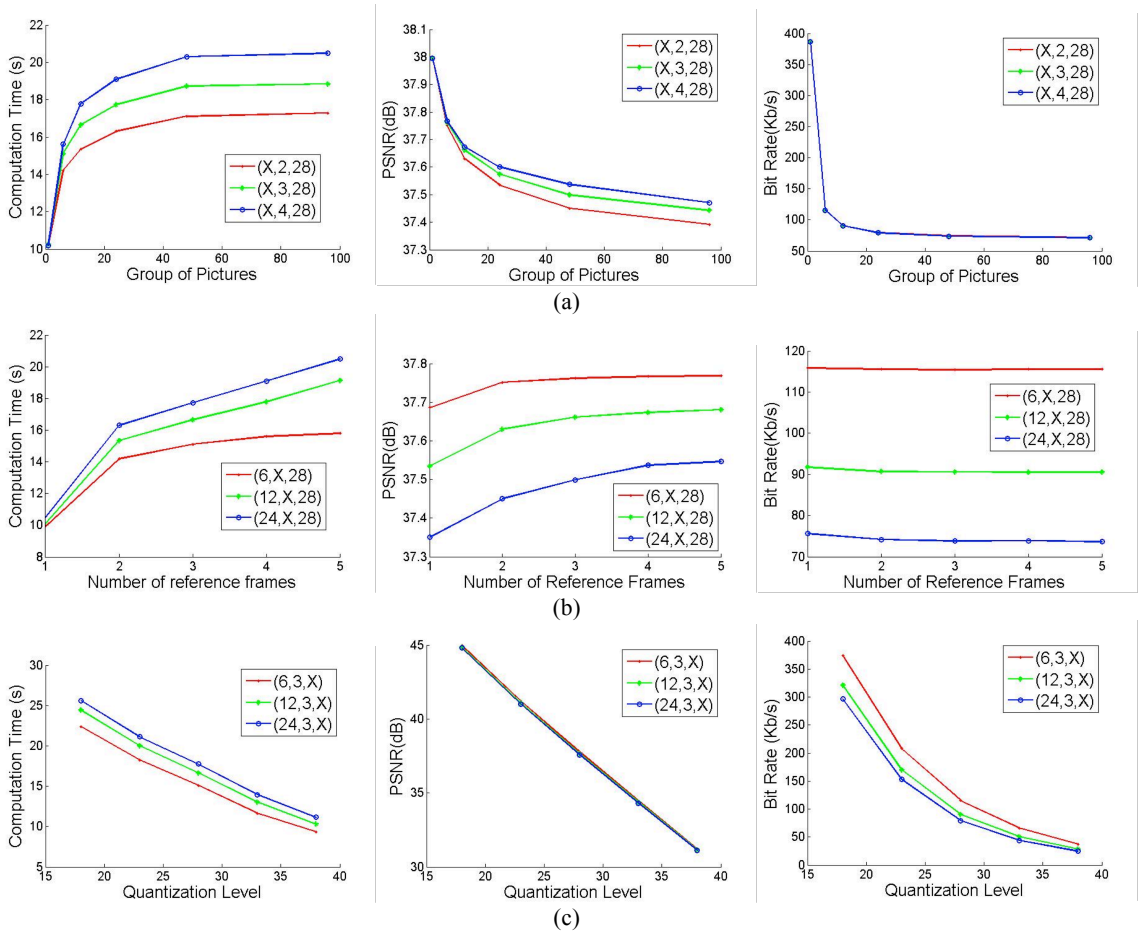


Figure 2.2 Effects of varying three selected parameters (a) GoP size, g, (b) number of reference frames, n, and (c) quantization level, q, on Run Time, Video Quality (PSNR), and Bit Rate of encoded video.

Figure 2.2 shows some representative data points from the characterization experiments, highlighting the effects of each of the parameters, (a) GoP size, (b) number of reference frames, and (c) quantization level, resepctively. While the results shown are for a

popular benchmark video, Foreman video sequence with 300 frames, we have observed similar effects displayed for other representative videos that we experimented with. Each graph represents an encoding setting where one of the encoding parameters is varied (marked by "X" in the associated setting) while the rest are kept fixed. For example, Figure 2.2(a) shows three graphs, each representing results obtained (computation time, video quality, and video bit rate) when GoP size, g, is varied according to Table 2.1 Parameters and their values used for characterization, keeping the values of the other two parameters fixed as shown. We draw the following observations from Figure 2.2:

1) Goup of Pictures: While the effect of GoP on video quality is nominal (< 1 dB in the range considered), it has high impact on computation time (up to 2X), and the resulting video bit rate (up to 4X). Specifically, GoP size can be reduced in the range of 48 to 1 to significantly reduce encoding computaion, but at the expense of also significantly increasing bit rate. Hence, it has to be used judiciously by the application adaptation algorithm, described in the next section.

2) Reference frames: From the results shown in Figure 2.2(b), we observe that this parameter can be utilized to significantly reduce computation time (up to 2X), while having nominal impact on both video quality (< 0.2dB), and bit rate (< 5%).

3) Quantization level: From Figure 2.2(c), this parameter can be used to significantly reduce encoding computation time (> 2X), but can severely degrade video quality (up to 1.5X), and increase bit rate (up to 6X). We also observe that while it affects computation time linearly, its effect on resulting bit rate is non-linear.

In the next section, we will develop an adaptation algorithm for the H.264 encoding application, using the selected parameters and the characterization of their effects.

2.5 Application Adaptation Algorithm

Figure 2.3 presents an overview of our application adaptation algorithm. To describe the adaptation algorithm, we first start from our input parameter space. Each set of parameters can be described as a point in a three dimensional space, in a way that each coordinate corresponds to one of the aforesaid parameters described in Section 2.4. For instance our input point P can be described as follow; $P = (x_1, x_2, x_3)$ where x_1 is GoP size, x_2 is the number of reference frames, and x_3 is quantization level. Furthermore each metric of a generated video can be demonstrated as a function of the input point:

$$\begin{aligned} VQ(P) &= f_1(x_1, x_2, x_3) \\ BR(P) &= f_2(x_1, x_2, x_3) \\ RT(P) &= f_3(x_1, x_2, x_3) \end{aligned} \qquad (2.3)$$

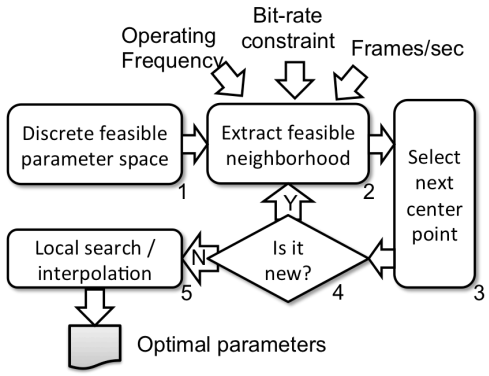


Figure 2.3 Application Adaptation Algorithm Overview

Based on the characterization described in Section 2.3, we construct functions that represent these three metrics of interest. The inputs to the adaptation algorithm are:

- A domain of possible parameters D where:

$$D = \{P|P = (x_1, x_2, x_3), x_1 \in X_1 = [1, \dots, 96] \ , x2 \in X_2 = [1, \dots, 5] \\ , x_3 \in X_3 = [18, \dots, 38]\} \tag{2.4}$$

- Three constraints C_1, C_2 and C_3. C_1 is the minimum required quality. C_2 is the maximum bit rate of the encoded stream that the encoder is allowed to produce. C_3 is the run time boundary which is extracted from the number of input frames per second. To satisfy the real-time requirement of application, the encoding rate should be equal to or faster than the number of input frames per second.
- Three metric functions: $f_1(P)$, $f_2(P)$, and $f_3(P)$

The output of the algorithm is a set of parameters that enables the encoder to work in its best condition in the presence of process variations.

The adaptation algorithm is based on two key techniques - *multi-resolution search* and *symbolic manipulation* of the metric functions. We empirically established that the computation time, bit-rate, and quality functions are monotonic functions of the adaptation parameters. Therefore, we first use a coarse-grained representation of the parameter space and use steepest ascent hill climbing [RN03] to quickly identify the sub-optimal parameter values, and then we use linear interpolation to select optimal parameters. In addition, we apply various constraints as symbolic operations on the parameter space and metrics functions, by restricting the domain of parameters to the values for which the multidimensional constraints are satisfied. We next describe the key steps of the algorithm in greater details (step numbers correspond to the labels in Figure 2.3).

STEP 2. In this step, given a point *P*, we find its feasible neighborhood. The algorithm starts with an arbitrary point and continues as shown in Figure 2.3. To define the feasible neighborhood, we first define the neighborhood of a point *P* as follows:

$$N\big(P=(x_1,x_2,x_3)\big)=\{(x_1',x_2',x_3')|x_i'\in\{b_i(x_i),x_i,a_i(x_i)\},i=1,2,3\} \quad (2.5)$$

where b_i and $a_i$are "before" and "after" operators on a parameter:

$$\begin{aligned} b_i(x)&=\max\{t|t\in X_i,t<x\}\\ a_i(x)&=\min\{t|t\in X_i,t>x\} \end{aligned} \quad (2.6)$$

The feasible neighborhood of a point P is a subset of $N(P)$that satisfies the bitrate and run time constraints of the problem:

$$N_f(P)=\{P'|P'\in N(P),f_i(P')\le C_i,i=2,3\} \quad (2.7)$$

STEP 3. In this step we find the point that maximizes our quality function in the feasible neighborhood:

$$P^*=\arg\max_{P'\in N_f(P)} f_1(P') \quad (2.8)$$

This step completes one iteration of steepest ascent hill climbing algorithm. Steepest ascent is designed to find function maximum point in a neighborhood and it continues searching until it reaches a local/global maximum. Point P refers to the last point used in previous iteration (last iteration P^*) or the arbitrary initial point.

STEP 4. The decision to end the hill climbing algorithm is made in this step. If the maximum point in the neighborhood P^* is the same as the last point P, then our initial search is completed.

STEP 5. Since the last point found in STEP 4 is based on a coarse-grained sampling of the parameter space, the selected point P is not the actual optimum solution. Hence, we perform a local search around point $P = (x_1, x_2, x_3)$ on all three metric functions by linear interpolation to derive the optimal parameter. Linear interpolation of metric function f_i for parameter x_j when other parameters are constant and only x_j varies is shown as:

$$m_{ij}x + h_{ij} \qquad (2.9)$$

Now if we solve each linear interpolation equation for the corresponding constraint we obtain nine solutions:

$$x_{ij}^* = \frac{C_i - h_{ij}}{m_{ij}} \qquad (2.10)$$

From each solution, x_{ij}^*, we extract one point, $P_{ij}' = (x_1', x_2', x_3')$, from sub optimal point, $P = (x_1, x_2, x_3)$, by only changing the corresponding parameter to x_{ij}^* and keeping the rest of them the same:

$$P_{ij}'(x_1', x_2', x_3') = \begin{cases} x_k' = x_{ij}^* & k = j \\ x_k' = x_k & k \neq j \end{cases}, k = 1,2,3 \qquad (2.11)$$

Out of these nine points, the one that satisfies all of the boundary conditions and also maximizes the video quality will be selected as the optimum point. Per our discussion in

Section 2.3, there might be times that a chip instance is severely affected by process variations to an extent that even with adaptation it does not qualify for product specification and therefore it should be discarded.

It is clear that our algorithm is independent of the number of parameters. One can use additional parameters for characterization and the algorithm is still capable of finding the optimum parameter set. In addition, this approach can be applied on any other application that has the same characteristics, meaning its metric functions are monotonic. In a nutshell, the algorithm derives the optimum values for parameters that satisfy all boundary conditions and maximize the target functions, which are all monotonic functions.

2.6 Experimental Results

We have designed an experimental test-bed and methodology to emulate the effects of variation induced frequency degradation on an embedded and server implementation of the H.264 encoder, and to demonstrate the efficacy of the proposed H.264 application adaptation algorithm. In this section, we first describe the test-bed and methodology used. Next, we present results of the frequency degradation of the H.264 encoder due to variation on an embedded platform, and the effectiveness of the proposed adaptation algorithm to tolerate the variation effects. In the last subsection, we study process variation effects on a server implementation of H.264 encoder.

2.6.1 Test-Bed and Methodology

We have implemented the proposed adaptation approach to the x264 encoder [X264] application, which is the most popular and efficient H.264 implementation, and evaluated it using the Beagle Board platform [BGL15] and an Intel Core i7 based server platform. Beagle board (shown in Figure 2.4(a)) is a low-power, low-cost single-board embedded system based

on Texas Instrument's OMAP3530 system-on-a-chip. OMAP3530 includes an ARM Cortex-A8, graphic accelerator, image accelerator, and DSP cores. The x264 encoder application is implemented on the ARM Cortex-A8 running at 600 MHz, and enables real time encoding of smaller resolution videos like QCIF at a rate up to 15 frames per second. We will discuss the details of the server-based implementation in Section 2.6.3. To measure the encoded video quality, we use the MSU Video Quality Measurement Tool [MSU15].

We next describe how we emulate the process variation effects on the x264 encoder application. The primary effect of process variations we are addressing in this work is frequency degradation of the underlying hardware. Hence, we would like to set the hardware platform frequency to different values to emulate the variation impacts. Since the frequency changes on the OMAP platform are limited to a small number of discrete values, we use an alternative approach. We developed a CPU loader application, which runs on the Cortex CPU, and which can keep the CPU busy for a desired length of time. To achieve a CPU frequency degradation effect of D% on the x264 application, the CPU Loader is executed such that D% of the CPU time is used by itself, leaving the rest for the x264 application. The degraded frequency is also provided to the adaptation algorithm, which computes the best parameter values as described in the previous section.

As mentioned in Section 2.1, we are looking at use cases where x264 will have to process video streams in real time. Therefore, if the underlying CPU frequency is degraded, it may not be possible to encode all frames; thus, dropping a portion of frames would be necessary. To emulate real-time behavior in the x264 encoder, including frame drops, we added a timer in the encoder to track frame timing and to drop delayed frames if necessary. We name the new real-time encoder RTx264, as shown in Figure 2.4(b). In summary, the video encoder application is implemented in form of real time x264 software, the process

variation impacts are emulated by CPU Loader, and the adaptation algorithm is executed offline for each given frequency degradation.

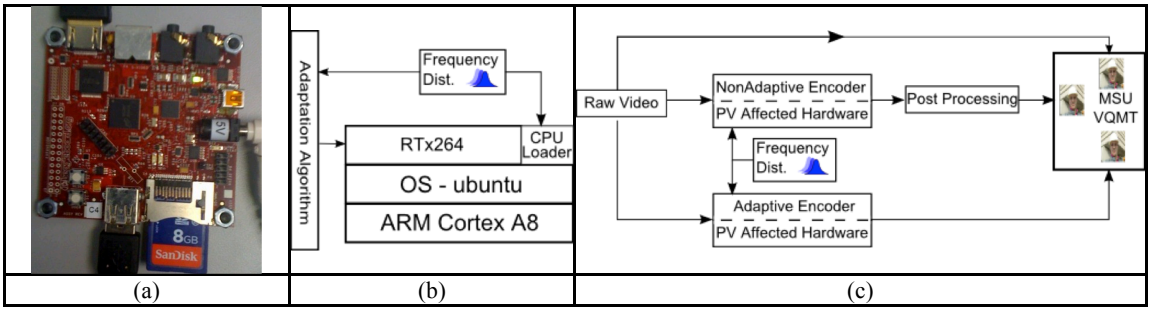


Figure 2.4 Test-bed for measuring the effectiveness of x264 application adaptation. (a) Beagle Board used to implement the x264 encoding application, (b) Software and hardware layers of the test bed, (c) Framework for measuring quality of video produced with variation-affected hardware, without and with adaptation.

Figure 2.4(c) shows the setup used to evaluate the proposed adaptive encoder. The same video sequence is fed to a variation affected non-adaptive encoder, and the equally variation affected adaptive encoder. The resulting bit streams are quantified for video quality based on the original video with the MSU VQM tool [MSU15]. However, this tool requires that the encoded video contain the same number of frames as the reference (original) video. On the other hand, frequency-degraded encoder's output video contains a lesser amount of frames compared to its original input sequence. To mitigate this problem, as shown in Figure 2.4(c), we adopt the following post-processing procedure for the non-adaptive encoder. We developed frame replicator software that replaces each dropped frame with its predecessor-encoded frame. It produces the same visual impression while making the input and output frame count the same. For instance, if the original video has five frames and only the first, second and fourth frames are encoded, the frame replicator produces the following 5 frames sequence from the encoded stream: {1,2,2,4,4}. Note that this post-processing does not need to

be applied on the video encoded by the adaptive encoder application, as the adaptation ensures that all frames are processed in time, and no frames are dropped.

2.6.2 Results: Embedded Platform

To assess the video quality of the encoded videos, we first create an original encoded video with no added CPU load using the parameters shown in Table 2.2 List of parameters used for original video encoder ("perfect encoder" in Figure 2.4(c)). Next, we use the methodology outlined in Figure 2.4(c) to quantify the encoded video quality produced by the "perfect encoder", non-adaptive encoder executing on variation affected hardware, and adaptive encoder executing on variation affected hardware. We use the PSNR metric of the MSU VQM tool [MSU15].

Table 2.2 List of parameters used for original video encoder

Video Name	Foreman	Foreman
Video Size	176×144	704×576
Number of frames per second	15	15
Total number of frames	300	300
Quantization level	22	28
Group of pictures	6	6
Number of reference frames	5	3
Platform	ARM	Intel

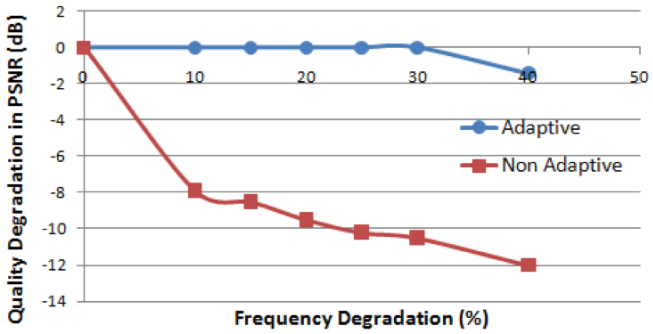


Figure 2.5 Video quality with and without application adaptation on the Beagle Board platform.

Figure 2.5 shows the video quality degradation (y-axis), measured by the PSNR metric, for various frequency degradation (x-axis) magnitudes. As can be seen, the encoded video produced by a variation affected hardware can result in significant quality degradation. On the other hand, the adaptive encoder displays significant tolerance to performance degradation due to process variations: it can peform real-time encoding without any

degradation in video quality even when the processor frequency degrades by upto 30%. Note that frequency degradation above 30%, results in a drop in adaptation algorithm effectiveness.

Table 2.3 Effects of frequency degradation on bitrate

Frequency degradation	Bitrate of adaptive video encoder (ARM)
0	228.4
10	228.4
15	234.8
20	234.8
25	234.8
30	234.8
40	234.8

Table 2.3 Effects of frequency degradation on bitrate shows the effects of frequency degradation on video encoding bit rate for embedded H.264. We observe that the adaptive video encoder can keep the video bit rate very close to the nominal bit rate ($\leq 2.8\%$) while the video quality is sustained (Figure 2.5).

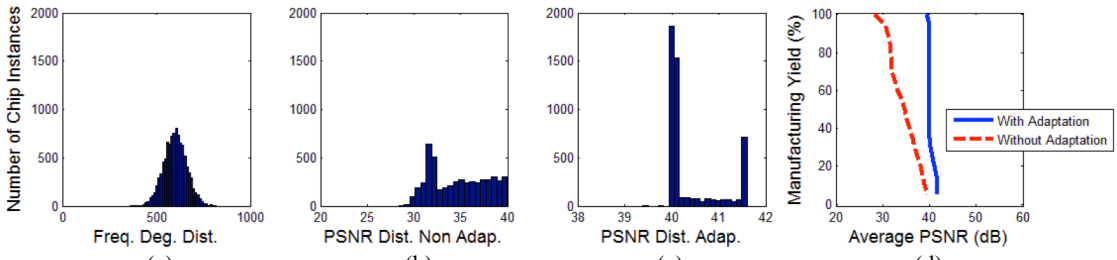


Figure 2.6 Impact of variations on the embedded platform (a) Distribution of frequency degradation, (b) Video quality distribution of non-adaptive encoder, (c) Video quality of adaptive encoder, (d) PSNR vs. manufacturing yield.

Figure 2.6 shows how the non-adaptive and adaptive encoders behave for 10,000 variation impacted instances. To model the variation impacts, we assumed a frequency distribution with $\mu = 600\,MHz$ and $\sigma = 0.1\mu$, which is shown in Figure 2.6(a). We ran a Monte Carlo simulation on 10,000 instances from this distribution; for each instance, we evaluated both the adaptive and non-adaptive encoders. The resulting PSNR distributions are shown in Figure 2.6(b) and (c), respectively. We observe that, for almost all instances, the adaptive encoder results in video quality of at least 40 dB, whereas the non-adaptive encoder in most instances produces video quality below 40 dB. Figure 2.6(d) illustrates how the manufacturing yield varies as a function of the PSNR requirement (we reject all chip instances with quality lower than PSNR requirement). The adaptive encoder is able to achieve close to 100% yield at a PSNR of around 12dB higher than the non-adaptive encoder. These results clearly indicate that the proposed adaptation approach contains significant potential to mitigate the impact of process variations in manufacturing yield without sacrificing end-user experience.

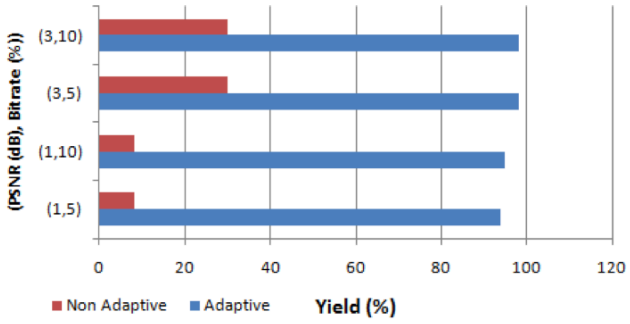


Figure 2.7 Embedded platform yield using original and adaptive encoder, for different (quality, bitrate) constraints.

Figure 2.7 shows the manufacturing yield obtained for the embedded encoder implementation, for different PSNR and bit rate constraints (ϵ, δ), with and without use of the proposed adaptive encoder approach. As the figure shows, use of the adaptive encoder significantly improves the yield for different quality and bitrate constraints. In addition, adaptation is able to keep more than 90% of variation affected chip instances' result in the 1 dB margin of "perfect encoder". In other words, if the inherent tolerance existing in the nature of video encoding application is used, most of the chip instances can tolerate the variation induced effects of hardware on functional performance. Otherwise, loosing these opportunities, hardware variations directly translate into serious impacts on functional performance of video application and many chip instances should be discarded.

2.6.3 Results: Server Platform

To demonstrate the broad applicability of our approach, we also evaluated it on a quad-core Intel core i7 2.7 GHz processor server platform. Note that real-time video transcoding (decoding and encoding) on server platforms has become an important use case with a significant rise in online video consumption from heterogeneous devices of varying resolutions and screen sizes. The major difference between the embedded and server platforms is computational capability. The embedded processor is a single core processor while the server platform contains a multi-core processor, with the ability to encode HD video (720p) at a rate of 25 frames per second. For our experiments, we used only one of the cores in order to avoid dealing with the impact of variations on parallel execution, which is an interesting topic for future research. By using a Virtual Machine (VM), we limited CPU access of the Ubuntu Linux OS to only one core. When only one core is used, the platform can encode 4CIF video (704×576) at a rate of 15 frames per second. As explained before, we ran the CPU loader and RTx264 on the same core to reflect process variation effects.

Figure 2.8 illustrates the video quality degradation of the H. 264 encoder, original and adaptive, for various performance degradations of the server platform. As in the case of the embedded platform, we see similar trends here: there is significant quality degradation without adaptation, whereas the adaptive encoder is able to tolerate up to 20% frequency degradation, without any loss in video quality. We ran the Monte Carlo simulation mentioned earlier for the server platform with $\mu = 2.7\ GHz$ for 10,000 samples.

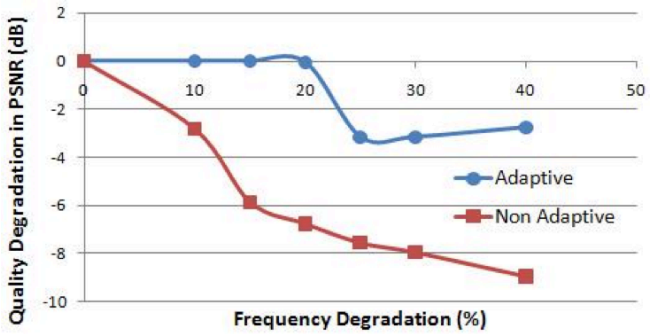


Figure 2.8 Effects of frequency degradation on video quality and effectiveness of adaptation algorithm on server implementation.

We also ran the Monte Carlo simulation mentioned earlier for the server platform with $\mu = 2.7\ GHz$ and and $\sigma = 0.1\mu$ for 10000 samples. Figure 2.9 shows the effectiveness of our algorithm on the video quality distribution and manufacturing yield. Figure 2.9(b) and (c) show that the distribution of PSNR is pushed significantly towards higher values in the case of the adaptive encoder, while the manufacturing yield vs. PSNR plot in Figure 2.9(d) shows the ability of the adaptive encoder to achieve close to 100% yield with approximately 6dB improvement in the PSNR. These results indicate that our approach for adaptation is not limited to embedded platforms or any specific architecture, and can be applied to a variety of platforms. However, its effectiveness may vary from platform to platform.

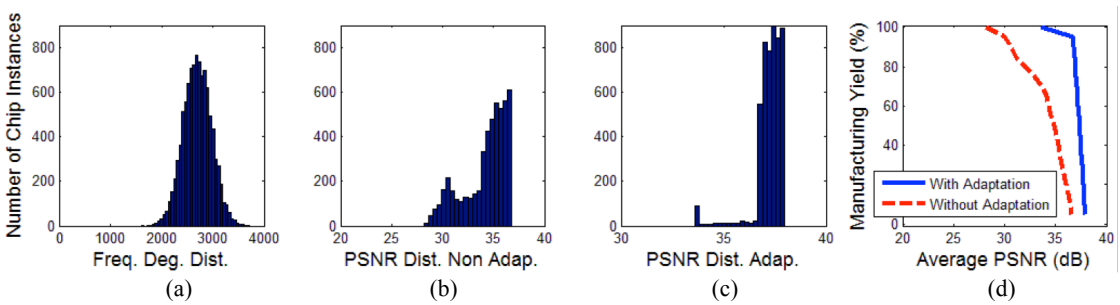


Figure 2.9 Impact of variations on the server platform (a) Distribution of frequency degradation, (b) Video quality distribution of non-adaptive encoder, (c) Video quality of adaptive encoder, (d) PSNR vs. manufacturing yield.

Figure 2.10 shows the significant impact of encoder adaptation in improving the yield of the server platform, reaching as high as 100% when allowing a 3dB PSNR margin and a 10% encoding bit rate margin. In addition, comparing Figure 2.7 and Figure 2.10, we observe hardware platform differences (Cortex A8 vs. Core i7), results in different yield given the same functional performance boundaries. For instance, considering 1 dB margin in quality, embedded platform yield is less than 10% while server platform yield is slightly over 20%. On the other hand, by using adaptation and thereby using the opportunities inherited from video application toward variations, yield increases in both platforms greatly.

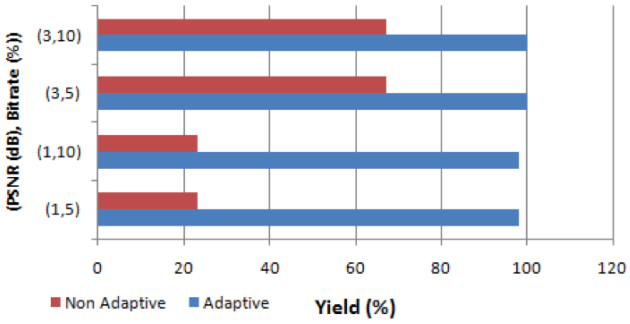


Figure 2.10 Server platform yield using original and adaptive encoder, for different (quality, bitrate) constraints.

2.7 Conclusion

In this paper, we presented an application adaptation technique to address the effects of process variations on applications such as multimedia and video encoding, and applied it to an H.264 encoder. Several parameters of the encoder were characterized and investigated to find those that have the most significant effects on video metrics such as bit rate, video quality, and run time. We proposed an adaptation algorithm to vary the values of these parameters to recover from the frequency degradation in the underlying hardware. We developed an experimental framework to emulate process variations and demonstrated the effectiveness of our approach on the Beagleboard embedded platform with the ARM Cortex-A8 processor and a server platform with the Intel Core i7 processor. We believe that the proposed approach can allow applications to share the challenging task of variation-tolerance, reducing the need to fully contain the effects of process variations through hardware techniques. The proposed approach also can be extended to a range of "elastic" applications in the domains of audio, image and video processing.

Acknowledgements

This section is co-authored with Dr. Sujit Dey and Dr Anand Raghunathan. In addition, authors want to acknowledge the discussion and comments made by Dr. Sara Zarei on parts of this section. The work presented in this section was supported by the National Science Foundation under Grant No. CNS-0917354.

Chapter 3

An Application Adaptation Approach to Mitigate Impact of Dynamic Thermal Management on Video Encoding

3.1 Introduction

In chapter 2, application adaptation was used to address process variation, which is a static problem. It means once a chip is affected by process variation, the its effect would not vary any more. In this chapter, application adaptation is used to address a dynamic problem that varies in time: thermal management and its side effects.

The increase in the transistor scaling, along with the rise in the complexity of today's semiconductors have resulted in a higher power density [SCC10], and hence increased the chip temperature [KSP08]. Chip's high temperature reduces its lifetime, functional reliability, and performance, while increasing its cooling costs [GQ11]. Due to limitations of cooling such as high costs for servers and data centers, or space and/or power requirements in portable devices such as smartphones and laptops, dynamic thermal management (DTM) has been developed as a supplement and sometimes, an alternative solution to the conventional cooling for thermal management of semiconductors [GQ11]. Several DTM methods have been developed for servers [PGP10; HAC11], general-purpose processors [KS04; JP07],

multiprocessors [AKC09; GGP12; JRP08], mobile platforms [SHK10], and embedded systems [ZC10]. While these techniques are efficient in managing temperature, they can introduce different types of performance related overheads or bottlenecks that effectively reduce the platform performance. For example, DTMs reduce the maximum number of instruction a CPU can execute in one second (MIPS) by reducing its frequency. This reduction in platform performance, caused by DTM, would simply translate into longer execution times for ordinary applications, but real-time applications carry timing constraints and a system performance reduction would impact these type of applications much more significantly. In case of hard real-time applications such as medical devices or vehicle control systems, timing constraint violation can be catastrophic, thereby absolutely unacceptable. In case of soft real-time applications such as video encoding or graphics rendering, the timing violation would affect application quality. However, the quality of an application, such as visual quality of an encoded video, not only is dependent on the platform performance, but also is dependent on how the application is designed to use the platform resources. In this chapter, we will develop an application layer technique (application adaptation) to ensure that the negative impact of DTM on the platform performance has minimum effect on quality of applications.

According to several recent market research reports, video traffic will grow significantly compared to other types of Internet traffic. For example, according to the Cisco Visual Networking Index [CSC14], the percentage of all form of IP video traffic (TV, video on demand [VoD], Internet, and P2P) out of all Internet traffic will increase from 66% in 2013 to 79% in 2018. Considering the growth rate of Internet traffic, video traffic will grow three fold from 2013 to 2018 globally while a service such as file sharing services will only grow 3%. The above indicates that video applications, including video encoding/decoding, will dominate the workload of servers, personal computers and mobile platforms. Hence, in this

chapter we focus on the H.264 video encoder, the most widely used video encoding standard, as the focus application. We study the effect of DTM on a H.264 video encoder, and develop a DTM-aware dynamically adaptive video encoding technique, that will vastly reduces the impact of DTM on the encoded video visual quality.

Forced convection cooling (using fan) is one of the most commonly used cooling methods in different computing platforms. While fans help with cooling, they will add acoustic noise to the environment [ZXL12], and consume additional power [XYP13]. In this paper we study the effects of fan speed on video encoder quality and show that by using the proposed adaptation technique, we can reduce fan speed and still achieve better video quality compared to conventional DTM with maximum fan speed.

3.1.1 Effects of DTM on H.264 Video Quality

Real-time video encoding which is a very complex application is required to perform its tasks within a given hard deadline (frames per second). Figure 3.1 shows the results of encoding a VGA video clip of 24 seconds with an x264 encoder (a very efficient implementation of H.264 encoder [X264]) implemented on a MacBook laptop; with and without the use of a commercial DTM tool, CoolBook [ML15] while the laptop fan is rotating at its maximum speed which is 6200 revolutions per minute (rpm). As shown in Figure 3.1, due to its high computational needs, running the x264 encoder without DTM will result in temperatures rising above 90^{O}C, while the encoded video quality is 44 dB PSNR (peak signal to noise ratio, the most widely used video quality measure). On the other hand, DTM maintains the temperature close to the desired value of 75^{O}C; however, the use of DTM reduces the hardware performance, resulting in the encoder not being able to meet its hard deadlines, and consequently significantly degrading the resulting encoded video quality (about 13 dB in this experiment). The above experimental data first shows that the cooling (fan) is

not sufficient for thermal management when a complex application is running on the platform. In addition, while available DTM techniques may be capable of maintaining a systems' temperature at desired levels, their use is unacceptable from the system's perspective, in particular for real-time applications due to loss in quality.

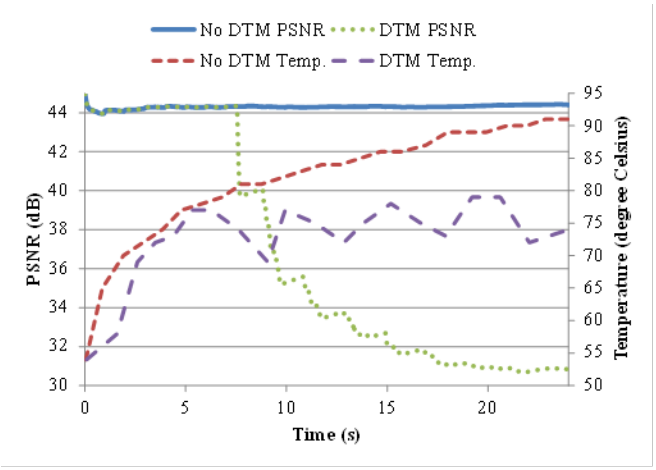


Figure 3.1 Effects of DTM on Visual Quality of a real-time video encoder.

3.1.2 Our Approach

Our DTM-aware adaptive video encoding approach is based on the following two key properties of video encoding applications. First, there is a correlation between the amounts of computational work needed, the bit rate of the encoded video stream, and the quality of the resulting encoded video. Second, there is an inherent tolerance towards a nominal variation from the specification of both the desired bit rate as well as the video quality. The above two properties allow us to design DTM-aware adaptive video encoder such that the applications would be able to adapt to the dynamic changes in hardware performance due to dynamic thermal management without causing any perceivable degradation in user experience (visual

quality). We note that it is important to recognize and consider the multi-dimensional nature of the tradeoff between encoding speed, video quality, and bit rate. For example, it is possible to drastically improve encoding speed (while maintaining video quality) by only utilizing the I-frame (I-frames have been defined in Section 3.2), or even by skipping the frame compression. However, both of these options would lead to unacceptable increases in the video bit rate.

The problem is that while it is necessary to use DTM to maintain platform temperature, this usage would degrade the performance of the platform. This performance reduction causes each tasks running on the platform to take longer time. In the case of real-time video encoding, this execution time increase will result in violation of tasks deadlines and eventually some of the frames would drop from the encoded video sequence. These frame drops result in visual quality decrease that can be measured using PSNR metric (Figure 3.1).

In this chapter, we develop a systematic approach to design a DTM aware real-time video encoder, based on the H.264 standard, to address the above problem. To achieve this goal, we first study the parameters used in the encoder, and model their effects on different metrics of the video encoder, namely encoding speed, video quality, and bit rate. We also characterize the impact of DTM on the application (encoder) run time, and produce a thermal policy characterization (TPC) table. Subsequently, we develop an application adaptation algorithm, which uses the identified parameters, their metrics model and the TPC table, to dynamically adapt the video encoding application in real-time. This process changes the tasks of the video encoder in order to compensate the DTM-induced performance effect on video quality, in a way that the impact on the encoded video quality is minimized. In summary, we use the inherited tolerance in video applications and dynamically eliminate the negative effect of DTM on visual quality by marginally increasing the bit rate. Using the same experimental setup described in Section 3.1.1, we demonstrate that use of the dynamic adaptation technique

in conjunction with DTM can reduce the average drop in video quality to only 2.4 dB at a marginal 4% bit rate increase, as opposed to the 9.8 dB video quality loss when DTM is applied without the dynamic adaptation. We perform extensive experiments using different videos and different DTM profiles to demonstrate the ability of the proposed dynamic adaptation approach to mitigate the video quality loss that DTM otherwise results in.

3.1.3 Related Work

Several adaptive bit rate streaming [LRP05] techniques are being adopted to encode and deliver video with optimal video bit rates to address changing network conditions. In adaptive bit rate streaming [LRP05], a set of different bit rate optimized streams of the same video are encoded and stored on a server in advance, and based on the network conditions, one of the bit rate versions is selected. However, it should be noted that adaptive bit rate streaming techniques do not dynamically change the computational complexity of video encoding, but rather the encoding bit rate, and hence cannot be used to address the negative impacts of dynamic thermal management on the system's computational resources.

Adaptive video encoding techniques have been developed to enable scaling of encoding complexity to enable real-time encoding even on constrained computing platforms, or when the encoder is implemented in software [IB07; IB10]. The main idea is to design a single video encoder application that is capable of running in real-time on different platforms without any platform specific programming such as assembly codes. The encoder regulates its complexity based on average encoding time by using faster but less optimal versions of various encoding steps (including motion estimation). Adaptive techniques have also been developed to make video encoding energy aware, primarily in the motion estimation step [SBH10]. These techniques, while changing the encoding complexity, do not address the fast and dynamic changes in the performance of the same platform that dynamic thermal

management produces, which is the focus of our work. The other difference between our work and these techniques is that we recognize and exploit the multi-dimensional nature of the tradeoff between encoding speed, video quality, and bit rate. Therefore, our adaptation methodology not only maintains real-time behavior of the encoder, but it also works under the specified bit rate constraint.

On the other hand, there have been recent efforts in modifying or developing new thermal management techniques for multimedia applications such as video encoding or decoding [GQ11; YLK07; YK08; LPP06; LPP08; HV12; FS13; PSA14]. In most of these research efforts [GQ11; YLK07; YK08; LPP08; HV12], existing DTM techniques have been modified to ensure real-time encoding/decoding, by making use of the additional time available from processing low complexity video frames. In [FS13], the authors developed a new thermal management approach based on balancing the encoding workload between different components of a video encoder in a multi-core processor platform. Based on the observation that there is a residual time after decoding each frame; a new scheduling-based DTM for MPEG-4 video decoding is proposed [YLK07] which utilizes the residual time for thermal management. Palomino et al. [PSA14] designed an application specific DTM for high efficiency video coding based on adopting dynamic frequency scaling for residual time of encoding each frame and motion characteristics of different video sequences. This way, they proposed an application specific DTM based on DFS. In [LPP08] the residual time of decoding each frame is used along with DVFS to reduce the temperature. If temperature-drop is not enough more residual time is borrowed by dropping a frame or just dropping its quality.

The above approaches have several limitations. First, they require modifications of existing DTM techniques, or development of new ones; therefore, they cannot be used with existing platforms. Second, some of these approaches are platform specific, for instance,

[YLK07] and [LPP08] are only based on dynamic voltage and frequency scaling, [FS13] is based on multi-processor platform for load balancing. Therefore, they cannot be considered as general solutions. Third, some of these techniques have a very limited applicability; for example, a DTM method which works for MPEG-2 video decoding based on residual time per frame [LPP06] is only effective when the computational requirements of an application is much less than the computational capacity of the system. Therefore this cannot be applied to computationally intensive cases such as video encoding or decoding of complex standards like H.264, or even the decoding of high frame rate videos. Fourth, some of the proposed methods [YLK07, PSA14] are application specific DTM methods. They change the DTM decisions based on the requirements of a specific application. This might have negative impact on other applications' quality or it may be impractical to modify the DTM of a general processor for just one of its applications.

In contrast, our approach focuses on adapting the application (encoding) to the dynamic changes induced by DTM, instead of changing DTM decision to optimize for a specific application. Therefore, this method does not affect other applications' quality. It also means our approach will work with existing DTM software/hardware, without the need to develop application specific DTM techniques for every application. Moreover, since our approach is independent of the exact DTM mechanism, and is only dependent on performance reduction induced by DTM, our approach can be used with many different DTM methods (DVFS, multicore methods such as load balancing and deep-sleep modes, clock gating, fetch gating, etc.). Thirdly, our adaptation algorithm does not need to make any assumptions about computing or residual time, and hence can be applied to any video encoding standards, and videos containing different spatial or temporal characteristics. Finally, we believe this is the first attempt to dynamically control video encoding complexity in response to dynamic

thermal management, and perform multidimensional optimization of encoding speed, quality, and bit rate.

The rest of this chapter is organized as follows. In Section 3.2, we will first discuss a set of encoding parameters and describe their effects on encoding speed, quality, and bit rate. Next we review computing platform's thermal dynamics and DTM control mechanisms. After reviewing DTM and video encoding backgrounds, we formulate the DTM-aware adaptation of a video encoding as a multidimensional optimization problem, and describe the overall platform with interactions between the adaptive encoder, DTM controller, and the hardware. In Section 3.3, after analyzing the effects of adaptation on video encoding metrics, we discuss a general method to reflect DTM effects on platform speed/performance and then define our adaptation algorithm in detail. We will provide the details of our test platform and the results in Section 3.4, and conclusions in Section 3.5.

3.2 DTM Aware Adaptation: Approach and Problem Formulation

As stated in the previous section, our goal is to develop an adaptation technique associated with the H.264 video encoding, capable of monitoring the real-time impact of a given DTM process. The adaptation should adjust the encoding tasks dynamically so as to mitigate the impact of DTM on video encoding quality as much as possible. We begin this section by introducing the relevant video encoding parameters that we will use for encoding adaptation, and point out the tradeoff associated with parameter selection for encoding speed, video quality, and bit rate. Then we briefly introduce thermal dynamics of a computing platform and the opportunities a DTM method uses to control the temperature. Next, we formulate the DTM aware video encoding adaptation as a multidimensional constrained

optimization problem. Finally, we introduce the core blocks of adaptive platform, namely encoder adaptation and DTM controller, and discuss how they interact with each other.

3.2.1 H.264 Video Encoding Parameters and Tradeoffs

In general, video encoders are lossy data compressors that reduce the size of a raw video sequence based on two factors, humans' limited visual perception and redundant information in the video sequence. The influence of these two factors on video encoding can be tuned by some of the encoder parameters. The parameters of interest are quantization, group of pictures (GoP), number of reference frames, and search range. Selection of these parameters impacts the encoded video metrics such as video quality, bit rate, as well as the speed of video encoding. In the following paragraphs we introduce these parameters and briefly describe their effects on the encoding metrics.

Quantization is the main parameter that influences encoding based on humans' limited visual perception. It is used by the encoder to drop out/remove video sequence details that are unlikely to be perceived by the human eyes; thereby, reducing the amount of encoding needed. Higher quantization results in the removal of more video details and less computation time. In addition, using higher quantization also significantly reduces the bit rate and the file size of encoded streams. On the other hand, higher speed and lower bit rate comes with the price of video quality loss/reduction.

Group of pictures (GoP), number of reference frames, and search range are the parameters that mainly influence the encoder's effort to identify redundant information in the video sequence. It is necessary to introduce H.264 frame types to explain the role of above-mentioned parameters in the encoding process. There are 3 different frame types I, P, and B in H.264 standard. When a frame does not require any other frames to be decoded and is encoded only with its own information, it is called an I-frame. A P-frame can use its own data or the

data from previous frames to encode, thereby making it more compressible than I frames. Finally, B-frames are frames encoded based on the information from both previous and forward frames; therefore, they have the highest amount of data compression. The maximum effort for reusing redundant data occurs in B-frames while minimum efforts are made in I frames.

GoP size is the total number of frames starting from an I-frame and a sequence of P and/or B-frames, until the next I-frame is encountered. Higher GoP translates into better usage of redundant video information and a reduction in the final bit rate of the video encoder. However, this reduction comes with the price of higher computational needs, longer encoding time, and maybe even a lower video quality. The maximum number of frames needed to search for redundant information and the search range in each frame are set by number of reference frames and search range parameters. It is clear that using higher values for these parameters will result in lower bit rates and better video quality, but at the expense of more computational requirements and run time.

3.2.2 Thermal Dynamics and DTM

In this section we briefly discuss the contributing elements that impact computing platform's (e.g. processors) temperature. In addition, we will discuss the DTM control mechanism of these elements that will have significant negative impacts on application quality

Computing platform temperatures can be described using the formula introduced by Skadron et al. [SAS02]:

$$T' = \frac{P}{C_{th}} - \frac{T}{C_{th} \cdot R_{th}} \qquad (3.1)$$

where T is the silicon junction temperature relative to the ambient temperature, T is the rate of temperature change, P is the power consumed by computing platform, C_{th} is the thermal capacitance, and R_{th} is the thermal resistance of the computing platform. Thermal resistance and capacitance depend on the characteristics of the computing platform, such as its material and shape. On the other hand, system power consumption is dependent on design architecture (number of cores, pipeline, etc.) and the corresponding executing application. Power has two components: leakage (P_{leak}) and dynamic (P_{dyn}):

$$P = P_{leak} + P_{dyn} \qquad (3.2)$$

Traditionally, dynamic power had greater contribution than leakage power to total power consumed in computing platforms. Therefore, most of the techniques target dynamic power portion as the key contributor in system temperatures. The well-known formula for dynamic power of digital circuits is:

$$P_{dyn} \propto \alpha V^2 f \qquad (3.3)$$

where V is the circuit voltage, f is the clock frequency and α is the activity factor of the circuit [SSS04]. The activity factor is the average number of signal transitions in a circuit during one clock cycle. It is commonly used in the circuit level design and analysis, but calculating it for a complete system, which runs a complex application, is time consuming and impractical. Workload, w, (CPU usage for processors) is a normalized number that can replace the activity factor in this context. Workload is defined as the ratio of used computational resources to computational capacity in a platform. Therefore, dynamic power in terms of workload can be shown as:

$$P_{dyn} \propto wV^2 f \qquad (3.4)$$

Equation (3.1) describes a first order differential equation with the steady state solution of:

$$T_{steady} = R_{th} \cdot P \qquad (3.5)$$

While conventional cooling techniques target thermal resistance of the above equation by adding a fan or heat sink, DTM techniques mostly impact the power consumption for controlling systems temperature. This is achievable by affecting the voltage, frequency, architecture-level knobs, or the workload. For example, a lower fan speed is more desirable due to lower acoustic noise and power consumption of fan. However, this reduction increases the thermal resistance and therefore increases steady state temperature. Therefore, DTM should put more effort to bring down the power and reduce the temperature. This higher effort of DTM can cause lower system performance and lower application quality which is not desirable.

Figure 3.2 shows the DTM and application adaptation control loops. The solid-line of the figure depicts the conventional DTM based system. The DTM controller compares the computing platform's temperature with target temperature and based on their proximity, makes decisions to change frequency (f), voltage (V) or parallelism levels (w) in the computing platform. According to equation (3.4), these changes will affect the computing platform's power, and hence will affect its temperature (equation (3.5)). On the other hand reduction in the hardware frequency or parallelism levels (pipelines, cores, etc.) is equivalent to the system's computing resources reduction and will affect the functional quality of a complex application (e.g. video encoder) that heavily utilizes the system's computing

capacity. Thereby, speed, visual quality, and bit rate of video encoder application will be affected by DTM.

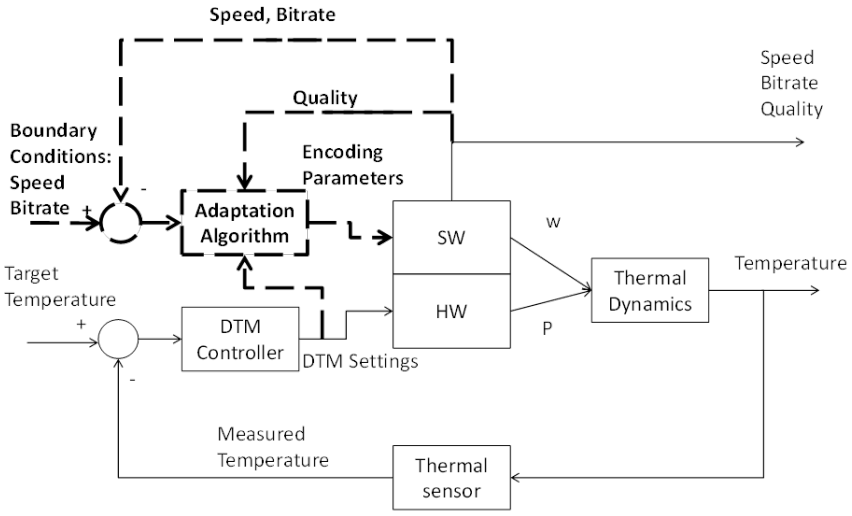


Figure 3.2 DTM control mechanism and adaptation algorithm. The solid line section of the figure describes conventional DTM control mechanism. The dashed line section shows our adaptation method that is added to DTM control loop to reduce DTM negative effects on video visual quality. *W: Workload. P: Power.*

The dashed-line section of Figure 3.2 depicts the application adaptation control loop added to the DTM system. Application adaptation monitors application metrics, compares them with the required metrics (which are called boundary conditions and are described in more details in Section 3.2.3), and monitors the DTM instructions given to the system. Then considering application characteristics, requirements, and current DTM settings, it changes application complexity to fit the system's available computing capacity. Therefore, the negative impact of DTM on the system's computing resources would be compensated in the application level.

On the other hand, changes in application complexity translate into changes in the system work load (w). Thereby, according to equation (3.4), it affects systems power consumption and hence it's temperature. Therefore, even though the adaptation algorithm addresses DTM's negative impacts on application metrics, it may have a positive impact on the thermal behavior of system as well.

3.2.3 Adaptation Problem Formulation

As discussed in Section 3.2.1, we can potentially use the four video encoding parameters to affect the computational complexity and thus the speed of video encoding. Therefore, when video encoding quality is impacted by DTM (frame drops and quality loss due to hardware performance degradation), choosing the right parameter values can help with reducing the computational needs of the encoder while allowing the encoding of video sequences in real time (avoid any frame drop). On the other hand, any such change in the values of the encoding parameters can affect the bit rate as well as the quality of the encoded frames. Dynamic adaptation algorithm is responsible for addressing the performance impact due to DTM, while keeping the encoded video quality as high as possible, and controlling the bit rate.

We formulated our dynamic adaptation algorithm problem as follows. We considered 3 video encoding systems: the first encoding system, E_o, is the original encoder on hardware with no DTM technique; the second encoding system, E_n, is also based on the original encoder but is on a DTM incorporated platform, resulting in dynamic changes in performance; third encoding system, E_a, uses adaptive video encoder and the same DTM based platform as E_n. The metrics of each encoder, E, is defined as follows:

$Q(E)$: Video quality of E in terms of PSNR.

$B(E)$: Bit rate of E in terms of kilobits per second.

$S(E)$: Encoding speed of E in terms of frames per second.

The goal of adaptation algorithm is to dynamically search and identify the optimal values for encoding parameters so even in the presence of DTM, the encoder can maintain its speed with the maximum quality. In Addition, this should not lead to a sizable increase in bit rate results. This goal can be shown in the following constrained maximization form:

$$\begin{aligned} &\text{Maximize: } Q(E_a), \\ &\text{Subject to: } S(E_a) \geq S(E_o) \quad \text{and} \quad B(E_a) \leq B(E_o) + \delta \end{aligned} \tag{3.6}$$

In equation (3.6), δ refers to the acceptable bit rate increase. The δ value has an important role on the significance and purpose of the maximization problem. For instance, if δ has a high value, the solution to the maximization problem refers to a no encoding condition. On the other hand, a small δ will prevent any possible usage of the application tolerance towards variations from nominal specifications that makes the maximization solution insignificant. The selection of δ value for this research has been discussed in Section 3.4.

Note that since the impact of DTM on the encoder speed can vary during an encoding session, satisfying the above conditions will require the adaptation algorithm to be dynamic and will, consequently, require adjusting the parameters in real time to address the DTM impact.

3.2.4 Interactions between DTM and Adaptive Encoder

Figure 3.3 shows the different components and layers of the overall system, with interactions between the DTM controller, hardware driver, and the underlying hardware on one hand, and the proposed adaptation layer on the other hand. The DTM controller makes the decision to change hardware settings based on the temperature readings it receives from the hardware driver. The hardware driver is responsible for reading the temperature from the

digital temperature sensors (DTS) in the hardware, and sending the appropriate commands to reconfigure hardware for thermal management purposes. When the DTM controller changes its settings, it conveys the information to both the hardware driver as well as the encoder adaptation layer. The adaptation algorithm, consequently, adapts the encoder parameters dynamically according to the algorithm described in the next section.

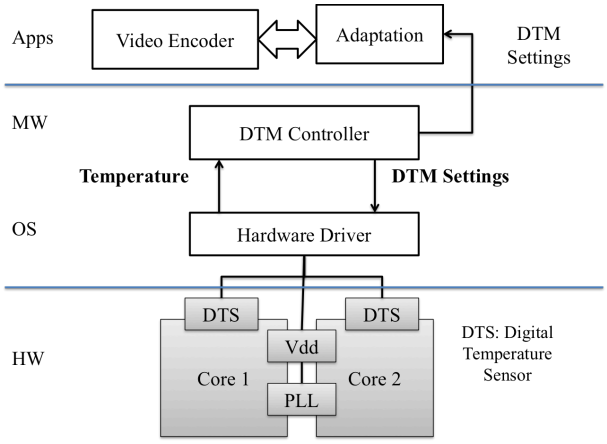


Figure 3.3 DTM Aware Encoder Adaptation Platform – interaction between video adaptation and rest of the computing system.

Even though from an implementation perspective, there is only a one-way communication between DTM and the application adaptation, in practice, the decisions made by the adaptation algorithm will eventually impact the DTM's functionality. As is depicted in Figure 3.2, decisions made by the adaptation algorithm will affect the workload and ultimately change the systems temperature. Meanwhile, DTM is monitoring these effects through thermal sensor readings. In fact, we expect that not only will adaptation directly improve the negative impacts of DTM, but also through its indirect impacts on temperature, it will prevent the DTM

from raising the throttling of system computational capacity. These effects will be discussed in more details in the results section.

3.3 DTM-Aware Adaptation Algorithm

In this section, we introduce our algorithm to dynamically adapt the video encoder in response to the performance changes resulting from DTM decisions. The algorithm uses (a) an empirical model that associates values of the encoder parameters with values of the encoding metrics, and (b) a Thermal Policy Characterization (TPC) table that characterizes the effect of each possible DTM setting on the speed of the encoder. We first describe how we derived the Parameters-Metrics model, and then we discuss how to perform the thermal policy characterization. Finally, we discuss the adaptation algorithm.

3.3.1 Modeling the Impact of Adaptation on Encoder Metrics

In order to enable the adaptation algorithm to make informed decisions, it is necessary to characterize the relationships between the knobs available to it, i.e., the encoding parameters, and the encoding metrics of interest. We formulate these relationships as functions over a multi-dimensional space, $\boldsymbol{C}$, in which each dimension represents one of the encoding parameters. For example, when the encoding parameters used are quantization (q), GoP (g), number of reference frames (r), and search range (s), each point in this space $\boldsymbol{c} = (q, g, r, s)$, corresponds to a specific encoding configuration. The range of values that each parameter can take is constrained by the encoding standard and the encoder's implementation. For the H.264 standard and x264 implementation, we used the following ranges:

$$\boldsymbol{C} = \{\boldsymbol{c} | \boldsymbol{c} = (q, g, r, s) \in \mathbb{N}^4, q \in [1,50],\ g \in [1,250], r \in [1,5], s \in\ [4,16]\} \tag{3.7}$$

For each encoding configuration $\boldsymbol{c} \in \boldsymbol{C}$, the encoder's behavior is characterized by three metrics, which each metric can be modeled as a function over the configuration space, $\boldsymbol{C}$. The first metric, which reflects visual quality, is denoted as $Q(\boldsymbol{c})$. In this work, we choose the widely used peak signal to noise ratio (PSNR) as the visual quality metric. The second metric is the encoding speed, $S(\boldsymbol{c})$, which is measured by the number of frames encoded per second. Finally, the bit rate of the encoded stream is denoted as $B(\boldsymbol{c})$ and is measured in terms of kilobits per second (kbps).

$$\begin{aligned} Q&: \boldsymbol{C} \to \mathbb{R}^+ \\ S&: \boldsymbol{C} \to \mathbb{R}^+ \\ B&: \boldsymbol{C} \to \mathbb{R}^+ \end{aligned} \tag{3.8}$$

Note that the encoding quality, speed, and bit rate are dependent on the content of the input video stream in addition to the encoding configuration. Hence, we utilize a combination of offline training and online calibration in our model. The development of quality metric model, $Q: \boldsymbol{C} \to \mathbb{R}^+$, is described in the next few paragraphs.

The offline training consists of three steps. First, we select a subset of points in the configuration space, denoted as $\boldsymbol{C_s} \subset \boldsymbol{C}$.

$$\begin{aligned} \boldsymbol{C_s} = \{\boldsymbol{c} | \boldsymbol{c} = (q, g, r, s), q \in \{18,23,28,33,38\}, g \in \{1,2,3,8,16,32,96\}, \\ r \in \{1,2,3,4,5\}, s \in \{4,8,12,16\}\} \end{aligned} \tag{3.9}$$

Second, we encode a set of training videos repeatedly and store their quality while setting the encoder parameters by enumerating the configuration points in $\boldsymbol{C_s}$. Due to the large size of the four-dimensional configuration space, $\boldsymbol{C}$, it would have taken more than a year to

encode all of the training videos with all the possible encoding configurations in $\boldsymbol{C}$; therefore only the above-mentioned sub-set is selected and encoded.

In the third step, we expand the collected data from the configuration points in $\boldsymbol{C_s}$, to all of the configuration points in $\boldsymbol{C}$ by means of linear interpolation of the measured video qualities. This provides us with a reference quality function, Q_{ref}: $\boldsymbol{C} \rightarrow \mathbb{R}^+$, that will be used along with the online calibration in the final model:

$$Q_{ref}(\boldsymbol{c}) = \begin{cases} \text{Quality measurement for configuration } \boldsymbol{c} & \boldsymbol{c} \in \boldsymbol{C_s} \\ \text{Linear interpolation from measured neighbors} & \boldsymbol{c} \notin \boldsymbol{C_s} \end{cases} \qquad (3.10)$$

As mentioned above, the reference function developed in the offline steps uses a set of training videos; hence, it suffers from some inaccuracy under variations in the video content. Therefore, we perform an online calibration on the reference function to come up with the model. For this purpose, we take advantage of the fact that video encoders actually measure the metrics of interest, namely the visual quality, bit rate, and speed (encoding frame rate), during the encoding of a video stream. Online calibration follows a history-based approach wherein we divide the encoding session into intervals (these intervals naturally correspond to the intervals at which the adaptation algorithm is invoked to modulate the encoding parameters). We use the actual measurements of quality metric during the current epoch, Q^*, and the value provided by the reference quality function, Q_{ref}, for the current encoding configuration, $\boldsymbol{c}^*$, to come up with a scaling factor to calibrate the reference function and derive the model. This model then is used to predict the quality metrics for the next epoch.

$$Q(\boldsymbol{c}) = \frac{Q^*}{Q_{ref}(\boldsymbol{c}^*)} Q_{ref}(\boldsymbol{c}) \qquad (3.11)$$

We extract the reference functions for both speed and bit rate metrics, S_{ref} and B_{ref}, the same way we do for quality metric while performing the three steps of offline training.

$$S_{ref}(\boldsymbol{c}) = \begin{cases} \text{Speed measurement for configuration } \boldsymbol{c} & \boldsymbol{c} \in \boldsymbol{C_s} \\ \text{Linear interpolation from measured neighbors} & \boldsymbol{c} \notin \boldsymbol{C_s} \end{cases}$$
$$B_{ref}(\boldsymbol{c}) = \begin{cases} \text{Bit rate measurement for configuration } \boldsymbol{c} & \boldsymbol{c} \in \boldsymbol{C_s} \\ \text{Linear interpolation from measured neighbors} & \boldsymbol{c} \notin \boldsymbol{C_s} \end{cases} \quad (3.12)$$

We also define the model for these metrics the same way through calibration of reference function using the online measured bit rate, B^*, and speed, S^*:

$$B(\boldsymbol{c}) = \frac{B^*}{B_{ref}(\boldsymbol{c}^*)} B_{ref}(\boldsymbol{c}), \;\; S(\boldsymbol{c}) = \frac{S^*}{S_{ref}(\boldsymbol{c}^*)} S_{ref}(\boldsymbol{c}) \quad (3.13)$$

To verify and study the accuracy of our modeling approach, we applied it to a set of test videos different from the training videos. Figure 3.4 (a), (b), and (c) show the plots of quality measurements versus model predictions for video clips Avatar, Soccer, and Wildlife, respectively. As the figure illustrates, the model is able to closely predict the visual quality when online calibration is performed. On the other hand, at the start of the encoding session, the calibration is not performed due to a lack of history, and consequently there is a bigger difference between the model and the measurements. Figure 3.4(d) shows the percentage of error from all three modeled metrics of our test videos. We observe that the model can predict all of the encoding metrics with less than a 3.6% error.

Note that the models described in this section are captured on a platform without any DTM enabled. In the next section, we describe how we capture the impact of DTM on encoding speed. In Section 3.3.3, we show how the models are used in an adaptation algorithm to reduce the negative effects of DTM on video encoding.

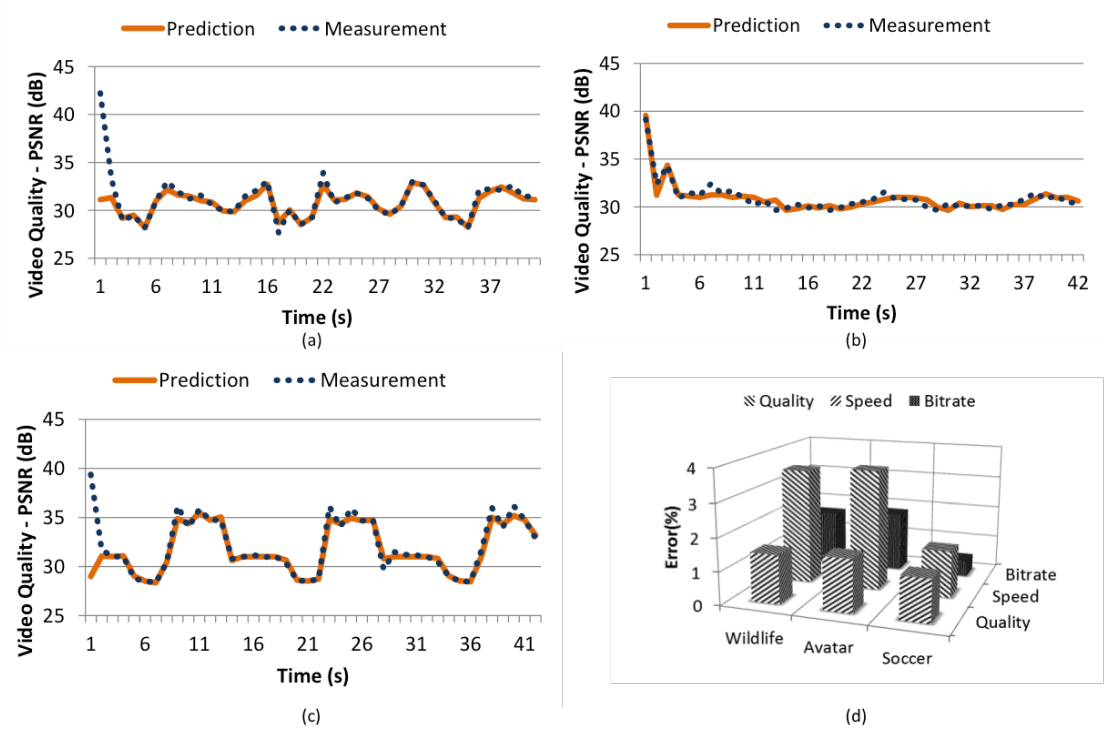


Figure 3.4 Comparison of model prediction and measurement of visual quality for clips (a) Avatar, (b) Soccer, and (c) Wildlife. (d) Percent error of model predictions for Quality, Speed and Bit rate of test videos.

3.3.2 Modeling the Impact of DTM on Encoder Speed

There are different dynamic thermal management methods such as dynamic frequency scaling, dynamic voltage and frequency scaling, clock gating, load migration away from hot cores, etc. In order to use our adaptation algorithm with any of these different DTM methods, we need to quantify and differentiate DTM's negative impact on the systems performance (e.g. instructions per second) and application performance (e.g. task run time) in a uniform manner. For example, different applications may experience different application performance (run time) degradation under the same DTM method depending on whether they are compute or memory intensive. To address these issues, we define Application Performance Index

(API), which captures the impact of DTM on a specific application (in our case, video encoder), and is measured through a characterization process.

We start with a premise that each of the DTM methods has a limited number of settings that are chosen based on the system temperature, and other factors such as the number of cores, number of voltage and frequency levels, etc. At any instant of time the system is operating in exactly one of its DTM settings.

The Application Performance Index is defined as the amount of speed reduction observed in a given application due to the selection and transition from a specific DTM setting to another. If the time an application takes to run under nominal conditions (no DTM) is t_N and the time it takes for the same application to perform the same task in DTM setting S^i while previous DTM setting was S^j is $t_{S^{i,j}}$; then the API for this DTM selection and transition is:

$$API_{S^{i,j}} = t_N \,/\, t_{S^{i,j}} \qquad (3.14)$$

Since application execution under nominal conditions is always equal or faster than any other DTM setting, we have:

$$A0 < t_N \leq t_{S^{i,j}} \Rightarrow 0 < API_{S^{i,j}} \leq 1 \qquad (3.15)$$

We characterize the API for the video encoder under various DTM settings, for a range of training videos and find their corresponding average. The values for each DTM setting selection and its corresponding API are stored in a lookup table which is called the Thermal Policy Characterization (TPC) table. At any time, the adaptation algorithm can look up the TPC table to determine the current performance impact on the encoder due to the current DTM settings.

In effect, TPC provides us with a tool that predicts the speed of an application in the presence of DTM from its expected speed in the absence of DTM. In our case, given the encoder speed in the absence of DTM (predicted by the model in Section 3.3.1) for a given configuration, $\boldsymbol{c}$, one can compute encoder speed in the presence of DTM as follows:

$$S_{DTM}(\boldsymbol{c}) = API \cdot S(\boldsymbol{c}) \qquad (3.16)$$

Note that for a given system, the thermal policy characterization has to be only performed once offline. In Section 3.4.1 we will show the results of the thermal policy characterization on the platform used in our experiments.

In the next section, we will introduce the adaptation algorithm and show how the above formula, along with the models for encoding metrics (from section 3.3.1), are used to address the effects of DTM.

3.3.3 Video Encoder Adaptation

In response to changes in the system temperature, DTM changes its settings, thereby changing the Application Performance Index (API). Given the current API, the objective of the adaptation algorithm is to select a point in the configuration space, $\boldsymbol{C}$, such that the video encoder will still function in real time and satisfy the bit rate constraints, while producing maximum video quality.

In this section, we will first provide an overview of the adaptation algorithm and how it fits into the runtime execution of the encoder. Next, we will describe the adaptation algorithm in detail. Finally, we will analyze the efficacy of our algorithm by comparing it to well-known optimization algorithms such as coordinate ascent and hill climbing.

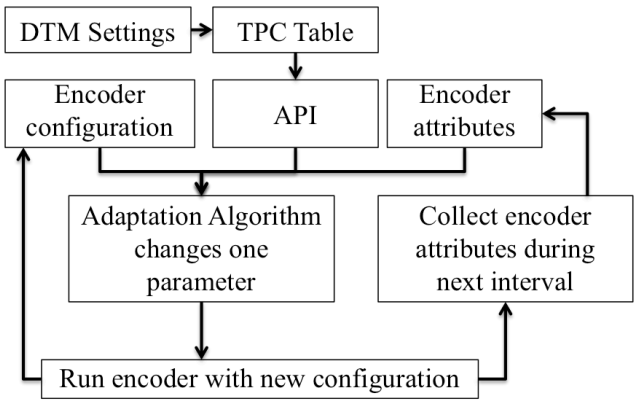


Figure 3.5 Workflow of the proposed adaptation algorithm

3.3.3.1 Adaptation Algorithm: Overview

Figure 3.5 shows the high-level workflow of the proposed adaptation algorithm in the context of the video encoder. As shown, the adaptation algorithm is periodically called to incorporate the DTM effects at the beginning of each time interval. Three sets of data are sent to the adaptation algorithm at the end of each time interval; the encoding metrics which are measured by the encoder during the last time interval; the new application performance index which is drawn from TPC look-up table based on the current DTM settings; and the encoder parameters from the last time interval. The adaptation algorithm starts with the encoding parameters from the previous time interval and iteratively changes one parameter at a time. At each iteration, the parameter whose value is determined to be the most effective in improving the objective function (Equation (3.6)) is chosen. The models constructed for the encoding metrics (Section 3.3.1) are used to guide this search.

encoding configuration in each interval by changing all the parameters simultaneously, that configuration may not remain the best for the rest of the time interval.

This is due to the possible dynamic change in both the video content and application performance index during the iteration.

3.3.3.2 Adaptation Algorithm: Details

We next discuss the details of the proposed adaptation algorithm. Figure 3.6 shows the flowchart of the adaptation algorithm. Each iteration of the algorithm, which corresponds to an encoding time interval, uses the following inputs:

- Encoding configuration from the previous interval: (q^*, g^*, s^*, r^*).
- Encoding metrics measured in the previous interval: Q^*, S^*, B^*.
- Application Performance Index, API, for the current interval, which is extracted from the TPC table based on the current DTM settings.
- Constraints on encoding speed and bit rate: S_T, B_T.

The adaptation algorithm consists of four key steps: *extract*, *scale*, *predict*, and *select*. The first three steps identify the optimum value of each parameter independently and the last task chooses the optimum parameter among all of them for the current iteration.

Since we change only one parameter per iteration, we need to study the effect of each parameter on the encoding metrics separately, independent of other parameters. The first step, *extract*, extracts the relationship between each encoding parameter and each metric from the characterized model; assuming that all the parameters except the one under consideration are unchanged from the previous interval. Since we have four parameters and three metrics, we obtain 12 extracted functions. For instance Q_g, S_g, and B_g are the extracted functions for the GoP parameter. We next show the extraction of $Q_{ref,g}(g)$ from the reference data $Q_{ref}(\boldsymbol{c})$; the same method applies to the other parameters and metrics as well.

$$Q_{ref,g}(g) = Q_{ref}(q^*, g, s^*, r^*) \qquad (3.17)$$

In the next step, *scale*, we scale each of the extracted functions as described in Section 3.3.1 to calibrate the reference function. For example, the scaled function of visual quality versus GoP, $Q_g(g)$, is calculated as follows:

$$Q_g(g) = \frac{Q^*}{Q_{ref,g}(g^*)} Q_{ref,g}(g) \qquad (3.18)$$

Similarly, the *scale* step finds the scaled functions corresponding to each of the other 11 extracted functions.

Given all three scaled functions of one parameter, the *predict* step calculates the optimum value of each parameter, i.e., the value that maximizes the visual quality while satisfying speed and bit rate constraints. Without loss of generality, we describe the functionality of this task for the GoP parameter, but it applies to all other parameters as well. Considering the fact that all the encoding metrics are monotonic functions of the parameters, we know that the optimal values are guaranteed to lie at the boundaries of the feasible solution regions. Therefore, we solve the following two equations at boundary conditions:

$$API \cdot S_g(g) = S_T \qquad (3.19)$$

$$B_g(g) = B_T \qquad (3.20)$$

In equation (3.19), the left hand side is multiplied by the application performance index in order to reflect the impact of DTM on the speed of the encoder (equation (3.16)). Solving the above equations could result in 2 separate values for GoP, g_1, g_2. Of these values, we select the one, g_i, which does not violate the other boundary condition and maximizes $Q_g(g)$.

As shown in Figure 3.6, the *extract*, *scale*, and *predict* steps are repeated for all four encoding parameters. After a value for each of the parameters is selected, they will be compared with each other in the *select* step. The parameter, which maximizes the quality without violating the constraints, is selected as the final choice for the current time interval.

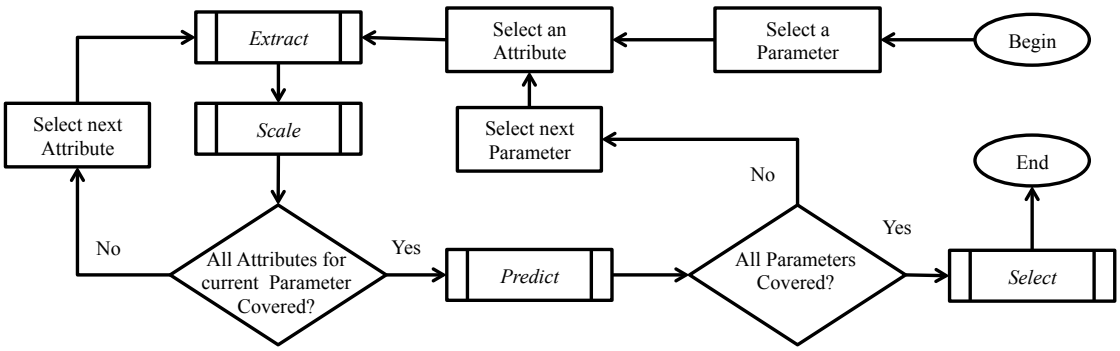


Figure 3.6 Adaptation algorithm details for each iteration.

The encoder is provided with the new parameter values, and the algorithm continues to iteratively adapt the parameters based on the actions of the DTM scheme.

3.3.3.3 Adaptation Algorithm: Discussion

The proposed algorithm effectively takes a greedy, iterative approach to solve a constrained optimization problem. In this section, we describe our algorithm mathematically and relate it to well-known approaches, such as coordinate ascent and hill climbing.

Consider a multivariable function:

$$f(\boldsymbol{x}), \quad \boldsymbol{x} = (x_0, x_1, \dots, x_{m-1}), \quad x_i \in X_i, \quad 0 \leq i < m \tag{3.21}$$

where m is the number of variables and X_i is the domain of the i^{th} variable. The objective is to identify the value of $\boldsymbol{x}$ in which maximizes the value of this function.

From this multivariable function, we can derive a single-variable function at any given point, $\boldsymbol{x^n}$, by keeping all variables other than the i^{th} one, constant. The new function can be mathematically described as follows:

$$f_i^{\boldsymbol{x^n}}(x) = f(x_0^n, \dots, x_{i-1}^n, x, x_{i+1}^n, \dots, x_{m-1}^n), \quad 0 \le i < m,$$
$$\boldsymbol{x^n} = (x_0^n, x_1^n, \dots, x_{m-1}^n) \tag{3.22}$$

The proposed optimization algorithm is to select the next variable, $\boldsymbol{x^{n+1}}$, from current variable, $\boldsymbol{x^n}$, in each iteration in a way that leads to the maximum value of $f(\boldsymbol{x})$.

In our proposed algorithm, all of the variables will be evaluated independently. Then we compare all the variables with each other and select the one with maximum improvement among others. In addition, we restrict the domain of each search through the use of constraints (encoding speed and bit rate). Mathematically, each iteration of the algorithm may be described as follows:

$$j = \arg\max_{i \in [0,m[} \left(\max_{x \in X_i - C} f_i^{x^n}(x) \right)$$
$$x_j^{n+1} = \arg\max_{x \in X_j - C} f_j^{x^n}(x) \tag{3.23}$$
$$x_i^{n+1} = x_i^n, \quad 0 \le i < m, i \ne j$$

where the j^{th} variable is the optimum variable and $X_i - C$ is all of the points in X_i points that do not violate the constraints.

Our adaptation algorithm is a variant of gradient ascent [JS05] optimization in discrete spaces. Gradient ascent is a first-order unconstrained optimization algorithm that finds the local maximum of a multivariable function using its gradient. The gradient ascent algorithm

was originally designed for a differentiable function in a continuous space. However, there are variants of gradient ascent designed for discrete spaces, such as hill climbing and coordinate ascent. We designed our algorithm as a greedy best-first-search implementation of gradient ascent which also considers problem constraints. Next, we briefly discuss the hill climbing and coordinate ascent algorithms and then discuss their similarities and the differences with our algorithm.

In the hill climbing algorithm, the optimized value of each variable is calculated independently from other variables at each iteration.

$$x_i^{n+1} = \arg\max_{x \in X_i} f_i^{x^n}(x) \tag{3.24}$$

However, in the coordinate ascent algorithm one variable is optimized in each iteration. Then, the next coordinate will be selected in the next iteration and this process continues in a round robin fashion.

$$\begin{aligned} x_i^{n+1} &= \arg\max_{x \in X_i} f_i^{x^n}(x), \quad i = n \bmod m \\ x_i^{n+1} &= x_i^n, \qquad 0 \le i < m, i \ne n \bmod m \end{aligned} \tag{3.25}$$

In fact, in the hill climbing algorithm all of the variables can change at one iteration but will be evaluated independently. On the other hand, in coordinate ascent, only one of the variables can change; therefore, only its corresponding direction will be evaluated.

The proposed algorithm covers the same search area as hill climbing, since all of the variables are evaluated. However, when it comes to updating variables, we only update one of

the variables per iteration like coordinate ascent by comparing all the variables with each other and selecting the one that yields the maximum improvement. The reason we do not use the hill climbing method directly, is that it may select points that violate the boundary conditions by changing multiple variables at the same time without considering their interactions. On the other hand, we did not find the coordinate ascent greedy enough for our purpose. Due to the variations in the video contents and DTM effects, the function and boundary conditions can vary in time. Therefore by selecting a variable in round robin order, we might choose the least effective variable and by the time of the next iteration, the opportunity would be lost. Therefore, to utilize all the available opportunities for maximization, we chose our greedy best-first-search over coordinate ascent.

3.4 Experimental Results

In this section, we discuss the experimental results using a MacBook Air, with specifications in Table 3.1 and a bit rate margin (δ) of 15%. Dynamic Thermal Management is done by CoolBook [ML15], a commercial DVFS-based DTM tool that allows programming and the use of different DVFS profiles for thermal management, consisting of different frequency, voltage, throttling, and trigger temperature settings. Throttling means how fast the DTM would react to thermal violation. The CoolBook tool does not provide any more information about this parameter. Trigger temperature is the thermal limit above which the DVFS method will be employed [LPP08]. The temperature results are the real temperature measurements that have been collected by Hardware Monitor [MB15] from the thermal sensors on the MacBook platform. The smc Fan Control [HH15] is used to set the CPU fan speed to its maximum at all times to make sure we are using the cooling effects as much as possible.

Table 3.1 Implementation Platform Specification

Operating System	Mac OSX 10.5.8
Processor	Intel Core 2 Duo
Frequency	1.8 GHz
Memory	2GB

We have used seven profiles; in order to first, study the adverse effects of the DTM on the video encoder; and second, to show the efficacy of the proposed dynamic adaptation algorithm on reducing these effects. We set the trigger temperature to 75℃ for all DTM profiles. Profile 0, which is the reference implementation, incorporates no frequency or voltage scaling and the processor runs at 1.8GHz (the nominal frequency). The thermal management for this profile is only based on system fan and heat sink. In the rest of the profiles, DVFS is enabled to assist with thermal management. Table 3.2 shows the throttling level and frequency settings used in each dynamic thermal management profile and Table 3.3 reports the voltage settings for each frequency. Note that the used voltage settings are among the ones predefined and available in the system.

Table 3.2 Thermal Management Profiles

Profile	Frequencies (MHz)	Throttling Level
0	1800	N/A
1	1800, 1400	Low
2	1800, 1400	High
3	1800, 900	Low
4	1800, 900	High
5	1800, 1600, 1400, 1200, 800	Low
6	1800, 1600, 1400, 1200, 800	High

Table 3.3 Voltage vs. Frequency Selection

Frequency (MHz)	1800	1600	1400	1200	≤900
Voltage (V)	1.175	1.1125	1.05	0.975	0.9

The numbering of the profiles has been selected in ascending order of frequency scaling options. As shown by [KS04], two frequency/voltage levels are sufficient for effective DTM; therefore, profiles 1 to 4 have been selected with only two frequency/voltage levels. Profiles 5 and 6, which use five different frequency/voltage levels, have been selected based on DVFS designer's recommendations.

In the following subsections, we first present the results of the thermal policy characterization (Section 3.3.2) of the test platform. Next, we carry out a detailed study on the temporal effects of DTM on the video encoder and compare the results of encoding with and without our adaptation algorithm. Then, we present results for the overall visual quality of the adaptive encoder for different videos and different dynamic thermal management profiles. Finally we show the effects of fan speed on application quality and show the proposed adaptation help reduce the required fan speed and hence its negative effects.

3.4.1 Thermal Policy Management Results

As shown in Table 3.2, our test platform uses six different frequency settings. Each frequency setting is one state of DTM as explained in the API definition. For each frequency setting, several VGA video clips have been encoded, and the application performance index measured according to section 3.2. The measurements show the API for each state (current frequency setting) is independent of its previous state (previous frequency setting) for the DVFS-based DTM used in the test platform. Therefore, the thermal policy characterization table is simplified to a form only reflecting current DVFS states, which is shown in Table 2.4.

Table 3.4 Application Performance Index for Test Platform

Processor Frequency (MHz)	API(%)
1800	100
1600	89.34
1400	77.72
1200	66.70
900	49.01
800	43.56

We observe that the APIs for the different settings of the test platform are very close to the ratio of CPU frequency to the nominal hardware frequency. This is due to the fact that the performance of the x264 encoder implementation is dominated by computation time, and not memory access time, as the latter has been implemented very efficiently.

3.4.2 Effects of Encoder Parameters

Before studying the effects and efficacy of the proposed adaptation algorithm, we take a look at the effects of the application parameters on the quality and bitrate of the encoded video, and the encoding runtime. As discussed in section 3.2.4 (Figure 3.3) the adaptation algorithm has a unidirectional communication with DTM and does not change the DTM. However, the introduction of the adaptation algorithm to the system has an indirect effect on the thermal management process creating an unintentional feedback loop to the system. This feedback effect was mentioned in section 3.2.2 (Figure 3.2). In this section the numerical results are studied to help understand the aforementioned feedback effects.

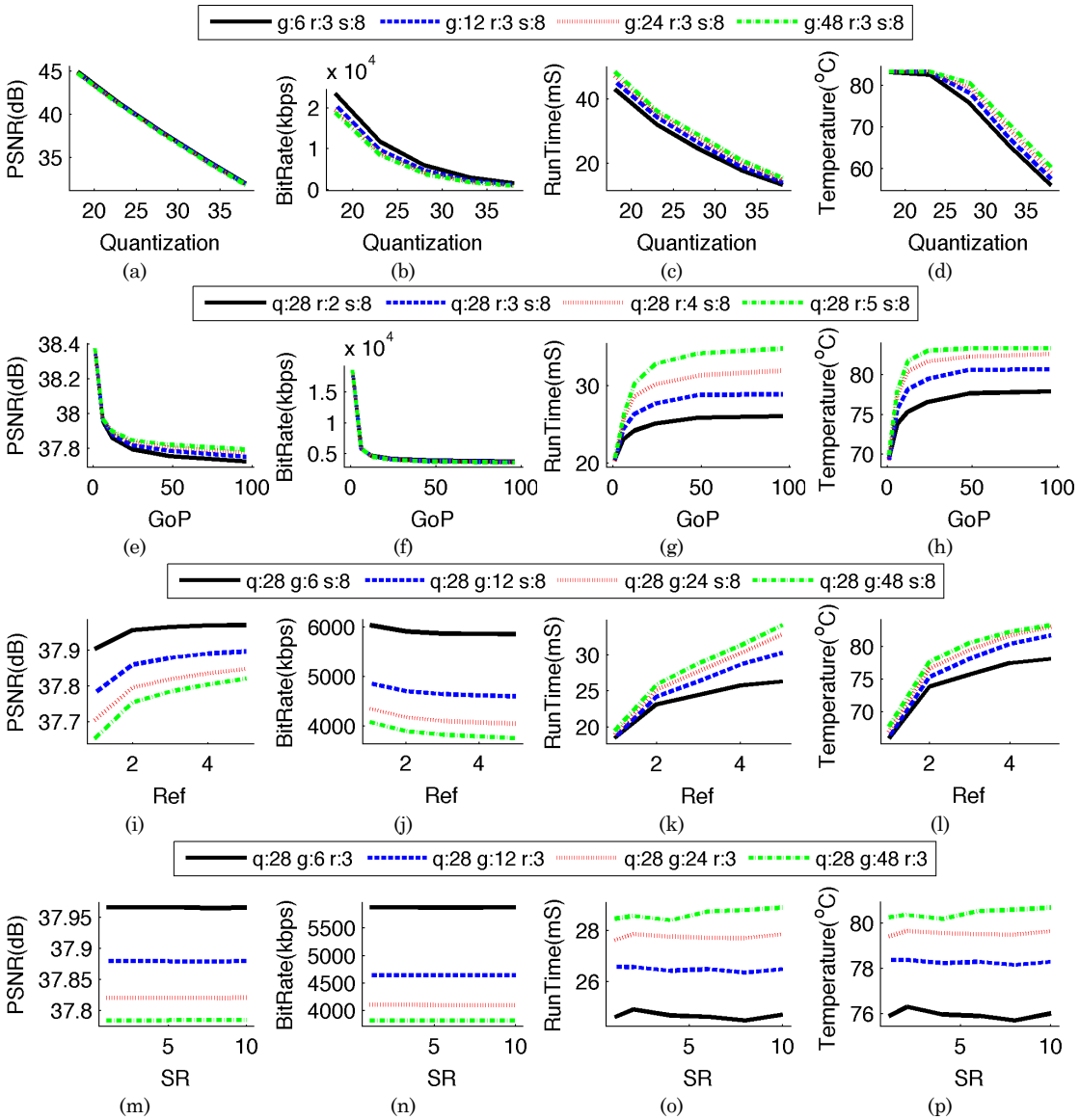


Figure 3.7 Effects of encoder parameters on video quality, video bitrate, encoding runtime, and platform temperature. The x-axis labels are defined as: GoP is for group of pictures, Ref refers to the number of reference frames, and SR refers to search range. The legend labels are defined as: 'q' - quantization, 'g' - group of pictures, 'r' - number of reference frames, and 's' - search range. For each graph, three of the encoder parameters are kept constant, which are mentioned in the legends, while the fourth parameter is used as the x-axis variable. The first column on the left shows the video quality in dB, the second column shows the video bitrate in kbps, the third column shows the average of encoder's run time per frame in millisecond, and the rightmost column shows the average steady temperature of the platform. All of these plots are produced when there is no DTM.

Figure 3.7 consists of 16 subfigures that show the video encoder parameter (quantization level, group of pictures, number of reference frames, and search range) effects

on encoded video quality (PSNR), encoded video bitrate, encoder runtime, and platform temperature. Each subfigure depicts four graphs, with each graph generated by keeping three of the encoder parameters constant and varying the fourth parameter. The values of the three constant parameters are mentioned in the figure legends and the varying parameter is mentioned in the x-axis label. First, the left three columns of Figure 3.7 are studied, namely the effects of the parameter on quality, bitrate and runtime and then the rightmost column and the effects of encoder parameters on platform temperature are discussed. It is crucial to recognize that the general behavior of the graphs in Figure 3.7 are more important than their values since those values will change from video to video. Plots in Figure 3.7 are constructed from data obtained through the characterization process described in section 3.4.1.

Figure 3.7(a) and (b) show that the quality of the video drops linearly by increasing quantization level and at the same time video bitrate also drops. But unlike quality, the rate of change in bitrate is not linear. In fact, as the quantization level becomes smaller the bitrate increases much faster. This non-linear bitrate increase would give a hint that while reducing quantization level would increase the quality, it should be stopped at some point due to huge increase in bitrate. This effect has been considered in the adaptation algorithm by introducing the constraint for bitrate. Figure 3.7 (c) suggests that the increase in runtime due to reduction in quantization level is linear; therefore, one would expect the main reason for not using lower quantization level would be the significant increase in bitrate.

Figure 3.7 (e) and (f) depict that it is better to use higher number of frames in a GoP because it can reduce the bitrate to one third with a modest quality drop. On the other hand, as depicted in Figure 3.7 (g), this bitrate reduction comes with about 50% increase in encoder runtime. If one only considers the quality-bitrate tradeoffs between quantization level and GoP size (Figure 3.7 (a), (b), (e), and (f)), it can be concluded it is better to have lower quantization

level and higher GoP size. That would be true for an offline video encoder; however, for a real-time video encoder, the timing requirement would prevent application of this method. In fact, this effect has been considered in the adaptation algorithm by introducing the timing constraint.

Figure 3.7 (i) and (j) suggest that increasing the number of reference frames increases the video quality and reduces the bitrate. Again without considering the timing requirements, one would simply choose the highest number of reference frames, but as is depicted in Figure 3.7 (k) this would increase the encoding time by approximately 50%, increasing the chance of timing violation and subsequently a drop in video quality. Once again the need to consider the timing constrain in the adaptation algorithm is evident.

Figure 3.7 (m), (n) and (o) suggest that changing the search range does not affect any of the encoder metrics in this test setup (x264 implementation on MacBook Air). It should be mentioned that the encoder is analyzed as a black box and not intended to be modified in its implementation. Only control knobs of the encoder are used to find and choose its optimum configuration during the DTM. As the results show, in this implementation of H.264 on an Intel platform, changing the search range is not effective on encoder metrics; however, we have seen that this same parameter increases the encoder runtime by 30% on an embedded platform (Beagleboard [GC09]) using ARM Cortex A8 as its main CPU. It shows that encoder behavior is significantly platform dependent and the characterization steps mentioned in section 3.3.1 should be performed for each platform.

The above analysis shows the selection of encoder parameters can be straightforward for an offline encoder. Without considering timing constrains (offline encoding), one would choose a large value for GoP and maximum number of reference frames and search range. The quantization level can be simply used for bitrate constraint and the parameter selection

problem is solved. But the parameter selection is not simple in the case of real-time video encoder because of significant variation in encoder runtime and possible timing violation, hence quality drops. Therefore, there is a need for an adaptive solution to dynamically choose these parameters when DTM affects the performance of the platform.

Figure 3.7 (d), (h), (l), and (p) show the effects of varying encoder parameters on platform average steady temperature. The thermal model provided by [MDR14] is used to calculate the average steady platform temperature by varying encoder parameters. These graphs depict that while the change in search range is not affecting platform temperature, the other three encoding parameters can cause the average steady temperature of the platform to vary by about 20 degree Celsius. As shown in Figure 3.2, when the system gets too hot the thermal management starts cooling down the system by reducing frequency and voltage. This frequency drop reduces the overall platform speed, hence video encoding speed as well. This means it would take longer time to do the same job in this new configuration. The proposed adaptation algorithm, which is constantly monitoring the status of DTM, changes encoder parameters to make it faster and compensates for the platform speed reduction to avoid timing constraint violations. It means the adaptation algorithm changes the encoder parameters in a direction in which run time is reduced (Figure 3.7 (c), (g), (k), (o)). Thus far, this is the expectation from application adaptation. However, by comparing the two rightmost columns in Figure 3.7, it is noted that the same change in parameters, which reduces encoder run time, also reduces the platform steady temperature. This additional thermal relief provided by the application adaptation would help the DTM to use higher frequencies in the next time periods and in general use higher average frequency compared to when the DTM is applied without any follow-up adaptation. This phenomenon has been measured and shown in the next section as well.

As mentioned in the problem formulation, the adaptation algorithm maximizes quality by sacrificing bitrate. It is interesting to see the side effect of increasing the bitrate on the platform temperature. By comparing Figure 3.7 (f) with (h) and (j) with (l), we see the aggregate effect of changing encoder parameters which increases bitrate, actually reduces the platform's steady state temperature. This occurs since the complexity of the encoder is reduced concurrently. On the other hand, comparing Figure 3.7 (b), (c), and (d) shows that increases in bitrate also increases the runtime and platform temperature. Therefore, the constraints on both timing and bitrate will control this side effect and would be limited.

In summary, the analysis of encoder parameters on the video metrics proved the necessity of having both bitrate constraint as well as timing constraint in the application adaptation algorithm. Moreover, the results show that the complex environment and trade-offs between encoder parameters require an adaptive and dynamic solution rather than a static selection. Finally, we see that adapting the parameters to reduce run time can also reduce platform temperature, and hence can indirectly benefit the DTM reduce frequency/voltage by less amount, which in turn reduces platform performance by less amount which in turn helps the application to have higher quality as well. This effect is shown in section 3.4.3.

3.4.3 Studying the Effects of DTM Policy on Adaptation

In this section, we study in detail the effects of one of the DTM policies, and the efficiency of our proposed dynamic adaptation approach to address DTM effects. We first study the behavior of the original encoder at nominal frequency in the absence of DTM. Then we discuss the effects of DTM on the original encoder quality; and finally we show how the adaptive encoder can significantly improve speed and quality while DTM is effectively

managing the thermal impacts. We have selected profile 6, which is the recommended profile for thermal management by CoolBook, and tested it with a video clip from the movie Avatar.

Encoding a clip from Avatar in real time with original x264 video encoder in the absence of DVFS resulted in a video stream with PSNR of 41.9 dB and a bit rate of 1505 kbps (Figure 3.8(b)). However, even with the processor fan spinning at maximum speed (6200 rpm), the CPU temperature increases up to 91 degrees Celsius (Figure 3.8(a)). Our tests for different video sequences showed that processor temperatures could rise up to 94℃ just by encoding a 24 second video clip. Based on the extrapolation of the temperature curves, the CPU temperature can rise way beyond 100 degrees and its thermal limits. This shows the need for a dynamic thermal management to control the temperature. As shown by Figure 3.8(c) and (d), in the presence of DVFS (Profile 6) with trigger temperature of 75℃, the processor temperature can be contained to near 75℃ (not to exceed over 78℃), but with adverse effects of a drop in processor frequency (Figure 3.8(c)), and a very significant drop in the quality of the encoded video by about 10 dB (Figure 3.8(d)).

In contrast, Figure 3.8(e) and (f) show what occurs when the proposed adaptive encoder is used while DTM is applied. Figure 3.8(e) shows that not only the temperature is maintained at the desired level of 75°C, but also the frequency degradation caused by DVFS, is reduced compared to the original encoder without dynamic adaptation (Figure 3.8(c)). This is possible because the adaptation algorithm dynamically changes the computational need of the encoder in response to the real-time changes in the application performance index affected by DVFS, allowing the processor to cool down faster, which in turn allows DVFS to use higher frequency levels (Figure 3.2). Moreover, the adaptation algorithm not only adjusts the encoder according to the change in application performance index so as to satisfy real-time speed constraints, but also maximizes video quality with limited increase in the bit rate. This

effect is shown in Figure 3.8(f): as DTM scales down the frequency, the PSNR drops, and the bit rate rises; but the adaptation algorithm finally balances them and at the end, we have a video with PSNR of 39.2, which is very close to the original encoder's PSNR.

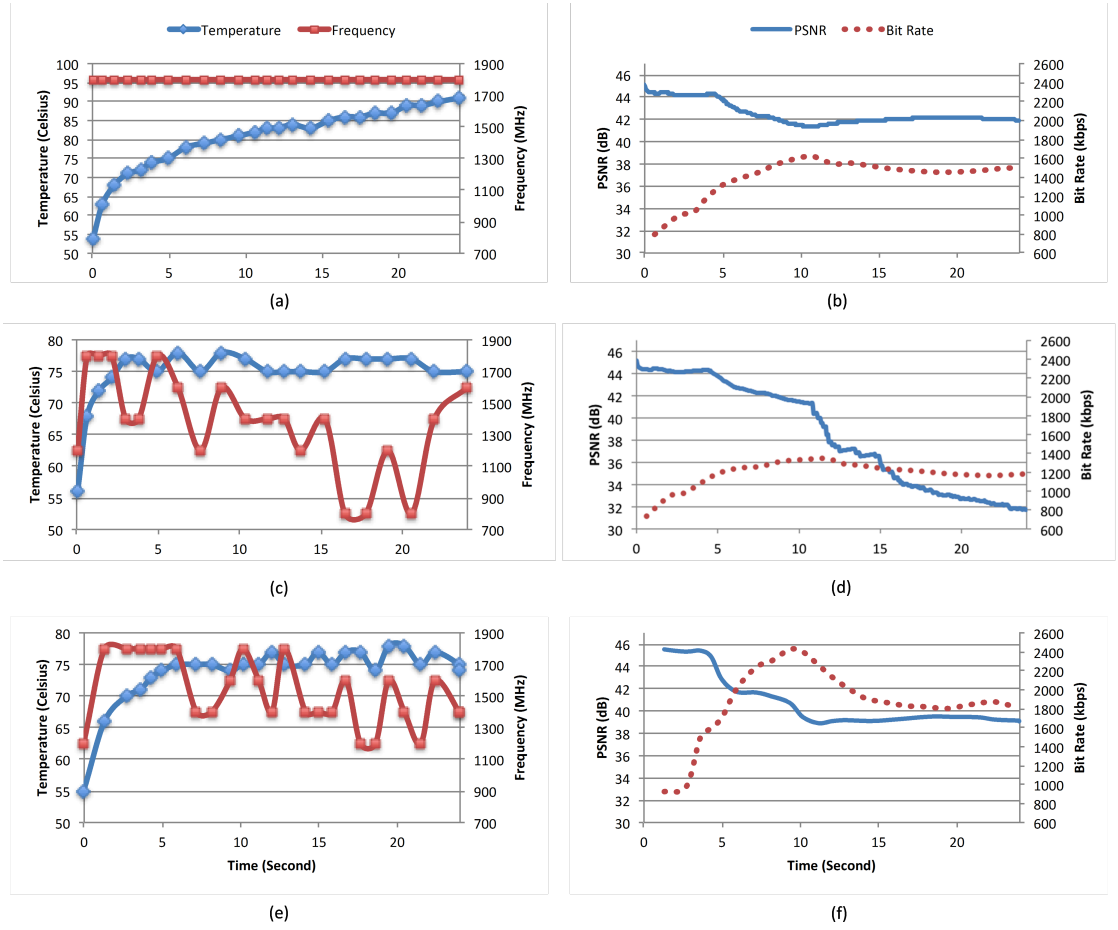


Figure 3.8 (a) Temperature and frequency of original encoder for Profile 0, (b) PSNR and bit rate of original encoder for Profile 0, (c) Temperature and frequency of original encoder for Profile 6, (d) PSNR and bit rate of original encoder for Profile 6, (e) Temperature and frequency of adaptive encoder for Profile 6, (f) PSNR and bit rate of adaptive encoder for Profile 6, while encoding Avatar video clip in real-time.

In addition we observe in Figure 3.8(f) that the initial quality of video encoder is slightly higher than the original video encoder. The initial higher PSNR is produced for two reasons. First, the video encoder initially works faster, i.e. it takes less time to encode frames; therefore, application adaptation algorithm observes the gap between the timing constraint and the encoder runtime and modifies the encoder parameters to produce better quality video. Second, the adaptation algorithm uses a history-based model whose accuracy is shown in Figure 3.4. This figure depicts the transient behavior of the model as well as its error. It is evident that the model has higher error in its first couple of seconds and then converges to the measurements. Therefore, the initial higher encoder speed and error in estimating encoder behavior causes the adaptation to choose encoding parameters that initially produces higher quality videos. After this period, encoder speed drops and the adaptation algorithm corrects the encoder parameters and the quality drops. Please note that the encoding parameters, which resulted in higher quality, could not have been used in the original encoder. It is due to the fact that the original encoder's parameters are static and after the initial period the frames begin dropping from the video sequence and would significantly degrade video quality.

In summary, without DVFS we can produce high quality video streams, but the processor temperature increases rapidly. On the other hand, having DVFS in the system helps control temperature but also drops the video quality more than 10 dB when the frequency scales down. Having the adaptive encoder and the DVFS together, the video quality does not drop more than 2 dB while processor temperature is remained in the safe region. This has come with the price of an increased bandwidth of about 12.5%.

3.4.4 Overall Adaptive Encoder Quality

In this subsection, we discuss the quality of the adaptation algorithm for three VGA size video clips Avatar, Soccer, and Wildlife. Please note that these video clips are different

from the training video clips used for extracting Q_{ref}, B_{ref}, and S_{ref}. For each video, we first encode it using the original video encoder without DVFS (Profile 0) to obtain the highest video quality and to ensure that the encoder uses all of the processing capacities. Then, we used the bit rate of those videos as the reference when encoding each specific clip with the original or adaptive video encoder in the presence of the dynamic thermal management.

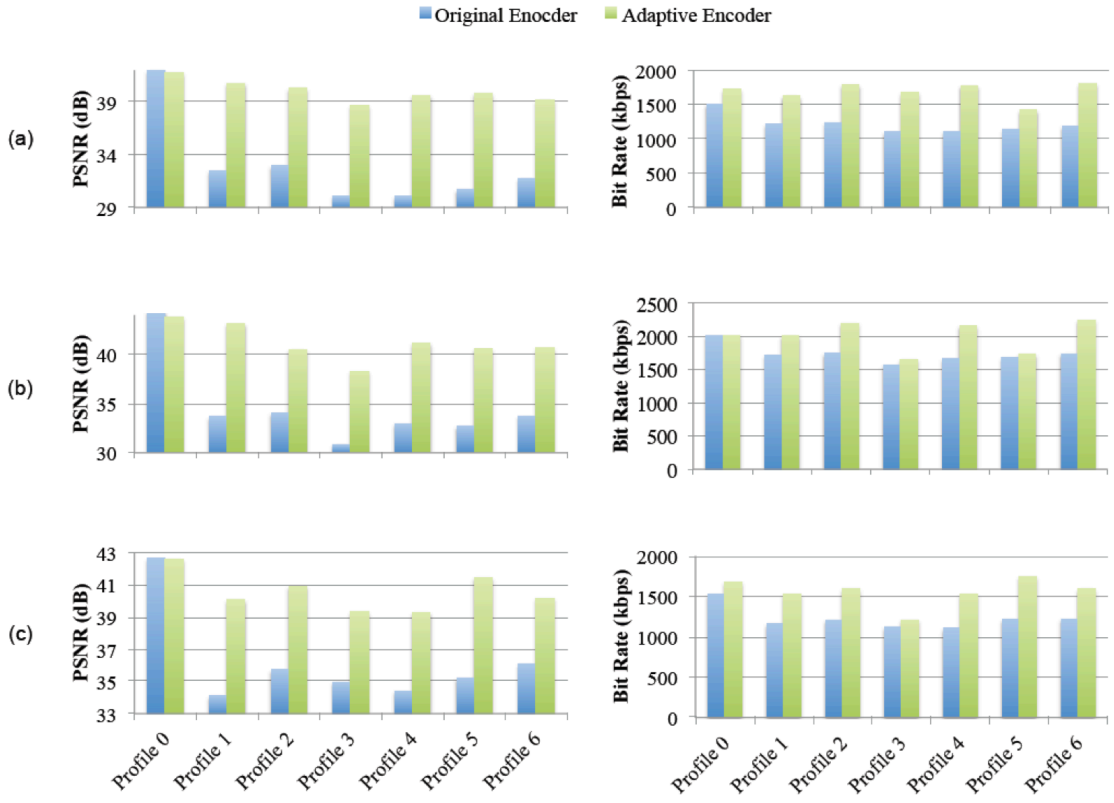


Figure 3.9 PSNR and bit rate of (a) Avatar video clip, (b) Soccer video clip, and (c) Wild Life video clip, when original and adaptive video encoder is used for real-time encoding, using DTM Profiles 0-6.

Clearly, adding the adaptation algorithm to the original encoder introduces some overhead to the encoding process. It takes some time to calculate the parameters and reconfigure the encoder and this time is taken from the total time budget available for the

encoding process. Based on our measurements, the adaptation algorithm takes 0.4 milliseconds to execute on the platform each time it is invoked. We have used 15 frames as the interval to invoke the adaptation algorithm. On the other hand, the adaptation algorithm considers this time difference and changes the encoder parameters to compensate for it. Therefore, the time (or the computation capacity) taken for the adaptation algorithm will be translated into PSNR drop and/or bit rate increase of the encoded stream. On average, the overhead due to the inclusion of the adaptation algorithm to the original encoder is equivalent to 0.26 dB drop in PSNR and 8.3% increase in bit rate.

As shown in Figure 3.9, the proposed adaptation algorithm has been able to improve video quality compared to the original encoder for all the DVFS profiles. While video quality of the original encoder drops about 10dB on average, the video quality of the adaptive encoder has dropped by an average of only 2.4 dB, with a marginal bit rate increase of about 4%.

Finally, we observe from Figure 3.9 that the best DVFS policy for the adaptive encoder is policy 2 (2 frequency/voltage levels with 12% frequency derate); where the average video quality drop has been 1.6 dB with a marginal 2.7% bit rate increase.

3.4.5 Effects of Fan Speed on Video Quality

In this section we analyze the effects of varying forced convection cooling (changing fan speed) on the quality of encoded video in a system that utilizes DTM. As mentioned earlier, additional power consumption and acoustic noise are the main drawbacks of forced convection cooling. We show that the proposed adaptation technique can be used to achieve a specified video quality using a lower fan speed and hence to reduce these drawbacks.

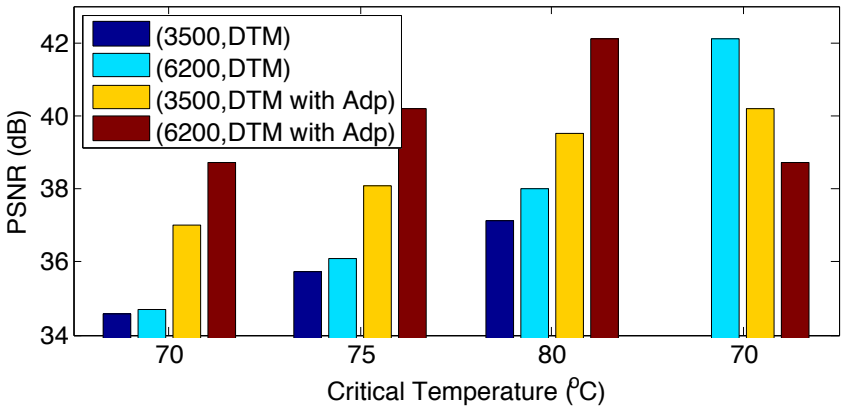


Figure 3.10 Effects of fan speed on application quality under DTM only and DTM with application adaptation (wildlife clip and DTM profile 6). The pairs mentioned in the legend of the figure are defined as fan speed in rpm during the test and thermal management method used during the test respectively. DTM means only the conventional thermal management is used and "DTM with Adp" means the proposed adaptation algorithm is used along with the DTM in a test.

The costs associated with forced convection cooling are the power consumption of fans [XYP13] and acoustic noise produced by rotating fans [ZXL12]. Unlike DTM, increasing the fan speed does not directly impact performance. On the other hand, using DTM directly affects the performance of the platform by reducing the number of active cores, frequency, etc. and hence influences the quality of application running on the platform. Therefore, the strategy that is adopted to jointly use forced convention cooling and DTM is as follows. First, since forced convection cooling does not have a direct impact on performance or application quality, one should use it as much as possible (until the fan is at the highest speed). Next, the DTM mechanism should be employed as needed to satisfy the temperature constraint. This way if cooling is enough to satisfy the thermal constraint, DTM will not kick in and there would be no negative effect on the objective function. However, cooling is not sufficient in our experimental platform (and in many systems in practice), requiring DTM to be invoked to

maintain the temperature. In all the results presented in previous sections, the fan was already at its maximum speed of 6200 rpm.

Unfortunately, as explained above, using high fan speed adds significant power consumption and noise. Therefore, it is desirable to operate the fan at lower speeds. To explore this possibility, we reduced the fan speed to 3500 rpm and evaluated the effects of increased DTM use on application quality. Figure 3.10 shows the results of using DTM profile 6 with two different fan speeds of 6200 rpm and 3500 rpm while encoding the wildlife video clip at three critical temperatures of 70, 75, and 80°C. As expected, the quality of video dropped with lower fan speeds. This is because the increased use of DTM results in lower system performance and eventually lower application quality. For example, when the critical temperature is 80°C and the fan speed drops from 6200 rpm to 3500 rpm, the video quality drops from 38 dB to 37.11dB due to lower convective heat dissipation and higher performance throttling due to the increased use of DTM.

This general trend is also true for the case when application adaption is used. For example, when the fan speed is dropped from 6200 rpm to 3500 rpm for the adaptive case, the quality of encoded video drops from 42.11dB to 39.51dB when the critical temperature is 80°C. However, an interesting observation can be made from Figure 3.10 by comparing the DTM only scenario at 6200 rpm and the DTM with adaptation scenario at 3500 rpm. For each critical temperature, the encoded video quality with adaptation at the lower fan speed (3500 rpm) is better than the quality without adaptation at even the higher fan speed (6200 rpm). In other words, if we assume that the quality of a video encoder with maximum fan speed and conventional DTM is acceptable, then by using the proposed application adaptation technique we can achieve the same or even better quality while using a significantly lower fan speed (thereby reducing system power and noise).

In summary, the analysis in this section suggests a different use of the proposed application adaptation technique: to reduce the fan speed for a given application quality constraint. An interesting avenue of future work could be an algorithm that optimizes fan speed utilizing application adaptation under given thermal and application quality constraints.

3.5 Conclusion

Due to the limitations of cooling and the increase in power density, the use of Dynamic Thermal Management in electronic systems is inevitable. However, the DTM impacts on the system's performance can be particularly problematic for compute-intensive and real-time applications, such as video encoding, which are the same applications that may need DTM the most. In this paper, we focused on H.264 video encoding as an application, and showed that the DTM can have an unacceptably high impacts on the performance and quality of the video encoding. We presented an approach that can dynamically adapt the encoder tasks in response to the DTM performance impact, such that the encoder can perform in real time, and with significantly fewer effects on the encoding quality, all with a marginal increase in the encoding bit rate. We demonstrated the approach using a commercial DVFS based DTM tool on an Intel® Core™ 2 Duo processor. In the future, we plan to investigate the applicability of the proposed approach on other compute-intensive and real-time applications like graphics rendering, and other platforms like mobile and multi-core server platforms, besides minimizing fan speed as explained in Section 3.4.4.

Acknowledgements

This chapter has been co-authored with Dr Sujit Dey and Dr Anand Raghunathan and is accepted to appear in ACM Transactions on Design Automation of Electronic Systems

(TODAES). Parts of this chapter are published on the proceeding of 2012 International Green Computing Conference (IGCC'12) as well. Authors also would like to acknowledge the discussions with Dr. Sara Zarei regarding Section 3.3.3.3.

Chapter 4

Joint Work and Voltage/Frequency Scaling for Quality-Optimized Dynamic Thermal Management

4.1 Introduction

The proposed solution in chapter 3 not only mitigated the impact of DTM on video quality but also proved to be helpful in thermal management. This interesting behavior raises the question of what if application adaptation can be used as a DTM method tool itself. The answer to this question is investigated in this chapter, and two application adaptation base DTMs are provided.

Many DTM methods have been developed for general purpose processors [SSS04; AKC09], servers and data centers [PGP10; HAC11], mobile platforms [SHK10], and embedded systems [ZC10]. Even though they vary in their approaches, a common attribute is that they all negatively affect the performance of the computing platform, leading to an increase in the runtime of tasks. In the case of many real-time applications, longer runtime result in a loss of the application's *functional quality*, which we broadly define as a metric of how well an application is performing the task that it is supposed to perform. For instance, we

define the functional quality of a video encoder as the visual quality of the video; the functional quality of a Turbo decoder is defined as its effective decoding throughput. The efficacy of previous DTM methods has been mostly quantified by the runtime increase [SCC10; KSP08; GQ11; PGP10; HAC11].

In this research we directly focus on optimizing the functional quality of a real-time application rather than its tasks' run-times. We observe that complex real-time applications such as video encoding, gaming (3D rendering), communication coding, *etc*., present parameters that can be used to tune their computing requirement, *i.e*., the workload presented to the computing platform. Based on this insight, we first introduce a DTM method called dynamic work scaling (DWS), an application level DTM technique in which we scale the computing requirements of an application by tuning one of its parameters, and hence the workload that it presents, to manage the temperature of the underlying platform. Then, we motivate and propose a hybrid method based on DWS and conventional dynamic voltage and frequency scaling (DVFS). The proposed approach is called joint dynamic work and voltage/frequency scaling (DWVFS). We formulate this hybrid DTM as a multidimensional constrained optimization problem and present an analytical solution. Furthermore, we propose a fast algorithm to solve the optimization problem in real time to perform quality-optimized DTM. Overall, the contributions of this chapter are twofold:

- Viewing DTM from the lens of its impact on an application's functional quality, and formulation of quality-optimized DTM as a constrained optimization problem.
- Proposal of a hybrid DTM technique based on joint DWS and DVFS, and demonstration of its benefits over conventional DTM techniques.

In the rest of the chapter, we first review related work on dynamic thermal management (Section 4.2). Then, we introduce DWS as a DTM method and show its utility in thermal management (Section 4.3). We then formulate quality-optimized DTM as a constrained optimization problem, discuss the relationship between work and voltage/frequency scaling with the application's quality and the platform's temperature and describe an analytical solution (Section 4.4). Next, we present a fast algorithm to perform quality-optimized DTM using DWVFS in real-time (Section 4.5). Finally, we apply the proposed DTM approach to two real-time applications – a Video encoder and a Turbo decoder – and compare its efficacy with conventional DTM methods (Section 4.6).

4.2 Related Work

A variety of techniques have been proposed for dynamic thermal management, including instruction fetch gating [KS04], dynamic voltage and frequency scaling[KS04; PUA12], scheduling [CWT09; QZW08; ZXD08; HCQ13; WB08; RV07; WX10; YBR13] and thermal aware load balancing [ZXD08]. Many hybrid methods also have been introduced to use the benefits of multiple approaches and to avoid their drawbacks [KS04; KSP06].

In [KS04] authors have used a combination of fetch gating and DVFS. When the thermal violation is small, fetch gating helps thermal management and when thermal violation increases, DVFS kicks in to control the temperature. Kumar et al. [KSP06] have used a hybrid of clock gating and thermal aware scheduling for dynamic thermal management. They assign a thermal characteristic to each running task so that the scheduler is able to reduce the priority of hot tasks. Basically, this method prevents hot tasks from running when platform's temperature rises. If this method is not effective enough, then clock gating is used to prevent more temperature rise.

Application-specific DTM methods have been extensively studied in the context of video encoding/decoding applications [GQ11; YK08; YLK07; LPP08; HV12; FS13; LPP06]. In these studies, common DTM techniques were adopted and the timing characteristics of video encoding/decoding applications were used to guarantee real-time execution.

The main difference between our work and most of the above efforts [SCC10; GQ11; YK08; LPP08; HV12; HCQ13; WB08; RV07; WX10; YBR13] arises from the objective function. In previous work, the objective has been to maximize performance in the presence of thermal constraints. Note that the term performance has been quantified by application run time, response time, whether deadlines are met, or other timing metrics of an application. Instead, we propose to look at the end user experience and maximize the functional quality that is required from an application. This matters especially in the context of real-time applications where different timing violations (e.g., dropping of different frames in a video stream) may affect the application's quality differently. In addition, we use an application layer approach compared to scheduling approaches which are OS level methods [CWT09; QZW08; HCQ13; WB08; RV07; WX10; YBR13]. This enables us to exploit application properties to design a quality-aware method, compared to other methods [CWT09; QZW08; KSP06; HCQ13; WB08; RV07; WX10; YBR13] which are oblivious to their effects on application quality and only optimize for timing requirements. Furthermore, we change the application itself to not only help with thermal management, but also improve application quality while computing capacity is limited. In contrast, other scheduler-based methods affect the order and the time in which tasks are being executed and they do not modify the application tasks themselves. Another contribution of this work is a more generic formulation and treatment of DWS for real-time applications. Previous DTM methods [GQ11; YK08; YLK07; LPP08; HV12; FS13; LPP06] that have been applied to just real-time video

encoding/decoding are dependent on application specifics. Therefore, they cannot easily be adapted to other real-time applications. In this paper we formulate the relationships between application-level functional quality and temperature on one hand and the computational requirement and computing capacity on the other, and utilize them to propose a DTM technique that is applicable to a broad range of real-time applications.

4.3 Dynamic Work Scaling: a Knob for Thermal Management

As we mentioned, many hardware and software (scheduling) methods have been developed and used for dynamic thermal management. These methods may adversely affect the quality of the application running on the system[1] by reducing the system's performance; however, they do not actually change the running application itself. In this section, we introduce a new method of thermal management by modifying an application's tasks through its parameters. This modification results in changing the computing requirements of a real-time application running on the system, and thereby changing the application's workload on the system.

There are many real-time and complex applications whose computing requirements can be controlled by their parameters. For example, quantization level for H.264 video encoder [PGS12], view distance for 3D graphics rendering [WD10], and number of decoding iterations for a Turbo decoder [SP04] in a Software Defined Radio [LMM06] can affect the computing requirements of the application. We study the effects of workload change on a

[1] The two terms *platform* and *system* have been used interchangeably in this chapter.
[2] These parameters were extracted for an Apple MacBook Air laptop with a 1.8 GHz Intel Core2Duo processor. Temperature and power have been measured in real-time while giving a step input to the system (jump from low power to high power state). The parameters were then extracted using MATLAB curve fitting tools and equation (4.2).

system's temperature in Section 4.3.1 and then we briefly discuss how changing these parameters can affect user experience and the application's functional quality.

4.3.1 DWS Concepts

In real-time applications, specific tasks should be completed in a given time interval or within a deadline. For instance, in video encoding with the frame rate of 30 frames per second, all the tasks required for encoding a frame should be completed in less than a $1/30^{th}$ of a second. In such applications, if the platform does not have enough computing capacity for the application's tasks to finish on time, the functional quality of the application would be degraded. On the other hand, if the system's computing capacity is more than the computing requirements of the application, the platform is not fully utilized. Modern computing platforms have various hardware techniques that exploit even fine-grained intervals of less-than-complete utilization for power reduction (*e.g.*, through clock gating). Note that this is orthogonal to DVFS techniques, which are applied at a coarser granularity and under software control. This reduced power dissipation under lower utilization in turn leads to lower thermal load. This characteristic of real-time applications is the foundation of the dynamic work scaling (DWS). We try to control the computing requirements of a real-time application and thereby trade-off the functional quality of the application against the thermal load that it generates.

Thermal dynamics of junction temperature in an integrated circuit can be described as follows [SAS02]:

$$\Theta' = \frac{P}{C_{th}} - \frac{\Theta}{C_{th} \cdot R_{th}} \qquad (4.1)$$

where, P is the power consumed in the silicon, C_{th} is the thermal capacitance of the silicon, R_{th} is the thermal resistance of the silicon, Θ is the silicon junction temperature relative to the ambient temperature and Θ' is the rate of change in junction temperature.

In a short time window that power is constant, the solution of equation (4.1) is:

$$\Theta(t) = R_{th}P + (\Theta_i - R_{th}P)e^{-\frac{t}{\tau}} \qquad (4.2)$$

where Θ_i is the initial temperature and $\tau = R_{th}C_{th}$ is the thermal time constant.

We formulate the periodic characteristic of real-time applications mentioned earlier as follows. The time interval in which the application needs to perform a set task is called T. For instance, T can be the required time to encode a frame in case of real time video encoding, or the time needed to decode a packet of data in the case of Turbo decoder in a wireless communication. We define the stress ratio, u, as the percentage of the time that the application is stressing the platform, where $0 \leq u \leq 1$. Therefore, the percentage of the time that system is in rest is $(1-u)$. Hence, u is equal to one when the computing requirement of the application is equal or more than the computing capacity of the platform.

We define the average power consumed during the application stress time as P_s and the average power consumed during the rest time as P_r (where $P_r < P_s$). Therefore, based on equation (4.2) the temperature follows $\Theta_s(t)$ during the stress time and $\Theta_r(t)$ while the application is at rest:

$$\begin{aligned} \Theta_s(t) &= R_{th}P_s + \left(\Theta_{i_s} - R_{th}P_S\right)e^{-\frac{t}{\tau}} \\ \Theta_r(t) &= R_{th}P_r + \left(\Theta_{i_r} - R_{th}P_r\right)e^{-\frac{t}{\tau}} \end{aligned} \qquad (4.3)$$

Platform temperature increases during application stress time. During rest time, platform gets the chance to release thermal energy and the temperature drops. Figure 4.1 displays this pattern over time for $u = 0.75$ and compares it with $u = 1$, for a platform with $R_{th} = 1.46\ C/W$, $C_{th} = 41.1\ J/C$, $P_s = 65\ W$, $P_r = 13\ W$[2]. Figure 4.1 illustrates that when the application leaves the platform in rest, the temperature drops but the amount of change in the temperature depends on the platform's instantaneous temperature. Figure 4.1 suggests that the temperature of the platform reaches a steady condition. We are interested to know how this thermal steady condition is related to the stress ratio, and then use it for thermal management of the platform.

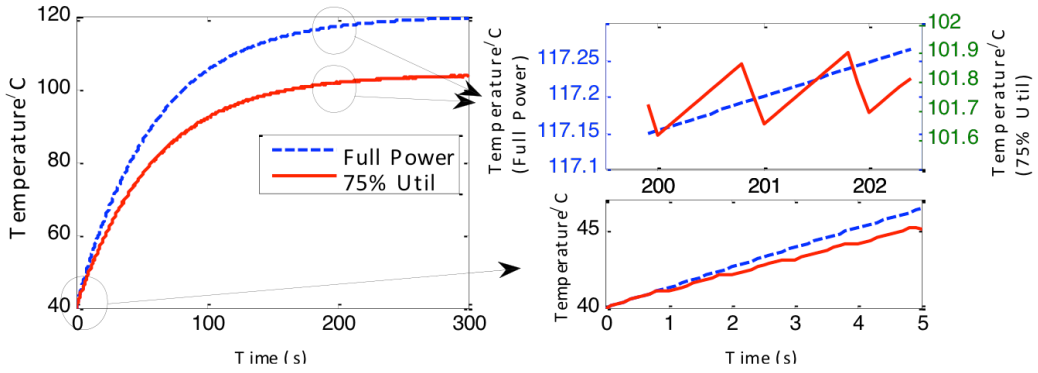


Figure 4.1 Thermal behavior of a platform based on stress ratio of 100% and 75%.

The system goes to steady thermal condition when the thermal behavior repeats itself after each time interval T. This condition can be mathematically described as follows:

$$\Theta_s(uT) = \Theta_{i_r}$$
$$\Theta_r\big((1-u)T\big) = \Theta_{i_s} \qquad (4.4)$$

[2] These parameters were extracted for an Apple MacBook Air laptop with a 1.8 GHz Intel Core2Duo processor. Temperature and power have been measured in real-time while giving a step input to the system (jump from low power to high power state). The parameters were then extracted using MATLAB curve fitting tools and equation (4.2).

By solving equation (4.3) with the conditions in equation (4.4) we derive that in the steady condition, temperature will oscillate between the two values of Θ_{min} and Θ_{max} with the period of T.

$$\Theta_{min} = R_{th}\left(P_r\left(\frac{e^{\frac{T}{\tau}} - e^{\frac{uT}{\tau}}}{e^{\frac{T}{\tau}} - 1}\right) + P_s\left(\frac{e^{\frac{uT}{\tau}} - 1}{e^{\frac{T}{\tau}} - 1}\right)\right)$$
$$\Theta_{max} = R_{th}\left(P_r\left(\frac{e^{\frac{-uT}{\tau}} - e^{\frac{T}{\tau}}}{1 - e^{\frac{-T}{\tau}}}\right) + P_s\left(\frac{1 - e^{\frac{-uT}{\tau}}}{1 - e^{\frac{-T}{\tau}}}\right)\right) \quad (4.5)$$

In practice, the time interval T for real-time applications is in the order of microseconds or milliseconds; however the thermal time constant is usually in the order of seconds ($T \ll \tau$). By using Taylor's polynomial approximation, equation (4.5) simplifies to the following:

$$\Theta_{min} = \Theta_{max} = \Theta_{ss}(u) = R_{th}(P_r(1-u) + P_s u) \quad (4.6)$$

Equation (4.6) concludes the relationship between the stress ratio and the steady temperature of the platform. It states that there is a linear relationship between the stress ratio and the steady temperature of the platform assuming the thermal time constant is large enough compared to the real-time application's time interval T.

In summary, the parameters of a real-time application can be used to change its computing requirements and hence its work load. Thereby, the portion of time that the application stresses the platform can be affected. Furthermore, the steady temperature of the platform is a linear function of the stress ratio. This means by controlling the application's

workload through changing its parameters, one can control and manage the temperature of the platform (dynamic thermal management).

4.3.2 DWS Implementation

As we mentioned, work scaling is a potential control knob for thermal management. In this section, we define the objective of DWS as a DTM technique, and by using a greedy implementation will show its effectiveness.

Since changing the work load of a real-time application by scaling its computational requirements/complexity can also affect its functional quality, we define the objective of DWS with respect to application function quality as follows:

> ***The objective of DWS for thermal management is to maximize the functional quality of the real-time application while making sure the temperature of the platform does not exceed a given critical temperature.***

The above statement can be described in the following mathematical form:

$$\text{Maximize: } Q(p)$$
$$\text{Subject to: } \Theta(p) \leq \Theta_c \qquad (4.7)$$

where Q is the application's functional quality, and Θ is the platform's temperature, as a function of the application's parameter p. Therefore, the control variable in the DWS is the application parameter that would vary application computing requirement and workload.

DVFS:	DWS:
Every 100 millisecond:	Every 100 millisecond:
Θ= Platform temperature	Θ= Platform temperature
If $\Theta > \Theta_c$: Decrease frequency and voltage one step	If $\Theta > \Theta_c$: Decrease computing requirements one step
Else: Increase frequency and voltage one step	Else : Increase computing requirements one step

Figure 4.2 Greedy implementation of DVFS and DWS.

Figure 4.2 shows a greedy implementation of DWS and DVFS for DTM. Both DWS and DVFS algorithms will control platform temperature (perform DT M) but they use different control variables. We have used the greedy algorithms on two applications, H.264 video encoder and Turbo decoder. In the case of the H.264 video encoder, the application parameter p is constant rate factor (CRF), and the functional quality Q is the encoded video stream's peak signal to noise ratio (PSNR). The number of iterations in the decoding loop is the application parameter for Turbo decoder and its effective throughput is its functional quality. The details and description of these applications, their parameters and their functional quality along with the platform specifications are provided in the results section (Section 4.6).

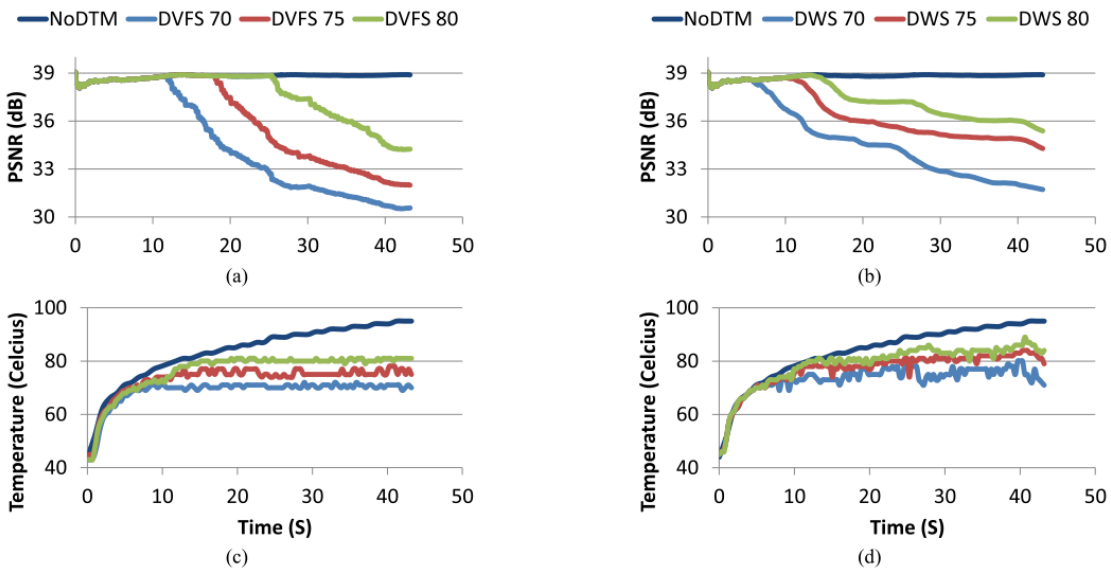


Figure 4.3 The effects of DWS and DVFS on an H.264 video encoder application's functional quality and platform's temperature(a) Visual quality for DVFS, (b) Visual quality for DWS, (c) Platform temperature with DVFS, (d) Platform temperature with DWS.

Figure 4.3 compares the results of DWS and DVFS implementations described in Figure 4.2 for four test conditions. One test is without any DTM and the other three tests are with 70°C, 75°C, and 80°C as critical temperature. Figure 4.3(a) and (b) show the visual quality of the encoded stream (in PSNR) over time for DVFS and DWS respectively. It is clear that in this example, quality drop in DVFS is faster than DWS and eventually, the video stream produced under DWS has a better quality (higher PSNR). On the other hand, from Figure 4.3(c) and (d) we observe that DVFS controls the temperature better than DWS meaning the violation of critical temperature occurs less frequently.

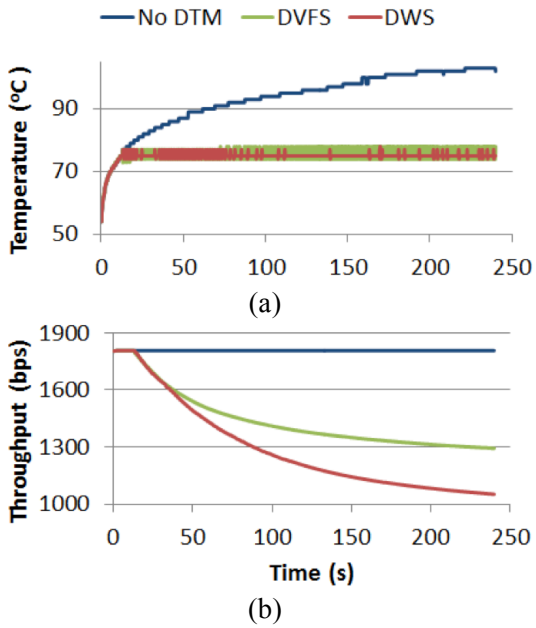


Figure 4.4 Using DVFS and DWS for Turbo decoder application. (a) Platform temperature in degree Celsius. (b) Turbo decoder throughput in bps.

Figure 4.4 shows the result of using DWS and DVFS for a Turbo decoder application for a critical temperature of 75 degree Celsius. Figure 4.4(a) shows that both DWS and DVFS can control the temperature around the target temperature while without DTM, the

temperature rises above 100 degree Celsius. Figure 4.4(b) shows the effects of each DTM method on the functional quality which in this case is decoder's effective throughput. In this scenario, the negative effect of DWS on application quality is more than DVFS.

These examples are provided as proofs of concept to show that DWS can be used as a DTM tool in practice. In addition, in the case of video encoder, DWS produces better quality results than DVFS while in the case of Turbo decoder, DVFS produces better functional quality. The results tell us that while both DWS and DVFS can be used for DTM, their effectiveness and impact on application quality can be different under different application conditions. We investigate these conditions in the rest of this paper and study the joint effects of DWS and DVFS on both applications' functional quality and platform temperature.

4.4 Quality Optimized Dynamic Thermal Management

Our goal is to provide a method that best utilizes both DWS and DVFS in order to obtain a quality optimized DTM methodology. To achieve this goal, we first study the joint effects of DWS and DVFS on temperature as well as functional quality of a real-time application. Next, we formulate quality-optimized DTM in the form of a constrained optimization problem, whose solution is the new joint dynamic work and voltage/frequency scaling (DWVFS) approach for DTM.

4.4.1 Quality and Thermal Contour Lines

We start with some definitions that will help in quantifying different behaviors of any real-time application in general terms, and eventually develop a generic DTM algorithm. Changing a platform's voltage/frequency affects the computing capacity of the platform.

Therefore we define our first variable as the Computing Capacity or C_c which represents the effect of DVFS on the platform. On the other hand, when we change a parameter of an application, we affect a change in the amount of computation requested from the platform. Therefore we define our second variable as the application's Computing Requirements or C_r to represent the effect of DWS. Hence the application's functional quality and platform's temperature can be describes as functions of these two variables:

$$\text{Application functional quality: } Q(C_c, C_r)$$
$$\text{Platform temperature: } \Theta(C_c. C_r) \tag{4.8}$$

Since both $Q(C_c, C_r)$ and $\Theta(C_c, C_r)$ are functions of two variables, we use contour lines to study their general behavior. A contour line (or isoline) of a function with two variable, is a curve in which the value of the function is the same [CR96]. Contour lines can be used to show the general behavior of a two-variable function like its extremums or rate of change. For example, Figure 4.5 shows the contour lines plots of $Q(C_c, C_r)$ and $\Theta(C_c, C_r)$ for an H.264 video encoder application. The horizontal axis is the CPU frequency; as frequency increases the computing capacity of the platform increases. The vertical axis is the CRF which is one of the H.264 video encoder's parameters. The increase in CRF decreases the computing requirements of the encoder (More information about CRF is provided in Section 4.6.6). These plots illustrate that peak quality and temperature occur at minimum CRF and maximum CPU frequency. We first derive the general shape of temperature contour lines and then study functional quality contour lines. Then, we will use them in the next subsection to formulate the quality optimized dynamic thermal management.

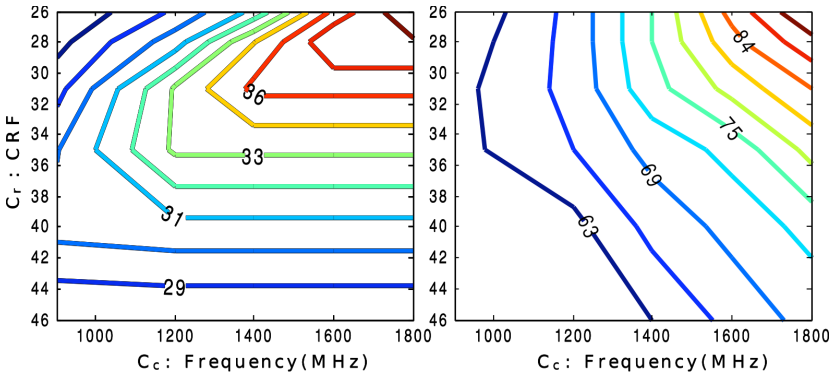


Figure 4.5 Contour level plots of an H.264 encoder on Intel Core2Duo, (a) functional quality in PSNR, (b) platform temperature in degree Celsius.

4.4.1.1 Temperature Function

As we mentioned earlier, platform temperature is a function of C_c and C_r. In addition, it is self-evident that increasing computing requirements of application increases the platform utilization as long as computing capacity is available:

$$\begin{cases} \frac{\partial u}{\partial C_r} = 0 & C_r \geq C_c \\ \frac{\partial u}{\partial C_r} > 0 & C_r < C_c \end{cases} \tag{4.9}$$

From equation (4.6) (Section 4.3.1) and equation (4.9) we derive that by increasing computing requirements while computing capacity is available, the temperature increases:

$$\begin{cases} \frac{\partial \Theta}{\partial C_r} = 0 & C_r \geq C_c \\ \frac{\partial \Theta}{\partial C_r} > 0 & C_r < C_c \end{cases} \tag{4.10}$$

The same type of relationship holds between temperature and computing capacity. Computing capacity is proportional to platform's frequency and the platform's temperature increases by increasing its frequency and voltage:

$$\frac{\partial \Theta}{\partial C_c} > 0 \qquad (4.11)$$

Equation (4.10) and equation (4.11) provide us with enough information to find out the general thermal behavior of a platform. In fact, the contour lines characteristics can be extracted from the function's partial derivatives as described in the following corollary:

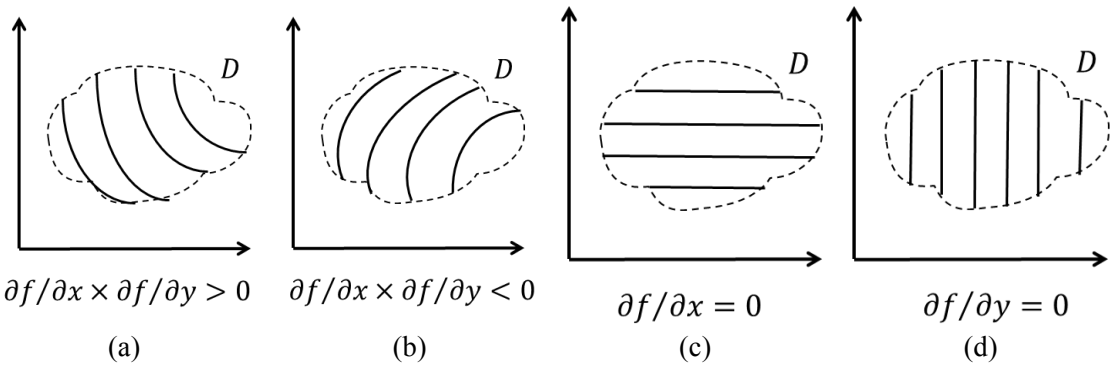


Figure 4.6 General shapes of contour lines with respect to the function's partial depravities with the conditions specified in Corollary (a) I, (b) II, (c) III, (d) IV.

Corollary. The contour lines plot of a two-variable function $f(x, y)$ in a domain (D), where the sign of the partial derivatives does not change and at least one of them is non-zero, will be (Figure 4.6):

I. (Theorem A1 & A2) Non-crossing decreasing lines if $\partial f/\partial x \times \partial f/\partial y > 0$

II. (Theorem A1 & A2) Non-crossing increasing lines if $\partial f/\partial x \times \partial f/\partial y < 0$

III. (Theorem A1) Parallel horizontal lines if $\partial f/\partial x = 0$

IV. (Theorem A1) Parallel vertical lines if $\partial f/\partial y = 0$

We provide proofs of Theorems A1 and A2 in the Appendix. By applying the above corollary on equation (4.10) and equation (4.11), we come up with the general thermal behavior of a real-time application. We define the maximum computing capacity of the platform as C_M. Therefore the domain of temperature function D is defined as:

$$D = \{(C_c, C_r) | 0 \leq C_c \leq C_M, 0 \leq C_r \leq C_M\} \tag{4.12}$$

This domain does not fall into any of the categories mentioned in the corollary. Therefore we break it into two sub-domains:

$$D_1 = \{(C_c, C_r) | 0 \leq C_c \leq C_M, 0 \leq C_r < C_c\};$$
$$D_2 = D - D_1 \tag{4.13}$$

The first domain, D_1, follows Corollary I formulation and the second domain, D_2, follows Corollary IV formulation (equation (4.10) and equation (4.11)). Figure 4.7 displays the general shape of the temperature contour lines of a real-time application where the contour lines in D_1 and D_2 are made from Figure 4.6, Corollary I and IV plots.

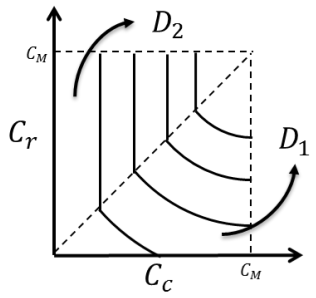


Figure 4.7 The contour lines plot of platform's temperature executing a real-time application with respect to computing capacity and computing requirements. The diagonal dashed-line represents $C_r = C_c$.

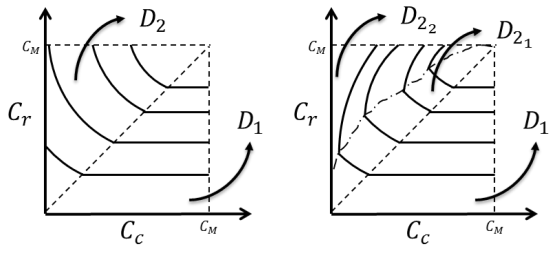
4.4.1.2 Quality Function

We start analysis of the quality function, $Q(C_c, C_r)$, with the sub-domain defined in equation (4.13) , D_1. In this subdomain, the computing requirement is smaller than computing capacity of the platform. Therefore, a slight change in the computing capacity will not affect the quality of the application. This phenomenon can be represented mathematically as:

$$\frac{\partial Q}{\partial C_c} = 0 \,, \quad \text{if: } C_r < C_c \tag{4.14}$$

Using Corollary III, we conclude that the contour lines of the quality function are horizontal lines in D_1 (Figure 4.8). Next we study the D_2 domain where the computing capacity is less than computing requirements. Let's consider an arbitrary point in this domain. Since the computing capacity is less than the computing requirements, the functional quality of application would be less than expected. Now, if the computing capacity drops, the functional quality suffers even more due to higher limitations induced by the platform. Therefore we can derive:

$$\frac{\partial Q}{\partial C_c} > 0 \,, \quad \text{if: } C_r > C_c \tag{4.15}$$



(a) (b)

Figure 4.8 The contour lines plots of a real-time application's quality function Q with respect to computing capacity and computing requirements. The diagonal dashed line represents $C_r = C_c$. (a) Quality contour lines plot when $\partial Q/\partial C_r > 0$ for all the points in D_2. (b) Quality contour lines plot when $\partial Q/\partial C_r$ changes sign in D_2. The dashed-dot line represents these points ($C_r = f(C_c)$).

Now we take a look at computing requirements of an application and its role on the application's quality. An application is usually designed in a way that the increase in computing requirements is going to increase the application's functional quality unless it faces any limitations from the platform. It means:

$$\frac{\partial Q}{\partial C_r} > 0\,, \quad \text{if: } C_r < C_c \tag{4.16}$$

When $C_r > C_c$, the effects of C_r change on quality function would be complex. One would expect to see higher quality in response to C_r increase; however, due to limited computing capacity, not only we may not achieve as much quality increase as we expected, but we may also lose quality due to application tasks that may not be completed. We know the rate of quality change with respect to computing requirement is positive right before entering D_2 (equation (4.16). Therefore, $\partial Q/\partial C_r$ in D_2 is either always positive, or it starts positive and changes to negative at some point in D_2. We show the points where the derivative sign changes with the following notation:

$$C_r = f(C_c) \qquad \text{for } (C_c, C_r) \in D_2 \tag{4.17}$$

In summary, there would be two scenarios. First scenario is when $\partial Q/\partial C_r > 0$ for all the points in D_2(equation (4.19)). Using equation (4.15) and Corollary I, we conclude that the general shape of quality contour lines would be in the form of Figure 4.8(a) for this scenario. The second scenario would be if the sign of $\partial Q/\partial C_r$ changes from positive to negative at some points in D_2 described by equation (4.17). Therefore, we divide D_2 into two smaller subdomains based on equation (4.17):

$$D_{2_1} = \{(C_c, C_r) | 0 \le C_c \le C_M, C_c < C_r < f(C_c)\} \tag{4.18}$$

$$D_{2_2} = D_2 - D_{2_1}$$

Accordingly, the application quality change with respect to computing capacity will be equation (4.20):

$$\frac{\partial Q}{\partial C_r} > 0 \quad \text{while } C_c < C_r < C_M \tag{4.19}$$

$$\begin{cases} \frac{\partial Q}{\partial C_r} > 0 & \text{while } C_c < C_r < f(C_c) \\ \frac{\partial Q}{\partial C_r} < 0 & \text{while } f(C_c) < C_r < C_M \end{cases} \tag{4.20}$$

Using equation (4.15), Corollary I, and Corollary II, we conclude that the general shape of quality contour lines would be in the form of Figure 4.8(b) for the second scenario. Next, we formulate quality optimized DTM into an optimization problem and solve it using the contour plots we introduced in this section.

4.4.2 Quality and Thermal Management: Problem Formulation

As we mentioned earlier, we are interested to come up with a quality optimized DTM based on joint dynamic work and voltage/frequency scaling (DWVFS). The objective of DWVFS for DTM can be described in the following statement:

> ***The objective of DWVFS for thermal management is to maximize the functional quality of the real-time application while making sure the temperature of the platform does not exceed a given critical temperature.***

The above statement can be described in the following mathematical form:

$$\begin{aligned} &\text{Maximize: } Q(C_c, C_r) \\ &\text{Subject to: } \Theta(C_c, C_r) \leq \Theta_c \end{aligned} \tag{4.21}$$

In the rest of this section, we derive a solution for the above optimization problem.

Lemma 1. The solution of the optimization problem is located on the critical temperature contour line:

$$\Theta(C_c, C_r) = \Theta_{c,} \tag{4.22}$$

Proof. From Figure 4.7 and Figure 4.8, we know that both functional quality and temperature functions will be at their peak values when $(C_c, C_r) = (C_M, C_M)$. In addition, all of the points between (C_M, C_M) and critical temperature contour line (points in the red area in Figure 4.9) violate the thermal constraint and cannot be part of the solution. From equation (4.14) and equation (4.16), we know that for any point in D_1 and the blue area in Figure 4.9, there is another point on the θ_c contour line with a higher quality. Also from equation (4.15) we know that for any point in D_2 and the blue area in Figure 4.9, there is another point on the θ_c contour line with a higher quality. Therefore, the solution of the equation (4.21) is located on the critical temperature contour line. ■

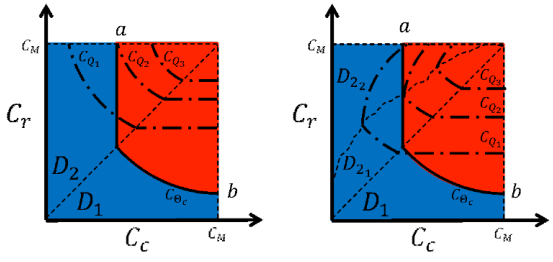


Figure 4.9 Superimposition of temperature and quality contour lines plots. Solid line is the contour plot of the platform's temperature at critical temperature. The dashed-dot lines are functional quality contour lines. The points in the red area will violate the thermal constraint and conversely, the points in the blue area will not violate the thermal constraint. (a) When $\partial \mathrm{Q}/\partial \mathrm{C_r}\,(\mathrm{C_c}, \mathrm{C_r})$ does not change sign. (b) When $\partial \mathrm{Q}/\partial \mathrm{C_r}\,\big(\mathrm{C_c}, \mathrm{f(C_c)}\big)$ changes sign.

In the next step, we prove Lemma 2 which shows the solution of the optimization problem (equation (4.21)) is not in the D_1 domain.

Lemma 2. For any point$(C_{c_0}, C_{r_0}) \in D_1$, on the contour line $\Theta(C_c, C_r) = \Theta_c$, there exist another point $(C_{c_1}, C_{r_1}) \in D_1$ on the contour line $\Theta(C_c, C_r) = \Theta_c$, which is closer to the D_1 boundary $C_r = C_c$ and $Q(C_{c_1}, C_{r_1}) > Q(C_{c_0}, C_{r_0})$.

Proof. We choose $C_{c_1} < C_{c_0}$ on the contour line $\Theta(C_c, C_r) = \Theta_c$. Since temperature contour line's slope is negative (Theorem A1, equation (4.10) and equation (4.11)) we derive: $C_{r_1} > C_{r_0}$. Therefore: $C_{c_1} - C_{c_0} < 0 < C_{r_1} - C_{r_0} \xrightarrow{\text{Both points} \in D_1} \frac{|C_{c_1} - C_{r_1}|}{\sqrt{2}} < \frac{|C_{c_0} - C_{r_0}|}{\sqrt{2}}$, where both sides of the inequality are the distance formula of a point from a line [DD09]. It means (C_{c_1}, C_{r_1}) is closer than (C_{c_0}, C_{r_0}) to $C_r = C_c$.

From above, equation (4.14) and equation (4.16) we derive: $Q(C_{c_1}, C_{r_1}) = Q(C_{c_0}, C_{r_1}) > Q(C_{c_0}, C_{r_0})$■

Lemma 2 tells us for any point to be assumed as the solution of equation (4.21) in D_1, there will be another point with better quality, which contradicts the initial assumption. Therefore the solution of equation (4.21) is not in D_1. To identify the solution in D_2, we first analyze applications with general contour lines of Figure 4.8(a) where $\partial Q / \partial C_r > 0$ for all points in D_2.

Theorem 1. For a real-time application where $\partial Q / \partial C_r > 0$, the solution to the optimization problem (equation (4.21)) is the following point:

$$(C_c, C_r): C_r = C_M \text{ and } \Theta(C_c, C_M) = \Theta_c \tag{4.23}$$

Proof. Let's consider an arbitrary point $(C_{c_0}, C_{r_0}) \in D_2$ on the temperature contour line $\Theta(C_c, C_r) = \Theta_c$. We then choose another point $(C_{c_0}, C_{r_1}) \in D_2$ where $C_{r_1} > C_{r_0}$. It is clear that the second point is also on the same temperature contour line (equation (4.10)). From equation (4.19), we know $Q(C_{c_0}, C_{r_1}) > Q(C_{c_0}, C_{r_0})$. This means for any point in D_2 that is on

the temperature contour line $\Theta(C_c, C_r) = \Theta_c$, there is always another point on the same contour line with higher quality and shorter distance to C_M. Per Lemma 2, we know the solution is within D_2. Therefore the maximum quality occurs when $C_r = C_M$. ■

In addition to the above formal proof, a visual explanation helps with better understanding the solution. The solution given in Theorem 1 can also be achieved by superimposing the two contour lines plots of quality and temperature as shown in Figure 4.9(a), where C_Q and C_Θ curves are quality and temperature contours respectively. It is evident that $Q(C_{Q_3}) > Q(C_{Q_2}) > Q(C_{Q_1})$. However, none of the points on the contour line C_{Q_3} can be used for equation (4.21) solution due to lying in the red area and violating the thermal constraint. When walking across the quality contour lines from C_{Q_3} to C_{Q_1} the quality drops. Therefore, the solution of equation (4.21) can be found when the first quality contour line intersects with the critical temperature contour line. This occurs with C_{Q_2} contour line. As it is shown in the plot, the first point in which a quality contour line intersects with critical temperature contour line is when $C_r = C_M$. This point is the same solution point given in Theorem 1.

Theorem 2. For a real-time application where $\partial Q / \partial C_r$ changes sign at the points with $C_r = f(C_c)$, the solution to the optimization problem (equation (4.21)) is the following:

$$(C_c, C_r): \Theta(C_c, f(C_c)) = \Theta_c \text{ and } C_r = f(C_c) \tag{4.24}$$

Proof. Per Lemma 2, we know the solution is not in D_1. In addition, based on Corollary IV, the temperature contour line $\Theta(C_c, C_r) = \Theta_c$ is a vertical line that can be shown as $C_c = C_{c_0}$. We extract a single-variable function from the quality function when $C_c = C_{c_0}$ in the following way:

$$Q_c(C_r) = Q\left(C_{c_0}, C_r\right) \Rightarrow \frac{dQ_c}{dC_r} = \frac{\partial Q}{\partial C_r}\Big|_{C_c = C_{c_0}} \tag{4.25}$$

From equation (4.20) and equation (4.25) we derive:

$$\begin{cases} dQ_c/dC_r > 0, & C_r < f\left(C_{c_0}\right) \\ dQ_c/dC_r < 0, & C_r > f\left(C_{c_0}\right) \end{cases} \tag{4.26}$$

From basic calculus we know that equation (4.26) is the definition of local maximum for $Q_c(C_r)$ at $C_r = f\left(C_{c_0}\right)$. ■

Figure 4.9(b) helps to visually understand the proof of Theorem 2. In fact the visual proof of Theorem 2 follows the exact same steps of proof of Theorem 1. The solution is at the intersection of the first quality contour line with the critical temperature contour line, when walking across quality contour lines from C_{Q_3} towards C_{Q_1}. From Figure 4.9(b), we see this point is located on the border of D_{2_1} and D_{2_2} which is $C_r = f(C_c)$.

In this section, we formulated DWVFS as a constrained optimization problem (equation (4.21)) and analytically characterized its solution. However, the derivation was performed under the assumption that quality and temperature, and the parameters they depend upon, are all continuous variables, which is not true in practice. Furthermore, even overlooking this assumption, equation (4.17) and equation (4.22) need to be solved numerically in real-time to perform DTM decisions. Therefore, in the following section, we propose an efficient algorithm to solve the optimization problem described in equation (4.21).

4.5 Joint Dynamic Work and Voltage/Frequency Scaling

In this section we introduce a fast and real time method for DWVFS based on the solution of optimization problem described in equation (4.21). Our method is a combination of offline steps and online steps. Figure 4.10 displays the block diagram of the proposed method. In the offline steps, we collect the necessary data to develop three models. The first model captures the relationship of application's functional quality with respect to the application parameter and the platform voltage/frequency. The second model describes the relationship between the application parameter and its work load. The last model is used to model platform's temperature with respect to the application's work load and platform's voltage/frequency. After a brief description of these models, we introduce the algorithm which finds the solution for equation (4.21), thereby maximizing application quality while performing thermal management.

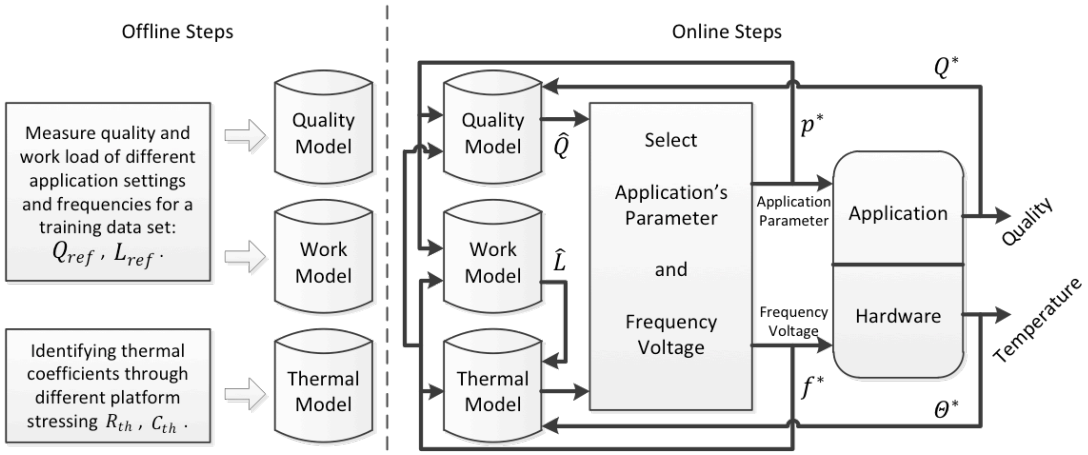


Figure 4.10 Joint work and voltage/frequency scaling block diagram showing offline and online steps.

4.5.1 Offline Models

4.5.1.1 Quality/Work Model

We model work load, $\hat{L}$, and functional quality, $\hat{Q}$, as functions of two variables, application parameter p and platform frequency f, in the following way:

$$\begin{gathered} \boldsymbol{D} = \{(p,f) | p \in \boldsymbol{P}, f \in \boldsymbol{F}\} \\ \hat{L}: \boldsymbol{D} \to \mathbb{R}^+ \\ \hat{Q}: \boldsymbol{D} \to \mathbb{R}^+ \end{gathered} \tag{4.27}$$

$\boldsymbol{P}$ is the set of possible values for the application parameter and $\boldsymbol{F}$ is the set of possible frequencies the platform can use.

We measure workload and functional quality of the application using a set of training data for all the points in $\boldsymbol{D}$. The results of these measurements are populated in two lookup tables (LUT) used for reference: $L_{ref}(p,f), Q_{ref}(p,f)$, where $(p,f) \in \boldsymbol{D}$.

The model uses these reference LUTs and linearly scales them based on the measurements of the functional quality, Q^*, and work load, L^*, of the application during execution time:

$$\begin{aligned} \hat{L}(p,f) &= \frac{L^*}{L_{ref}(p^*,f^*)} \cdot L_{ref}(p,f) \\ \hat{Q}(p,f) &= \frac{Q^*}{Q_{ref}(p^*,f^*)} \cdot Q_{ref}(p,f) \end{aligned} \tag{4.28}$$

Where p^* and f^* are application parameter and platform frequency used in the last measurement of Q^* and L^*. Equation (4.28) estimates load ($\hat{L}$) and quality ($\hat{Q}$) for any p and f in $\boldsymbol{D}$ based on the measured L^* and Q^* for p^* and f^*. We validate the accuracy of the above

models by applying them on the H.264 video encoder, whose quality function can be challenging to model as it is highly dependent on individual video streams and their content. The application parameter used is CRF and the functional quality metric is PSNR. We collected the reference LUT from a set of training video streams. Then we applied the reference LUT for the modeling of three other test video streams. Figure 4.11 shows the % error of the Quality and Load values estimated by the models from the actual measurements, which are only 4% and 2% respectively, demonstrating accuracy of the models. Please note that the DWVFS algorithm proposed later is independent of how these models are developed.

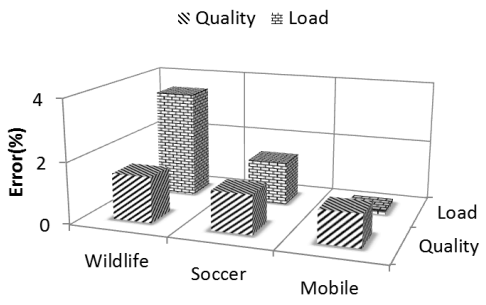


Figure 4.11 Accuracy of functional quality and work load models for three test videos of Wild life, Soccer and Mobile sequence.

4.5.1.2 Thermal Model

In this subsection, we describe our assumptions for thermal model that is incorporated in DWVFS. We discussed the thermal formula in equation (4.2) for a time period in which the power consumption is constant. Silicon power consumption is primarily composed of two components: dynamic, P_{dyn} and static P_{sta};

$$P = P_{dyn} + P_{sta} \tag{4.29}$$

The dynamic power is proportional to the hardware work load L as well as its frequency and voltage squared:

$$P_{dyn} \propto V^2 f L \tag{4.30}$$

The static power is proportional to the platform voltage and leakage current where the leakage current itself increases exponentially with increase in temperature [BS00].

$$P_{dyn} \propto V^2 f L \tag{4.31}$$

where α is the exponential proportionality coefficient and Θ is the temperature in degree Celsius. Therefore, based on equation (4.2) and equation (4.29) through equation (4.31), we get the following formula for estimating the junction temperature of the platform after a time interval of Δt with initial temperature of Θ^* and the ambient temperature of Θ_a:

$$\hat{\Theta}(\Delta t) = \Theta^* + \left(\Theta_a - \Theta^* + K_1 V^2 f L + K_2 V e^{\alpha \Theta^*}\right)\left(1 - e^{-\frac{\Delta t}{\tau}}\right) \tag{4.32}$$

The model provided in equation (4.32) follows the same characteristics mentioned in Section 4.4 for thermal contour lines (Figure 4.7). When $C_r < C_c$ (L is variable) then the contour lines are decreasing and when $C_r > C_c$ (L is constant at 100%) the contour lines are vertical. The next step is to find the coefficients of the model described in equation (4.32) for a given platform. These coefficients are extracted for a MacBook Air (test platform) by stressing it in different conditions and using curve fitting methods. Figure 4.12 displays the plots of the model provided in equation (4.32) and the measured temperature of this test platform in different operating conditions. Each curve on the plot corresponds to different frequency/voltages of the platform where these levels are predefined by manufacturer. The mean squared root errors (MSRE) of the model vs. measurements are about 1 degree Celsius.

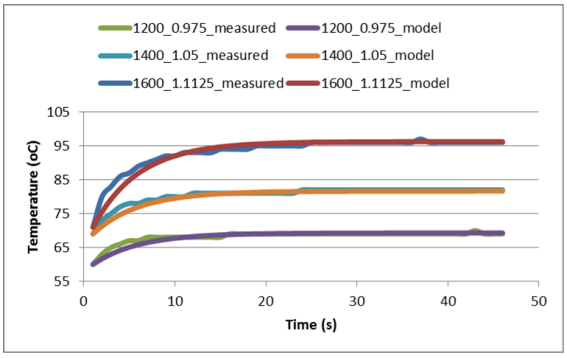


Figure 4.12 Measured and modeled junction temperature of MacBook Air platform with Core2 Duo processor at 100% work load.

4.5.2 Online DWVFS Steps

Figure 4.13 shows the pseudo code of the online steps for the joint dynamic work and voltage/frequency scaling method which involves selecting the application's parameter and platform's voltage/frequency. The following paragraphs describe the proposed algorithm.

The algorithm selects the application parameter from a set $\boldsymbol{P} = \{p_1, p_2, \dots, p_n\}$, sorted in decreasing order of computing requirement ($C_r(p_1) > C_r(p_2) > \cdots > C_r(p_n)$). The frequency is also selected from a set $\boldsymbol{F} = \{f_1, f_2, \dots, f_m\}$ sorted in increasing order ($f_1 < f_2 < \cdots < f_m$). The outputs of the DWVFS algorithm are the new application parameter p, and platform's frequency f. The voltage of the platform is predefined for each frequency level.

The proposed algorithm is a combination of two nested searches in the directions of computing capacity and computing requirement (horizontal and vertical axes respectively in Figure 4.9). We have mentioned that the temperature of the platform is an increasing function with respect to its computing capacity (equation (4.11)). We use this property and run a binary search in the direction of computing capacity over the temperature function, $\hat{\theta}(C_c, C_r)$, as the inner search loop. In fact, the inner loop is based on Lemma 1 and the algorithm uses

the thermal model to identify the points that are closest to the critical temperature contour line ($\hat{\theta}(\hat{p},\hat{f}) \leq \Theta_c$). Please note the computing capacity is represented by platform's frequency in Figure 4.13.

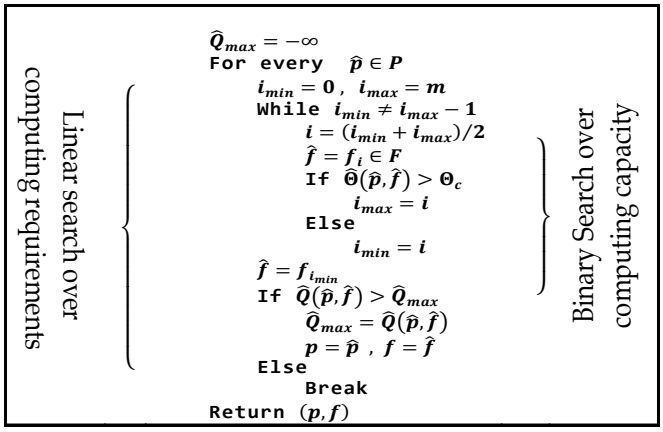


Figure 4.13 Pseudo code of DWVFS algorithm (online steps).

Then the selection process completes with a linear search in the direction of computing requirement over the quality function, $\mathrm{Q}(C_c, C_r)$, as the outer loop. Computing requirement is represented by the application parameter in Figure 4.13. Since $\boldsymbol{P}$ is sorted in decreasing order of computing requirement, the outer loop sweeps the critical temperature contour line in the direction of point a to point b in Figure 4.9. By using functional quality model, it compares the functional quality of the points on θ_c contour line, $(\hat{p},\hat{f})$, and ends as soon as the quality drops. At the end, we have identified all the points on the critical temperature contour line (C_{θ_c} in Figure 4.9) and selected the point which produces the maximum functional quality among them (C_{Q_2} and C_{θ_c} intersect in Figure 4.9).

The proposed DWVFS algorithm is correct for both types of applications in Figure 4.9. For the first type of application ($\partial Q/\partial C_r > 0$), Theorem 1 says the solution is the point

on the θ_c contour line with the highest computing requirement. As we mentioned, this algorithm sweeps computing requirement in the decreasing order from point a to b. In the first iteration, it chooses the parameter with the highest C_r and the frequency which is located on θ_c contour line. In the next loop iteration, algorithm compares the next point of critical temperature contour line and since it has smaller quality than the first point (Theorem 1), the algorithm ends. For the second type of application, Theorem 2 claims that the functional quality increases from point a and somewhere in middle of θ_c contour line it drops. Again the algorithm captures the same behavior and finds the solution provided in Theorem 2. The above discussion shows that the DWVFS algorithm finds the solution for both types of applications and is correct by construction.

The proposed DWVFS algorithm has a time complexity of $O\big(nlog(m)\big)$, where n is the number of values of application parameter, and m is the number of available frequency levels for the platform. Since the value of n is not large for typical applications (for e.g., 11 and 10 for the H.264 encoder and Turbo decoder respectively), and the value of m for typical platforms is small (like 5 for the MacBook Air platform), the algorithm can execute in real time. For example, the execution of the algorithm takes about 10 to 15 micro Seconds on the test platform depending on the CPU frequency; it is less than 0.04% of the decoding time for one H.264 frame. In the next section, we have used different DTM methods and shown the efficacy of DWVFS compared to other DTM methods.

4.6 Experimental Results

In this section, we show how the proposed thermal management technique can be used on two different real-time applications, namely a video encoder and a Turbo decoder. First, we briefly introduce the experimental setup that is used in this research. Then we discuss

the results of applying the proposed thermal management algorithm on the video encoder and Turbo decoder.

4.6.1 Experimental Setup

We evaluated the proposed thermal management technique using a MacBook Air laptop with an Intel Core2 Duo dual core processor, 2GB of RAM and a solid state disk. The operating system is Mac OS X 10.5. The laptop has a fan and a heat sink, which help with thermal control; however, they are not sufficient while running highly compute-intensive applications, as shown in the following sub-section. The speed of the fan can reach up to 6200 rounds per minute. During all of our experiments, the fan operated at the maximum speed constantly.

The platform provides 8 different frequency levels. The four main frequency levels are 1200, 1400, 1600, and 1800 MHz. In addition, by enabling an internal clock divider we can set platform frequency to 600, 700, 800, and 900 MHz. The selection of voltage level for each of these frequency levels are given by the vendor as specified in Table 3.3.

For dynamic voltage and frequency scaling based thermal management, we used the two frequency levels of 1200MHz and 1800MHz. The reason for this selection is twofold. First, as mentioned in [KS04], effective thermal management is possible with only two frequency levels. In addition, the test results show that, having more number of frequencies reduces the application functional quality.

The temperature sensor in the platform reports the temperature with one degree Celsius accuracy up to 103 degrees. If the temperature rises over 103 degrees, it reports the constant value of 103. Therefore, for temperatures above 103, we extrapolated the curves in the plots.

4.6.2 H.264 Video Encoder

The first application used in this study is an H.264 video encoder. We used an open source and highly efficient implementation of the H.264 standard called x264 [X264]. The application parameter used for controlling video encoder's computing requirement is the constant rate factor (CRF). CRF is the default rate control method of x264 which tries to achieve the same perceptual quality throughout a video stream [W264]. The CRF parameter can vary from 0 to 51, where a CRF of 0 produces a lossless compression and a CRF of 51 produces the lowest quality result. We chose a range from 16 to 36 for CRF, with an increment of 2. It is evident that increasing CRF will decrease the computing requirement and vice versa.

To measure the functional quality, we used peak signal to noise ratio (PSNR), which is one of the most widely used video quality metrics. To calculate PSNR, the original video stream is compared to the encoded video stream and the difference is measured in dB.

The video encoder's behavior is highly dependent on its input stream. For example, encoding a video stream with very little motion is less complex and easier than a high motion video stream. We used different video streams to capture this dependency. The first video stream is called *wild life*. Figure 4.14 shows the results of encoding the *wild life* clip when different thermal management algorithms are used.

Figure 4.14(a) shows the temperature vs. time using different thermal management methods. As mentioned in Section 4.6.1, temperatures above 103^{o}C cannot be measured by the platform's temperature sensor. The dots in the plot are the projected values for temperature which cannot be read from the sensor. The critical temperature in this test is 80^{o}C. As shown in Figure 4.14(a), all the DTM algorithms are able to control the temperature around the critical temperature. However, this comes with different penalties in the video quality. Figure

4.14(b) shows the measured quality of the video stream over the duration of the encoding process. As we can see, DWVFS based thermal management produces better quality compared to DWS and DVFS. In addition, DWS produces better quality compared to DVFS at all times. Figure 4.14(c) and (d) show the selection of frequency and CRF for different DTM algorithms. In the absence of DTM, the platform always runs with the maximum frequency and nominal CRF of 21. Comparing DWVFS with DWS and DVFS, we can see that DWVFS utilizes a combination of scaling frequency and CRF. Initially, DWVFS reduces the CRF (leading to higher quality) and as time passes it increases the CRF (leading to lower quality) for thermal management purposes. Concurrently, it also decreases the frequency in order to control the temperature while achieving higher quality. As depicted in Figure 4.14(c) and (d) the amount of increase in CRF in the case of DWVFS is not as much as in the case of DWS.

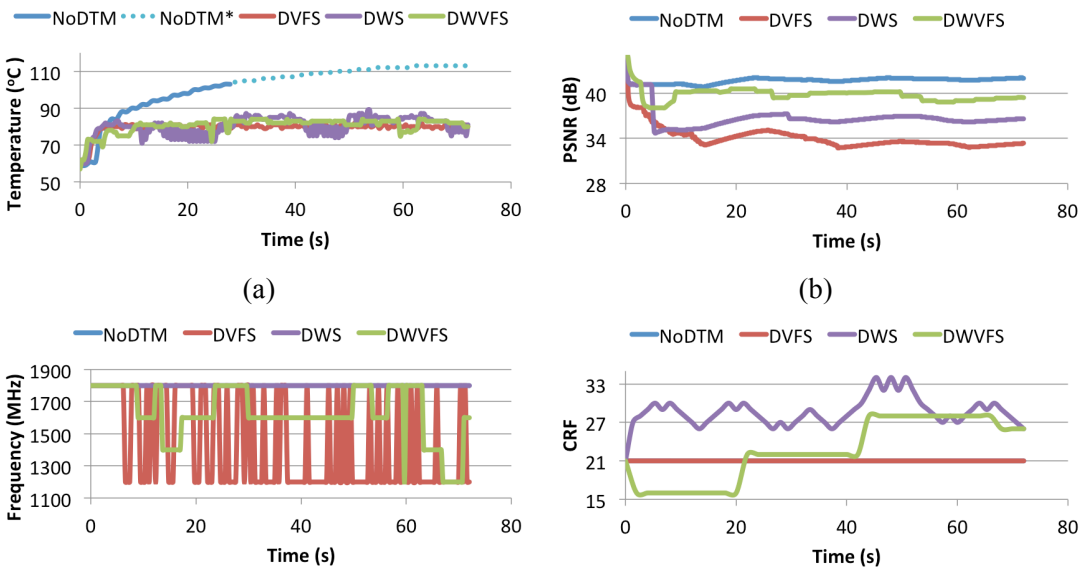


Figure 4.14 Results of encoding wild life video clip with H.264 encoder using different thermal management algorithms. (a) Effects on the temperature. Due to thermal sensor limitation, it cannot report values more than 103 degree Celsius. NoDTM* plot is the projection of temperature as it rises. (b) Effects of different DTM methods on the encoding quality. (c) Frequency levels used during different thermal management algorithms. (d) CRF values used during different thermal management algorithms.

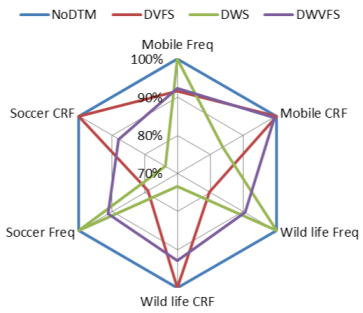


Figure 4.15 Normalized average of the selected CRF and frequency values with different DTM and video streams.

Since the complexity and quality of a video encoder is highly dependent on the video stream, we used three video clips to compare the thermal management techniques. Figure 4.15 shows the normalized averages of the selected CRF and frequency values. Normalized average of selected frequency is the ratio of the average frequency used during a video encoder run to the nominal frequency of the platform (1.8GHz). Since the CRF has inverse correlation with the video quality, we have used the ratio of the nominal CRF to the average of the selected CRF values as the normalized average CRF. As depicted in Figure 4.15, in the absence of any DTM algorithm, processor is always at its highest frequency and CRF is constantly at its nominal value. When DVFS is used, the CRFs for all the streams are still at the nominal values but the average frequencies can go as low as 80% of the nominal frequency depending on the stream. On the other hand, the frequency is always at the nominal value when DWS is in use for all the streams but the average CRF used varies. The plot for DWVFS is always in between DVFS and DWS for both frequency and CRF. This means that DWVFS uses each control knob moderately to ensure that the drop in quality is minimized. Eventually, the different selection of CRF and frequency in the different DTM techniques leads into different stream quality, as shown in Figure 4.16.

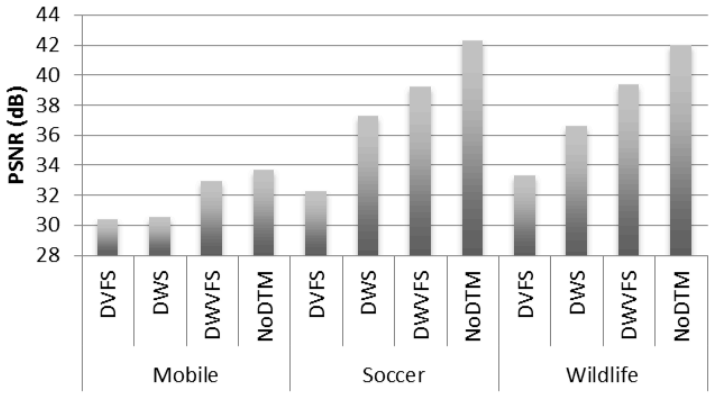


Figure 4.16 Visual quality of the different video streams under different thermal managements.

As depicted in Figure 4.16, the maximum quality drop due to the thermal management is about 10 dB. However, DWVFS helps to maintain a higher video quality and reduces the negative effects of thermal management. In addition, we should note that the high quality in the absence of DTM has been achieved with the price of increasing the processor's temperature over 110^{o}C. In practice, this will not happen if thermal protection is enabled, since a thermal management technique would be triggered when the platform reaches a maximum allowed temperature.

4.6.3 Turbo Decoder

The second application we used to verify the proposed thermal management scheme is a Turbo decoder used in a Turbo coding based hybrid automatic repeat request (HARQ) [YLD03] communication system. Figure 4.17 shows the block diagram of the HARQ Turbo coding system. HARQ is a hybrid method that uses both forward error-correcting (FEC) codes and ARQ error controlling mechanism. In ARQ, the data is sent with some error detecting (ED) code bits such as parity bits or cyclic redundancy check (CRC) bits [SPM06]. When the receiver detects an error in the data stream using ED codes, it requests a re-transmission. On the other hand, FEC codes help to not only find errors in the bit stream but also help to correct the errors to some extent. In HARQ, the receiver first tries to detect and correct errors using

the FEC codes. If the bit stream is not successfully retrieved using FEC codes, then the receiver requests re-transmission. The receiver communicates with the sender through ACK or NACK messages (Figure 4.17). X_k is the source information. The Turbo encoder produces two series of parity bits: P_{1k} and P_{2k}. The information and parity bit symbols go through the channel and the noisy information and parity symbols (x'_k, p'_{1k}, p'_{2k}) are received by the decoder. We considered an additive white Gaussian noise (AWGN) channel with a noise distribution of $N(0, \sigma^2)$, with $\sigma = 0.95$. The decoded bits are denoted by $\widehat{X_k}$.

Figure 4.17 Block diagram of the Turbo coding system.

In our experiments, we focus on the Turbo decoder in the receiver side. Figure 4.18 shows the block diagram of the Turbo decoder. In this figure, $L(X_k)$ is the likelihood of finding the source information, X_k. The Turbo decoder uses an iterative algorithm; every iteration of the loop increases the likelihood of finding the correct source information. Eventually, a decision will be made whether to request for a retransmission or the decoding is successfully completed.

The number of decoder iterations is the application parameter that is used in the DWS and DWVFS. The functional quality of Turbo decoder application is its effective throughput. The effective throughput is the number of decoded bits, $\widehat{X_k}$, per second. We have used the terms *effective throughput* and *throughput* interchangeably throughout this dissertation. When the platform gets too hot, by dropping the number of iterations, we reduce the work load and hence the platform temperature drops (Figure 4.19). However, by reducing the number of

iterations, the likelihood of correctly decoding source information decreases and we need more retransmissions. Higher number of retransmissions translates to a drop in the decoder's throughput. Therefore, the objective is to maximize the decoder's throughput while controlling the platform temperature. In this exercise, the frame size is 4701 bits and the interleaver is designed based on the UMTS standard [UMTS00].

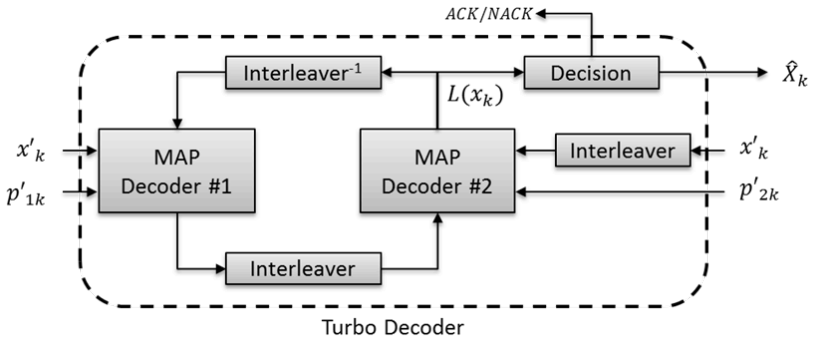


Figure 4.18 Turbo decoder block diagram consisting of two MAP decoders.

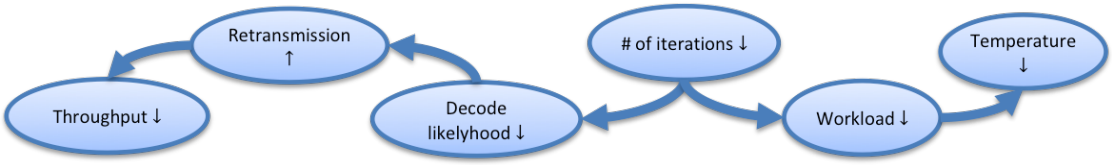


Figure 4.19 Effect of changing the number of iterations in the Turbo decoder on its throughput and temperature.

Figure 4.20 shows the quality vs. temperature contour plots for the Turbo decoder. The contour plots for the video encoder were presented earlier in Figure 4.5. As shown in Figure 4.5 and Figure 4.20, the contour plots of the video encoder and Turbo decoder follow the general shape that we discussed in Figure 4.7 and Figure 4.8.

Figure 4.21 shows the results of running the Turbo decoder with different thermal management algorithms on the MacBook Air platform. As shown in Figure 4.21(a), the platform becomes very hot (over 100°C) without thermal management which can deteriorate the reliability of the platform, necessitating the use of thermal management. Figure 4.21(a) also shows that all the DTM methods control the temperature to the specified critical temperature of 75°C. However, as we see in Figure 4.21(b), different DTM methods affect the effective throughput of the decoder differently. Figure 4.21(c) and (d) show how the average values of frequency and application-level parameter (number of decoder iterations) vary across DTM algorithms. We see that in the case of DWVFS, the average frequency is higher than DVFS and average number of decoding iterations is higher than DWS. This suggests that DWVFS uses the best of both knobs to ensure that the functional quality, i.e., throughput, is maximized.

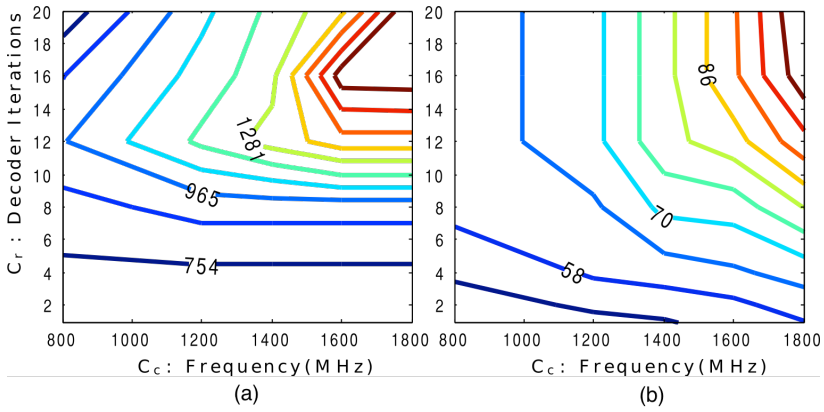


Figure 4.20 Contour plots of (a) functional quality of turbo decoder measured as throughput in bps, (b) platform temperature while running turbo decoder.

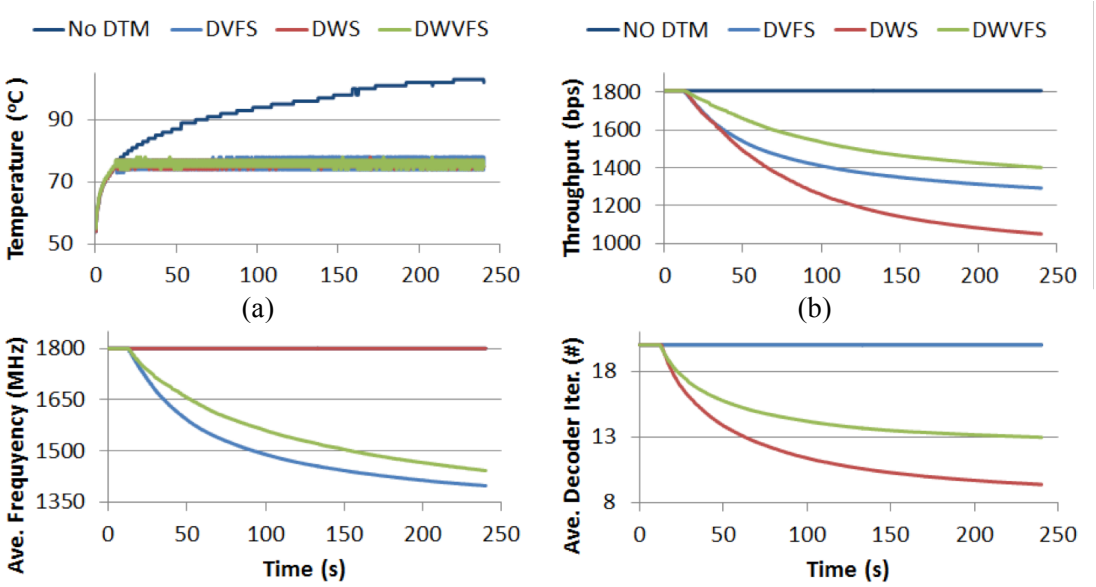


Figure 4.21 Results of running the Turbo decoder using different thermal management algorithms. (a) Effect on temperature. (b) Effect of different DTM methods on the Turbo decoder throughput. (c) Average of frequency levels used by different thermal management algorithms. (d) Average number of decoder iterations by different thermal management algorithms.

Figure 4.22 compares the throughput of different DTM methods. In this scenario, the throughput of DVFS is better than DWS. However, DWVFS again achieves the best results compared to other DTM methods. According to this figure, the throughput can drop by about 33% by using DWS but with the help of DWVFS the drop is limited to 22% only. In fact DWVFS improves the throughput by 8% and 33% compared to DVFS and DWS based DTMs respectively.

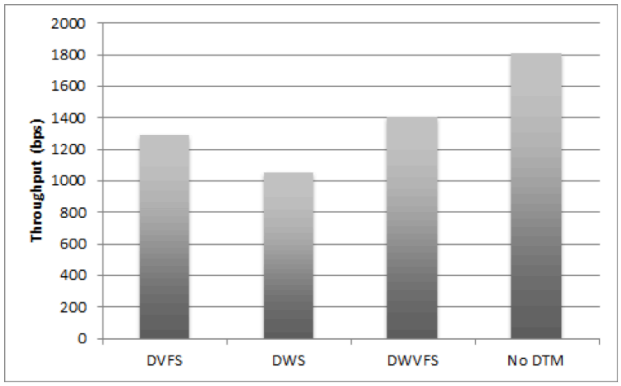


Figure 4.22 Effects of different DTM methods on the throughput of the transmission.

4.7 Conclusion

In this research, we showed how joint dynamic work/complexity scaling of real time applications could be used as a new DTM method. We analyzed the effects of the platform's computing capacity and the application's computing requirements on both the platform's temperature and the application's functional quality. We showed how one can use these parameters to achieve a quality-optimized DTM. In addition, we formulated and analytically solved the optimization problem. We also proposed a fast algorithm to implement DWVFS.

In this research we only considered DVFS as the mechanism to change the computing capacity. However, there are other parameters that can be used to vary computing capacity, for example, the number of active cores. Moreover, we only considered one parameter to scale the application's complexity.

In future work, the proposed approach can be extended to a) cover other mechanisms to vary computing capacity such as throttling number of CPU cores, b) have multiple parameters affecting the workload of the application at the same time, and c) scenarios when multiple real-time applications are running on the platform concurrently.

Acknowledgements

This chapter has been co-authored with Dr Sujit Dey and Dr Anand Raghunathan and is accepted for publication by IEEE Transactions on Very Large Scale Integration (TVLSI).

Appendix

In this section we present the necessary background needed to conclude the Corollary I-IV in Section 4.4.

Proposition 1. For a two-variable function such as $f(x,y)$, a contour line that is extracted from $f(x,y)=c$ can be represented parametrically by the following equation [HA84b]:

$$\begin{cases} x = x(t) \\ y = y(t) \end{cases} \tag{4.33}$$

Proposition 2. For a differentiable parametric curve such as equation (4.33), the slope of the curve is equal to [HA84a]:

$$\frac{dy}{dx} = \frac{dy/dt}{dx/dt} \tag{4.34}$$

Theorem A1. The slope of contour lines of a differentiable two-variable function$f(x,y)$ can be extracted from the function's partial derivatives and is equal to:

$$\frac{dy}{dx} = -\frac{\partial f/\partial x}{\partial f/\partial y} \tag{4.35}$$

Proof. A contour line is extracted from the equation: $f(x,y)=c$.

Considering Proposition 1 and getting the derivate of the above equation with respect to parameter of $x(t)$ and $y(t)$, we have:

$$\frac{d}{dt}(f(x,y)=c) \xrightarrow{\text{Chain Rule}} \frac{\partial f}{\partial x}\frac{dx}{dt}+\frac{\partial f}{\partial y}\frac{dy}{dt}=0 \Rightarrow \frac{dy/dt}{dx/dt}=-\frac{\partial f/\partial x}{\partial f/\partial y}$$

$$\xrightarrow{\text{Eq. (3.34)}} \frac{dy}{dx}=-\frac{\partial f/\partial x}{\partial f/\partial y} \blacksquare$$

Theorem A2. If the sign of partial derivatives of a two-variable function does not change in a region, and at least one of the partial derivatives is non-zero, then the contour lines of the function do not cross each other.

Proof. Without loss of generality, we consider $\partial f/\partial x$ is the non-zero partial derivative. We will prove this theorem by contradiction. Assume two contour lines C_1 and C_2 cross each other at point (x_0, y_0) (Figure 4.23). Therefore, the value of the function is equal to $f(x_0, y_0)$ at any point on contour lines C_1 and C_2. Assume a new point on y axis $y_1 > y_0$. Therefore the point (x_{1_1}, y_1) is on the contour line C_1 and point (x_{1_2}, y_1) is on the contour line C_2. Since these points are on the contour lines, then by definition we conclude: $f(x_{1_1}, y_1) = f(x_{1_2}, y_1) = f(x_0, y_0)$.

From basic calculus we know if the value of a differentiable function is the same in two points, either the slope of the function is zero or the function has at least one extremum between the two points. If it is the former, it contradicts the initial assumption of non-zero partial derivative for $\partial f/\partial y$. If it is the latter, the existence of an extremum means that the sign of the slope changes in this region which also contradicts the initial assumption of the proof.■

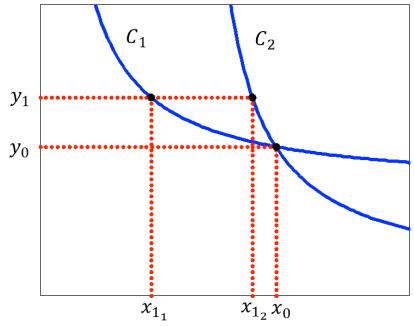


Figure 4.23 Two contour lines C_1 and C_2 crossing at the point(x_0, y_0).

Chapter 5

Conclusions and Future Directions

This chapter summarizes the contributions made in this research in addition to some recommendations for future research.

As process variations increase in significance and magnitude, the cost of hiding them through hardware over-design will become prohibitive. In chapter 2, we investigated application adaptation as a strategy to deal with hardware that is impacted by variations, and demonstrate it through real-time video encoding. Given that the net impact of variations is to reduce the computing capability (performance) of the underlying hardware, we adapt applications to proportionally reduce their computational requirements while preserving their run time, visual quality and bit rate as much as possible. We identified and characterized the effect of encoder parameters on run time, video quality, and bit rate of the encoded video. We described an adaptation algorithm that could select the best values of the parameters to maintain real-time performance under variation degraded frequency of hardware, while satisfying specified quality and bit rate constraints. We implemented the proposed approach for a very efficient implementation of H.264, x264, on two different hardware platforms – an embedded platform (Beagleboard) with the ARM Cortex A8 processor, and a server platform with the Intel Core i7 processor. Our measurements showed that the proposed adaptation approach can tolerate up to 30% in frequency degradation, while being able to sustain encoded video quality to near-identical levels and incurring tolerable overhead in bit rate.

Due to limitations of cooling methods such as using fan and heat sink, dynamic thermal management (DTM) is being widely adopted to manage the temperature of computing systems. However, application of DTM can reduce the system performance, and thereby affect the quality of real-time applications. Real-time video encoding, which has high computational need and hard deadlines, is a commonly used application that can be severely affected by the usage of DTM. In chapter 3, we studied the effect of DTM on a widely used H.264 video encoder, and formulated a multi-dimensional optimization problem to maximize video quality and minimize bit rate while ensuring that the video encoder can run in real-time in spite of DTM effects. We modeled the effects of adapting encoding parameters on video quality, bit rate, and encoder speed. We proposed a dynamic application adaptation method to efficiently solve the optimization problem by optimally adapting the encoding parameters in response to DTM effects. In addition, we showed that the proposed dynamic application adaptation method would reduce the need for cooling methods such as forced convection cooling. We implemented the proposed approach on an Intel® Core™ 2 Duo platform where dynamic voltage and frequency scaling (DVFS) is used for DTM. Our measurements with several videos reveal that when DTM is applied, the video quality is affected significantly. However, using the proposed adaptation algorithm, the encoder can run in real-time, and the quality loss is minimized with only a marginal increase in the bit rate.

In chapter 4, we focused on real-time applications in which degradation in performance translates to a loss in application quality, and addressed the problem of quality-optimized DTM, wherein the objective of DTM is to satisfy specified temperature constraints while optimizing application quality metrics. We first introduced a new DTM method called Dynamic Work Scaling (DWS), which is based on modulating an application's computational requirements. Next, we observed that application quality and platform temperature are

effectively determined by two key parameters, viz. the application's computational requirement and the platform's computing capacity, and formulated the relationship between them. Finally, we proposed a quality-optimized DTM based on joint dynamic work and voltage/frequency scaling (DWVFS). We have implemented the proposed DTM technique and evaluated it for two applications: H.264 video encoding and Turbo decoding. Our results demonstrated that DWVFS can provide superior results in terms of application quality compared to both DVFS and DWS based DTM at identical temperature constraints.

We studied a scenario where a single application is using most of the platform's computing capacity in chapter 3. For future work, it would be very interesting to study a scenario where multiple compute-intensive applications are running on the platform simultaneously, each having significant influence on the thermal characteristics of the platform. When multiple applications are being considered, maximizing the quality of all the applications (or their aggregate quality) must also be considered, while ensuring the joint impact of each of the applications' runtime is below their specified timing criteria. This would lead to a multi objective problem with a set of pareto opitmal solutions.

Extending single application problem to multi applications problem can be applied to the proposed solution in chapter 4 as well. However, there is a big difference between the implementation of the solutions of these two problems. The solution provided in chapter 3 is a reactive technique and only affects the application it is dealing with. Therefore, it can be adapted easier for multiple applications since each application adaptation decision could be made independently. However, the solution provided in chapter 4 is a proactive technique meaning it changes the DVFS decisions while adaptation decisions are being made. Therefore a centralized solution is needed that can consider all applications and hardware DTM and make proper decisions for all of them concurrently.

Finally, we showed in chapter 3 that application adaptation could help to reduce the cooling efforts. One interesting problem would be to include the cooling efforts into the optimization problem and introduce a problem that maximizes application quality while minimizes the cooling cost using application adaptation. It is shown that application adaptation can work very well in a hybrid solution for thermal management; therefore, we expect it to be a good candidate to be used in a hybrid solution that minimizes the cooling cost as well.

Bibliography

[AKC09] Coşkun, Ayşe Kıvılcım. "Efficient thermal management for multiprocessor systems." University of California San Diego (2009).

[BAS09] Bowman, Keith A., Alaa R. Alameldeen, Srikanth T. Srinivasan, and Chris B. Wilkerson. "Impact of die-to-die and within-die parameter variations on the clock frequency and throughput of multi-core processors." Very Large Scale Integration (VLSI) Systems, IEEE Transactions on 17, no. 12 (2009): 1679-1690.

[BGL15] Beagle Board. [Online] [Cited: May 9, 2015.] http://www.beagleboard.org.

[BKN03] Borkar, Shekhar, Tanay Karnik, Siva Narendra, Jim Tschanz, Ali Keshavarzi, and Vivek De. "Parameter variations and impact on circuits and microarchitecture." In Proceedings of the 40th annual Design Automation Conference, pp. 338-342. ACM, 2003.

[BS00] Butts, J. Adam, and Gurindar S. Sohi. "A static power model for architects." InProceedings of the 33rd annual ACM/IEEE international symposium on Microarchitecture, pp. 191-201. ACM, 2000.

[CLR06] Chandra, Saumya, Kanishka Lahiri, Anand Raghunathan, and Sujit Dey. "Considering process variations during system-level power analysis." In Low Power Electronics and Design, 2006. ISLPED'06. Proceedings of the 2006 International Symposium on, pp. 342-345. IEEE, 2006.

[CLR10] Chandra, Saumya, Kanishka Lahiri, Anand Raghunathan, and Sujit Dey. "Variation-aware system-level power analysis." Very Large Scale Integration (VLSI) Systems, IEEE Transactions on 18, no. 8 (2010): 1173-1184.

[CR96] Courant, Richard, and Herbert Robbins. What is Mathematics?: an elementary approach to ideas and methods. Oxford University Press, 1996. pp. 344.

[CSC14] Cisco. 2014. Cisco Visual Networking Index: Forecast and Methodology, 2013-2018. June 2014.

[CWT09] Chen, Jian-Jia, Shengquan Wang, and Lothar Thiele. "Proactive speed scheduling for real-time tasks under thermal constraints." In Real-Time and Embedded Technology and Applications Symposium, 2009. RTAS 2009. 15th IEEE, pp. 141-150. IEEE, 2009.

[DD09] Deza, Michel Marie, and Elena Deza. Encyclopedia of distances. Springer Berlin Heidelberg, 2009. pp. 86.

[DMK10] Duggirala, Parasara Sridhar, Sayan Mitra, Rakesh Kumar, and Dean Glazeski. "On the theory of stochastic processors." In Quantitative Evaluation of Systems (QEST), 2010 Seventh International Conference on the, pp. 292-301. IEEE, 2010.

[ES07] Elgebaly, Mohamed, and Manoj Sachdev. "Variation-aware adaptive voltage scaling system." Very Large Scale Integration (VLSI) Systems, IEEE Transactions on 15, no. 5 (2007): 560-571.

[FS13] Forte, Domenic, and Ankur Srivastava. "Energy-and Thermal-Aware Video Coding via Encoder/Decoder Workload Balancing." ACM Transactions on Embedded Computing Systems (TECS) 12, no. 2s (2013): 96.

[GC09] Gerald Coley. BeagleBoard System Reference Manual Rev C4. BeagleBoard.org, December 15, 2009.

[GGP12] Ghasemazar, Mohammad, Hadi Goudarzi, and Massoud Pedram. "Robust optimization of a Chip Multiprocessor's performance under power and thermal constraints." In Computer Design (ICCD), 2012 IEEE 30th International Conference on, pp. 108-114. IEEE, 2012.

[GM08] Garg, Siddharth, and Diana Marculescu. "System-level mitigation of WID leakage power variability using body-bias islands." In Proceedings of the 6th IEEE/ACM/IFIP international conference on Hardware/Software codesign and system synthesis, pp. 273-278. ACM, 2008.

[GM13] Garg, Siddharth, and Diana Marculescu. "System-level throughput analysis for process variation aware multiple voltage-frequency island designs." ACM Transactions on Design Automation of Electronic Systems (TODAES) 13, no. 4 (2008): 59.

[GQ11] Ge, Yang, and Qinru Qiu. "Dynamic thermal management for multimedia applications using machine learning." In Proceedings of the 48th Design Automation Conference, pp. 95-100. ACM, 2011.

[HA84a] Anton, Howard. Calculus with Analytical Geometry. 2nd edition, Addison-Wesley, 1984. pp. 724.

[HA84b] Anton, Howard. Calculus with Analytical Geometry. 2nd edition, Addison-Wesley, 1984. pp. 908.

[HAC11] Huang, Wei, Malcolm Allen-Ware, John B. Carter, Elmootazbellah Elnozahy, Hendrik Hamann, Tom Keller, Charles Lefurgy, Jian Li, Karthick Rajamani, and Juan Rubio. "Tapo: Thermal-aware power optimization techniques for servers and data centers." In Green Computing Conference and Workshops (IGCC), 2011 International, pp. 1-8. IEEE, 2011.

[HCQ13] Huang, Huang, Vivek Chaturvedi, Gang Quan, Jeffrey Fan, and Meikang Qiu. "Throughput maximization for periodic real-time systems under the maximal temperature constraint." ACM Transactions on Embedded Computing Systems (TECS) 13, no. 2s (2014): 70.

[HG08] Huynh-Thu, Quan, and Mohammed Ghanbari. "Scope of validity of PSNR in image/video quality assessment." Electronics letters 44, no. 13 (2008): 800-801.

[HH15] Hendrik Holtmann, smcFanControl [Online] [Cited: May 9, 2015] http://www.eidac.de.

[HV12] Hanumaiah, Vinay, and Sarma Vrudhula. "Temperature-aware DVFS for hard real-time applications on multicore processors." Computers, IEEE Transactions on 61, no. 10 (2012): 1484-1494.

[IB07] Ivanov, Yuri V., and Chris J. Bleakley. "Dynamic complexity scaling for real-time H. 264/AVC video encoding." In Proceedings of the 15th international conference on Multimedia, pp. 962-970. ACM, 2007.

[IB10] Ivanov, Yuri V., and Chris J. Bleakley. "Real-time H. 264 video encoding in software with fast mode decision and dynamic complexity control." ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP) 6, no. 1 (2010): 5.

[JMR10] Joint Model reference software version 10.2. [Online] [Cited: May, 9, 2015] http://iphome.hhi.de/suehring/tml/index.htm

[JP07] Jung, Hwisung, and Massoud Pedram. "Stochastic dynamic thermal management: A Markovian decision-based approach." In Computer Design, 2006. ICCD 2006. International Conference on, pp. 452-457. IEEE, 2007.

[JRP08] Jung, Hwisung, Peng Rong, and Massoud Pedram. "Stochastic modeling of a thermally-managed multi-core system." In Proceedings of the 45th annual Design Automation Conference, pp. 728-733. ACM, 2008.

[JS05] Snyman, Jan. Practical mathematical optimization: an introduction to basic optimization theory and classical and new gradient-based algorithms. Vol. 97. Springer, 2005.

[KCC06] Kim, Yangsoo, Yoonsik Choe, and Yungho Choi. "Fast mode decision algorithm for H. 264 using AZCB prediction." In Consumer Electronics, 2006. ICCE'06. 2006 Digest of Technical Papers. International Conference on, pp. 33-34. IEEE, 2006.

[KK05] Kim, Changsung, and C-CJ Kuo. "A feature-based approach to fast H. 264 intra/inter mode decision." In Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on, pp. 308-311. IEEE, 2005.

[KRB06] Kannangara, C. Sampath, Iain EG Richardson, Maja Bystrom, Jose R. Solera, Yafan Zhao, Andrew MacLennan, and Robert Cooney. "Low-complexity skip prediction for H. 264 through Lagrangian cost estimation." Circuits and Systems for Video Technology, IEEE Transactions on 16, no. 2 (2006): 202-208.

[KS04] Skadron, Kevin. "Hybrid architectural dynamic thermal management." InProceedings of the conference on Design, automation and test in Europe-Volume 1, p. 10010. IEEE Computer Society, 2004.

[KSP06] Kumar, Amit, Li Shang, Li-Shiuan Peh, and Niraj K. Jha. "HybDTM: a coordinated hardware-software approach for dynamic thermal management." InProceedings of the 43rd annual Design Automation Conference, pp. 548-553. ACM, 2006.

[KSP08] Kumar, Amit, Li Shang, Li-Shiuan Peh, and Niraj K. Jha. "System-level dynamic thermal management for high-performance microprocessors." Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on 27, no. 1 (2008): 96-108.

[LB06a] Liang, Xiaoyao, and David Brooks. "Microarchitecture parameter selection to optimize system performance under process variation." In Computer-Aided Design, 2006. ICCAD'06. IEEE/ACM International Conference on, pp. 429-436. IEEE, 2006.

[LB06b] Liang, Xiaoyao, and David Brooks. "Mitigating the impact of process variations on processor register files and execution units." In Microarchitecture, 2006. MICRO-39. 39th Annual IEEE/ACM International Symposium on, pp. 504-514. IEEE, 2006.

[LCL05] Li, Gwo-Long, Mei-Juan Chen, Hung-Ju Li, and Ching-Ting Hsu. "Efficient search and mode prediction algorithms for motion estimation in H. 264/AVC." InCircuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on, pp. 5481-5484. IEEE, 2005.

[LMM06] Lin, Yuan, Scott Mahlke, Trevor Mudge, Chaitali Chakrabarti, Alastair Reid, and Krisztián Flautner. "Design and implementation of turbo decoders for software defined radio." In Signal Processing Systems Design and Implementation, 2006. SIPS'06. IEEE Workshop on, pp. 22-27. IEEE, 2006.

[LPP06] Lee, Wonbok, Kimish Patel, and Massoud Pedram. "Dynamic thermal management for mpeg-2 decoding." In Proceedings of the 2006 international symposium on Low power electronics and design, pp. 316-321. ACM, 2006.

[LPP08] Lee, Wonbok, Kimish Patel, and Massoud Pedram. "GOP-level dynamic thermal management in MPEG-2 decoding." Very Large Scale Integration (VLSI) Systems, IEEE Transactions on 16, no. 6 (2008): 662-672.

[LRP05] Lotfallah, Osama, Martin Reisslein, and Sethuraman Panchanathan. "Adaptive bitstream switching of pre-encoded PFGS video." In Proceedings of the ACM workshop on Advances in peer-to-peer multimedia streaming, pp. 11-20. ACM, 2005.

[MB15] Marcel Bresink. Hardware Monitor [Online] [Cited: May 9, 2015] http://www.bresink.com.

[MDR14] Ali Mirtar, Sujit Dey, and Anand Raghunathan. "Joint Work and Voltage/Frequency Scaling for Quality-Optimized Dynamic Thermal Management." Very Large Scale Integration (VLSI) Systems, IEEE Transactions on , vol.PP, no.99, pp.1,1

[MG06] Marculescu, Diana, and Siddharth Garg. "System-level process-driven variability analysis for single and multiple voltage-frequency island systems." InComputer-Aided Design, 2006. ICCAD'06. IEEE/ACM International Conference on, pp. 541-546. IEEE, 2006.

[MKR09] Mohapatra, Debabrata, Georgios Karakonstantis, and Kaushik Roy. "Significance driven computation: a voltage-scalable, variation-aware, quality-tuning motion estimator." In Proceedings of the 14th ACM/IEEE international symposium on Low power electronics and design, pp. 195-200. ACM, 2009.

[ML15] Mangus Lundholm. CoolBook [Online] [Cited: May 9, 2015.] http://coolbook.se/CoolBook.html.

[MN04] Ted MacNeil, 2004. Don’t Be Misled by MIPS. IBM Systems Magazine, November, 2004

[MSG10] Momtazpour, Mahmoud, Esmaeil Sanaei, and Maziar Goudarzi. "Power-yield optimization in MPSoC task scheduling under process variation." In Quality

Electronic Design (ISQED), 2010 11th International Symposium on, pp. 747-754. IEEE, 2010.

[MSU15] MSU Video Quality Measurement Tool. Compression.ru. [Online] [Cited: May 9, 2015.]http://compression.ru/video/quality_measure/video_measurement_tool_en.html.

[MV06] L. Merritt, and R. Vanam. x264: A high performance H.264/AVC encoder, [Online] [Cited: May, 9, 2015] http://neuron2.net/library/avc/overview_x264_v8_5.pdf (2006).

[OMM08] Ogras, Umit Y., Radu Marculescu, and Diana Marculescu. "Variation-adaptive feedback control for networks-on-chip with multiple clock domains." InProceedings of the 45th annual Design Automation Conference, pp. 614-619. ACM, 2008.

[PGP10] Pakbaznia, Ehsan, Mohammad Ghasemazar, and Massoud Pedram. "Temperature-aware dynamic resource provisioning in a power-optimized datacenter." In Proceedings of the Conference on Design, Automation and Test in Europe, pp. 124-129. European Design and Automation Association, 2010.

[PGS12] Pant, Aashish, Puneet Gupta, and Mihaela Van Der Schaar. "AppAdapt: opportunistic application adaptation in presence of hardware variation." Very Large Scale Integration (VLSI) Systems, IEEE Transactions on 20, no. 11 (2012): 1986-1996.

[PSA14] Palomino, Daniel, Muhammad Shafique, Hussam Amrouch, Altamiro Susin, and Jorg Henkel. "hevcDTM: application-driven dynamic thermal management for high efficiency video coding." In Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014, pp. 1-4. IEEE, 2014.

[PUA12] Park, Junyoung, H. Mert Ustun, and Jacob A. Abraham. "Run-time prediction of the optimal performance point in dvs-based dynamic thermal management." InVLSI Design (VLSID), 2012 25th International Conference on, pp. 155-160. IEEE, 2012.

[PZH09] Venkatesan Packirisamy, Antonia Zhai, Wei-Chung Hsu, Pen-Chung Yew, Tin-Fook Ngai, 2009. Exploring speculative parallelism in SPEC2006. IEEE International Symposium on Performance Analysis of Systems and Software, 2009. ISPASS 2009. , pp.77,88, 26-28 April 2009.

[QZW08] Quan, Gang, Yan Zhang, William Wiles, and Pei Pei. "Guaranteed scheduling for repetitive hard real-time tasks under the maximal temperature constraint."

InProceedings of the 6th IEEE/ACM/IFIP international conference on Hardware/Software codesign and system synthesis, pp. 267-272. ACM, 2008.

[RN03] Russell, Stuart, Peter Norvig. "Artificial Intelligence: A Modern Approach. " Second edition Upper Saddle River, New Jersey: Prentice Hall, pp. 111–114, 2003.

[RPC06] Rao, G. Nageswara, R. S. V. Prasad, D. Jaya Chandra, and Srividya Narayanan. "Real-time software implementation of H. 264 baseline profile video encoder for mobile and handheld devices." In Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on, vol. 5, pp. V-V. IEEE, 2006.

[RV07] Rao, Ravishankar, and Sarma Vrudhula. "Performance optimal processor throttling under thermal constraints." In Proceedings of the 2007 international conference on Compilers, architecture, and synthesis for embedded systems, pp. 257-266. ACM, 2007.

[RW84] Reinhold Weicker, 1984. Dhrystone: A Synthetic Systems Programming Benchmark. Communications of the ACM 27 (10): 1013–30

[SAS02] Skadron, Kevin, Tarek Abdelzaher, and Mircea R. Stan. "Control-theoretic techniques and thermal-RC modeling for accurate and localized dynamic thermal management." In High-Performance Computer Architecture, 2002. Proceedings. Eighth International Symposium on, pp. 17-28. IEEE, 2002.

[SBH10] Shafique, Muhammad, Lars Bauer, and Jörg Henkel. "enbudget: A run-time adaptive predictive energy-budgeting scheme for energy-aware motion estimation in h. 264/mpeg-4 avc video encoder." In Design, Automation & Test in Europe Conference & Exhibition (DATE), 2010, pp. 1725-1730. IEEE, 2010.

[SCC10] Shin, Donghwa, Sung Woo Chung, Eui-Young Chung, and Naehyuck Chang. "Energy-optimal dynamic thermal management: Computation and cooling power co-optimization." Industrial Informatics, IEEE Transactions on 6, no. 3 (2010): 340-351.

[SHK10] Srinivasan, Vasudevan, Jim G. Hermerding, and Rahul Khanna. "An innovative approach to dynamic platform and thermal management for Mobile platforms." In Energy Aware Computing (ICEAC), 2010 International Conference on, pp. 1-4. IEEE, 2010.

[SMH10] Shafique, Muhammad, Bastian Molkenthin, and Jörg Henkel. "An HVS-based adaptive computational complexity reduction scheme for H. 264/AVC video encoder using prognostic early mode exclusion." In Proceedings of the Conference

on Design, Automation and Test in Europe, pp. 1713-1718. European Design and Automation Association, 2010.

[SP04] Schlegel, Christian, and Lance Perez. Trellis and turbo coding. Vol. 10. John Wiley & Sons, 2004.

[SPM06] Stigge, Martin, Henryk Plötz, Wolf Müller, and Jens-Peter Redlich. "Reversing CRC--Theory and Practice." (2006).

[SSS04] Skadron, Kevin, Mircea R. Stan, Karthik Sankaranarayanan, Wei Huang, Sivakumar Velusamy, and David Tarjan. "Temperature-aware microarchitecture: Modeling and implementation." ACM Transactions on Architecture and Code Optimization (TACO) 1, no. 1 (2004): 94-125.

[TST07] Tiwari, Abhishek, Smruti R. Sarangi, and Josep Torrellas. "ReCycle:: pipeline adaptation to tolerate process variation." In ACM SIGARCH Computer Architecture News, vol. 35, no. 2, pp. 323-334. ACM, 2007.

[TT08] Teodorescu, Radu, and Josep Torrellas. "Variation-aware application scheduling and power management for chip multiprocessors." In ACM SIGARCH Computer Architecture News, vol. 36, no. 3, pp. 363-374. IEEE Computer Society, 2008.

[UMTS00] ETSI, TS. "125 212 (V3. 1.1): "Universal Mobile Telecommunications System (UMTS)." Multiplexing and channel coding (FDD)(3G TS 25.212 version 3.1. 1 Release 1999) (2000)

[UTB06] Unsal, Osman S., James W. Tschanz, Keith Bowman, Vivek De, Xavier Vera, Antonio Gonzalez, and Oguz Ergin. "Impact of parameter variations on circuits and microarchitecture." IEEE micro 26, no. 6 (2006): 30-39.

[W264] MeWiki, “X264 Settings”, [Online], Available: http://mewiki.project357.com/wiki/X264_Settings#crf

[WB08] Wang, Shengquan, and Riccardo Bettati. "Reactive speed control in temperature-constrained real-time systems." Real-Time Systems 39, no. 1-3 (2008): 73-95.

[WC06] Wei, Zhe, and Canhui Cai. "Realization and optimization of DSP based H. 264 encoder." In Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium on, pp. 4-pp. IEEE, 2006.

[WD10] Wang, Shaoxuan, and Sujit Dey. "Rendering adaptation to address communication and computation constraints in cloud mobile gaming." In Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE, pp. 1-6. IEEE, 2010.

[WHB06] Weste, Neil, David Harris, Ayan Banerjee. "CMOS VLSI Design: A Circuits And Systems Perspective." Third edition. Pearson Education India, 2006.

[WNW07] Wang, Feng, Chrysostomos Nicopoulos, Xiaoxia Wu, Yuan Xie, and Narayanan Vijaykrishnan. "Variation-aware task allocation and scheduling for MPSoC." InComputer-Aided Design, 2007. ICCAD 2007. IEEE/ACM International Conference on, pp. 598-603. IEEE, 2007.

[WSB03] Wiegand, Thomas, Gary J. Sullivan, Gisle Bjontegaard, and Ajay Luthra. "Overview of the H. 264/AVC video coding standard." Circuits and Systems for Video Technology, IEEE Transactions on 13, no. 7 (2003): 560-576.

[WX10] Wu, Guowei, and Zichuan Xu. "Temperature-aware task scheduling algorithm for soft real-time multi-core systems." Journal of Systems and Software 83, no. 12 (2010): 2579-2590.

[WY08] Wang, Feng, and Yuan Xie. "Embedded multi-processor system-on-chip (mpsoc) design considering process variations." In Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on, pp. 1-5. IEEE, 2008.

[X264] x264. VideoLan. [Online] [Cited: November 15, 2010.] http://www.videolan.org/developers/x264.html.

[XYP13] Xie, Qing, Siyu Yue, Massoud Pedram, Donghwa Shin, and Naehyuck Chang. "Adaptive thermal management for portable system batteries by forced convection cooling." In Proceedings of the Conference on Design, Automation and Test in Europe, pp. 1225-1228. EDA Consortium, 2013.

[YBR13] Yang, Hoeseok, Iuliana Bacivarov, Devendra Rai, Jian-Jia Chen, and Lothar Thiele. "Real-time worst-case temperature analysis with temperature-dependent parameters." Real-Time Systems 49, no. 6 (2013): 730-762.

[YK08] Yeo, Inchoon, and Eun Jung Kim. "Hybrid dynamic thermal management based on statistical characteristics of multimedia applications." In Proceedings of the 13th international symposium on Low power electronics and design, pp. 321-326. ACM, 2008.

[YLD03] Yafeng, Wang, Zhang Lei, and Yang Dacheng. "Performance analysis of type III HARQ with turbo codes." In Vehicular Technology Conference, 2003. VTC 2003-Spring. The 57th IEEE Semiannual, vol. 4, pp. 2740-2744. IEEE, 2003.

[YLK07] Yeo, Inchoon, Heung Ki Lee, Eun Jung Kim, and Ki Hwan Yum. "Effective dynamic thermal management for MPEG-4 decoding." In Computer Design, 2007. ICCD 2007. 25th International Conference on, pp. 623-628. IEEE, 2007.

[ZC10] Zhang, Sushu, and Karam S. Chatha. "Thermal aware task sequencing on embedded processors." In Design Automation Conference (DAC), 2010 47th ACM/IEEE, pp. 585-590. IEEE, 2010.

[ZXD08] Zhou, Xiuyi, Yi Xu, Yu Du, Youtao Zhang, and Jun Yang. "Thermal management for 3D processors via task scheduling." In Parallel Processing, 2008. ICPP'08. 37th International Conference on, pp. 115-122. IEEE, 2008.

[ZXL12] Zhou, Xue, Lei Xie, Chaoqin Liu, Shuai Yang, Yi Chen, and Qiang Miao. "Noise analysis and reduction of cooling fans." In Quality, Reliability, Risk, Maintenance, and Safety Engineering (ICQR2MSE), 2012 International Conference on, pp. 656-660. IEEE, 2012.