Muhammad Usman Karim Khan
Muhammad Shafique
Jörg Henkel

Energy Efficient Embedded Video Processing Systems

A Hardware-Software Collaborative Approach

Springer

Energy Efficient Embedded Video Processing Systems

Muhammad Usman Karim Khan
Muhammad Shafique • Jörg Henkel

Energy Efficient Embedded Video Processing Systems

A Hardware-Software Collaborative Approach

Springer

Muhammad Usman Karim Khan
IBM Deutschland Research & Development GmbH
Böblingen, Germany

Muhammad Shafique
Institute of Computer Engineering
Vienna University of Technology
Vienna, Austria

Jörg Henkel
Department of Computer Science
Karlsruhe Institute of Technology
Karlsruhe, Germany

ISBN 978-3-319-61454-0 ISBN 978-3-319-61455-7 (eBook)
DOI 10.1007/978-3-319-61455-7

Library of Congress Control Number: 2017944437

Printed on acid-free paper

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

Contents

1 Introduction 1
- 1.1 Multimedia Systems 2
  - 1.1.1 Multimedia Processing Architectures 4
- 1.2 Fundamentals of Video Processing 5
  - 1.2.1 Video Compression 6
- 1.3 Design Complexity of a Video System 8
  - 1.3.1 The Dark Silicon Problem 9
  - 1.3.2 SRAM Aging 10
- 1.4 Video System Design Challenges 11
  - 1.4.1 Software Layer Challenges 14
  - 1.4.2 Hardware Layer Challenges 14
- 1.5 Limitations of State-of-the-Art 14
- 1.6 Design and Optimization Methods Discussed in This Book 15
  - 1.6.1 Software Layer Design 17
  - 1.6.2 Hardware Layer Design 18
  - 1.6.3 Open-Source Tools 20
- 1.7 Book Outline 21

References 21

2 Background and Related Work 25
- 2.1 Overview of Video Processing 25
  - 2.1.1 Intra- and Inter-frame Processing 28
- 2.2 Overview of Video Coding 29
  - 2.2.1 H.264/AVC and HEVC 32
  - 2.2.2 Parallelization 36
  - 2.2.3 DVC and HDVC 38
- 2.3 Technological Challenges 40
  - 2.3.1 Dark Silicon or Power Wall 41
  - 2.3.2 NBTI-Induced SRAM Aging 42
  - 2.3.3 Other Challenges 44

2.4 Related Work . . . 45
2.4.1 Video System Software . . . 45
2.4.2 Video Systems Hardware . . . 50
2.5 Summary of Related Work . . . 57
References . . . 57

3 Power-Efficient Video System Design . . . 67
3.1 System Overview . . . 67
3.1.1 Design Time Feature Support . . . 69
3.1.2 Runtime Features and System Dynamics . . . 71
3.2 Application and Motivational Analysis . . . 72
3.2.1 Video Application Parallelization . . . 73
3.2.2 Workload Variations . . . 75
3.2.3 HEVC Complexity Analysis . . . 76
3.3 Hardware Platform Analysis . . . 79
3.3.1 Heterogeneity Among Computing Nodes . . . 79
3.3.2 Memory Subsystem . . . 80
3.3.3 Analysis of Different Aging Balancing Circuits . . . 83
3.4 Summary . . . 86
References . . . 87

4 Energy-Efficient Software Design for Video Systems . . . 89
4.1 Power-Efficient Application Parallelization . . . 89
4.1.1 Power-Efficient Workload Balancing . . . 91
4.2 Compute Configuration . . . 94
4.2.1 Uniform Tiling . . . 94
4.2.2 Non-uniform Tiling . . . 96
4.2.3 Frequency Estimation ($f_{k,m}$) . . . 99
4.2.4 Maximum Workload Estimation ($\alpha_{k,m}$) . . . 100
4.2.5 Self-Regulated Frequency Model . . . 100
4.2.6 Retiling . . . 104
4.3 Application Configuration . . . 104
4.3.1 HEVC Application Configurations . . . 104
4.3.2 HEVC Configuration Tuning . . . 106
4.3.3 HEVC Parameter Mapping . . . 107
4.4 Workload Balancing on Heterogeneous Systems . . . 112
4.4.1 System Model . . . 112
4.4.2 Load Balancing Algorithm . . . 113
4.5 Resource Budgeting for Mixed Multithreaded Workloads . . . 116
4.5.1 Hierarchical Resource Budgeting . . . 118
4.5.2 Intra-Cluster Power Budgeting $p_{i,j}$ . . . 120
4.5.3 Inter-Cluster Power Budgeting p_i . . . 121
4.5.4 Selection of Cluster Size . . . 123
References . . . 124

5 Energy-Efficient Hardware Design for Video Systems 127
5.1 Custom Video Processing Architectures 127
5.1.1 Memory Analysis and Video Input 128
5.1.2 Video Preprocessing 129
5.1.3 DDR Video Write Master 129
5.1.4 Heterogeneous Computing Platform 131
5.2 Accelerator Allocation and Scheduling 132
5.2.1 Accelerator Sharing on Multi-/Many-Core 132
5.2.2 Multicast Video Processing Hardware 139
5.3 Efficient Hardware Accelerator Architectures 142
5.3.1 Low Latency H.264/AVC Encoding Loop 142
5.3.2 Distributed Hardware Accelerator Architecture 153
5.4 Hybrid Video Memory Architectures 157
5.4.1 AMBER Memory Hierarchy 158
5.4.2 NVM Reference Memory Architecture 159
5.4.3 Energy Management of NVM Reference Memories 161
5.4.4 System Computation Flow 163
5.5 Energy-Efficient Anti-aging Circuits for SRAM 164
5.5.1 Memory Write Transducer (MWT) 166
5.5.2 Aging-Aware Address Generation Unit (AGU) 167
5.5.3 Aging Controller 168
5.5.4 Generalization and Applicability 170
5.5.5 Sensitivity Analysis of SRAM Anti-aging Circuits 171
References 173

6 Experimental Evaluations and Discussion 175
6.1 Parallelization and Workload Balancing 175
6.1.1 Software Architecture and Simulation Setup 176
6.1.2 Compute and Application Configuration for Uniform Tiling 177
6.1.3 Compute Configuration with Non-uniform Tiling 182
6.1.4 Workload Balancing on Heterogeneous Platforms 184
6.2 Resource Budgeting 185
6.2.1 Experimental Setup 186
6.2.2 Results and Discussion 187
6.3 Memory Subsystem 188
6.3.1 AMBER: Hybrid Memories 189
6.3.2 SRAM Anti-aging Circuits 191
References 199

7 Conclusion and Future Outlook 201
7.1 Software-Level Techniques 201
7.2 Hardware-Level Techniques 203

7.3 Further Improvements 205
7.3.1 Approximate Computing 205
7.3.2 GPU-Based Acceleration 206
7.3.3 Reliability and Workload Management 206
7.3.4 Generalization 207
References 207

Appendices 211

Index 233

Chapter 1
Introduction

High-density, nanoscale fabrication technologies have enabled the chip designers to assemble billions of transistors on a single die. This has in turn provided the capability to realize high-complexity systems like video systems, which have universally penetrated into communication, security, education, entertainment, navigation, and robotics domains. The advancement in the fabrication technology has also driven the user expectations of the next-generation video processing systems. A prime example is high-resolution video capture and playback. Therefore, video applications like the latest video encoders [1] now target high-resolution video content compression beyond full-HD (like 4K ultrahigh definition, 3840×2160 pixels) at high frame rates (>120 frames per second). On the contrary, emergence and evolution of next-generation video systems (with increasing throughput and connectivity requirements and adaptability to application, battery life, etc.) require high processing capabilities and efficient utilization of the resources, which might be prohibitive on a resource- and power-constraint hardware platform. Though high-end systems can meet the throughput requirements, efficient and long-term deployment of such applications on small, battery-driven, autonomous systems is challenging, due to the high computational and power requirements while addressing the throughput constraints. Coupled with the high-throughput demands, a video system must be capable of responding in real time to changes in the workload of the application. Further, modern nano-era fabrication technologies have their own associated challenges (like power wall [2] and reliability [3]) which must be accounted for forging energy-efficient multimedia systems. This suggests that new design methodologies for next-generation video systems are needed, to address the abovementioned challenges on modern systems. This book presents some of these methodologies, both at the software and hardware layers of the system.

The authors would like to point out that this work was carried out when all the authors were in Department of Computer Science, Karlsruhe Institute of Technology, Karlsruhe, Germany.

M.U.K. Khan et al., *Energy Efficient Embedded Video Processing Systems*,
DOI 10.1007/978-3-319-61455-7_1

1.1 Multimedia Systems

Multimedia systems holistically store, process, and output/display several forms of data (like audio, image/video, text, and others), for tasks like entertainment, communication, security, computing, etc. These systems have penetrated as computers, televisions, ATM machines, medical instruments, and most recently handheld mobile devices. A multimedia system is a union of computers, communication, and signal processing domains. Typically, multimedia systems are required to process the content within a deadline and satisfy real-time quality of service (QoS) or service level agreement (SLA) constraints. This might become a significant challenge for small, mobile, battery-driven systems (like autonomous robots) as they encounter QoS requirements under resource and power constraints. Thus, multimedia system designers try to optimize different attributes of the system, which lead to optimization of a quality metric, like power efficiency. Concisely, maximizing power efficiency corresponds to maximizing performance per unit of power that goes into the system (generally referred to as maximizing throughput-per-watt). Further, the time complexity reduction (i.e., performance optimizations) of such system will also indirectly lead to power efficiency. This is because a lower clock frequency of the processing hardware can meet the QoS requirement, and thus, lesser power is required to meet the computational demands. Additionally, since 98% of all the computing devices are embedded devices [4], complexity and power efficiency gain prime importance in deploying these systems.

Since the most compute intensive part of multimedia systems is usually image and video processing (taking more than 70% of the total complexity [5]), therefore, optimizing the image or video processing pipelines is commonly targeted for optimizing the design of multimedia systems. As shown in Fig. 1.1a, the "time spent in front of the screen" for different viewing devices is generally consumed by battery-driven devices (e.g., smartphones); hence, power efficiency is one of the main goals for implementing video systems.

Further, from Fig. 1.1b, more than 50% of the data on the Internet and over the wireless channels is compressed video, suggesting that the majority of data processed on wireless-capable devices is video. It is also predicted that by 2019, about 80% of the data communication via WebRTC will be video. Moreover, many real-world critical applications embed video processing as their core algorithm. Examples are localization and navigation [6, 7], passive tracking, radar imaging, and medical imaging. Developers are even introducing real-time communication protocols within web browsers (see WebRTC [8]). Furthermore, new video processing algorithms were introduced in the near past, which not only require high computational efforts by demanding a higher QoS to maximize user satisfaction, but additionally incur high power/energy penalty. Video processing also influences other parts of the multimedia system; e.g., an improved video compression will result in lesser bits being transmitted and, hence, lesser

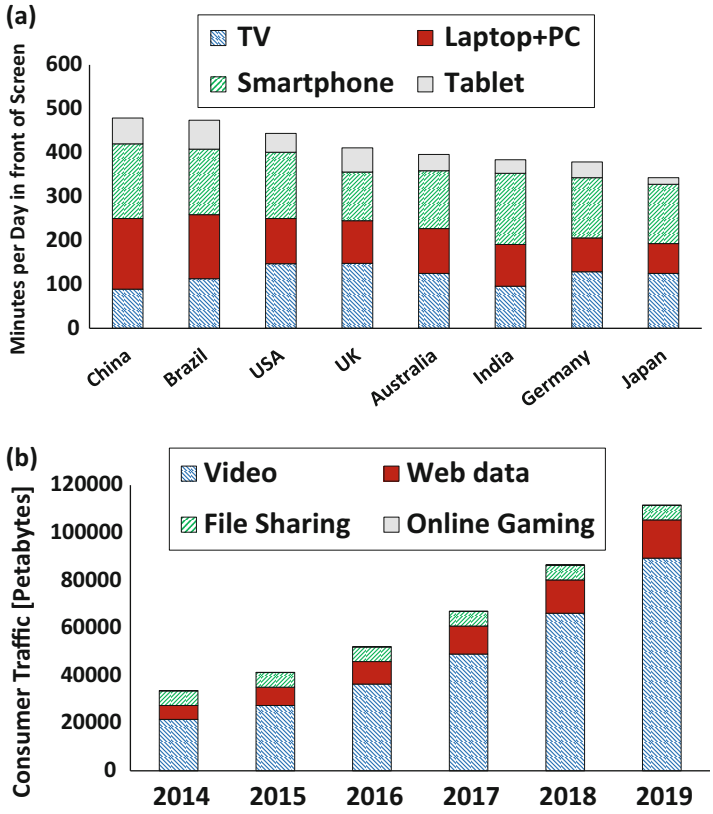


Fig. 1.1 (**a**) Amount of minutes spent in front of display mediums [9], (**b**) Global Internet traffic forecast (from Cisco Visual Network Index, VNI)

transmission energy. Hence, it becomes essential to devise efficient video algorithms and system architectures, to maximize the performance/throughput-per-watt metric when processing video on constraint devices.

The commonly used QoS/SLA metric in video processing is to meet the specified throughput, expressed as frames processed per second (FPS) or frame rate. A video system must be able to meet the throughput demands while providing considerable output video quality (more on this in Chapter 2). This deposits pressure on the underlying hardware to perform with high efficiency, and therefore, multiple architectural novelties and software enhancements are used for deployment, to maximize the throughput-per-watt for divergent set of video applications.

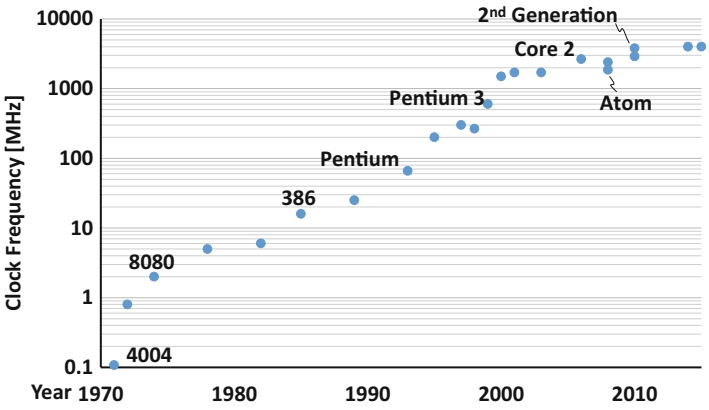


Fig. 1.2 Evolution of Intel processors over the past few decades

1.1.1 Multimedia Processing Architectures

Multimedia systems are implanted in almost every compute domain, ranging from high-end computational servers to small devices like smartphones. Processing devices have evolved over the past few decades and offer more speed, with reduced area footprint and power consumption. The evolution of Intel processor [10] is shown in Fig. 1.2. In 1978, Intel released the 8086 processor consisting of 29 K transistors, capable of running at a 5 MHz. However, the Core-i7 processors with four cores are capable of running at 3.8 GHz in the turbo boost mode. The seventh-generation Intel processors can run at 4.2 GHz. This high operating frequency of processing cores provides an opportunity to sustain high-throughput requirements of multimedia applications.

Similarly, the number of cores on a chip is constantly increasing with every new processor release. For example, Intel has released Polaris (80 cores) and Single-Chip Cloud Computer (SCC, 48 cores) [11] research chips for high-end computing servers. AMD's Opteron chips have up to 24 cores. Tilera Tile-Gx architecture has 100 cores, and Oracle's Sparc T3 houses 16 cores and can process 128 threads for Web services and database applications. IBM has introduced Power and System z architectures with multiple cores for high-speed data processing. Specifically, for video applications, different architectures like Larrabee and Xeon Phi were also launched. Furthermore, it is now common for the new smartphones to have four compute cores. These cores are heavily utilized by video applications. For example, Chrome Web browser (commonly used for viewing multimedia streams) tries to utilize all available cores on an Android-based smartphone. The YouTube Android application uses four cores on average on an eight-core processor. Moreover, Android games normally produce tens of threads and utilize most of the cores available on the system [12]. In summary, the compute power of processors is

increasing with every new generation, to sustain high-throughput expectations of newer video processing algorithms.

Usually, the compute power provided by general-purpose, software-programmable cores is not enough to meet the QoS or to meet the throughput-per-watt metric requirement. Moreover, their power consumption is high and, thus, makes them unsuitable for implementation in small, constrained systems like video enabled wireless sensor nodes or devices used in Internet of Things (IoT). Therefore, video systems are also implemented using custom hardware. Prominent examples are graphical processing units (GPUs) [13, 14], field-programmable gate arrays (FPGAs) [15, 16], and application-specific integrated circuits (ASICs) [17, 18]. Such systems increase the performance of the video application by many folds and reduce power considerably, thus increasing the throughput-per-watt ratio. Unfortunately, current computer-aided design (CAD) tools for custom video system hardware design and implementations do not offer low time to market. In addition, such designs are inflexible and require major debugging efforts, compared to their software counterparts. To exploit the advantages offered by both software- and hardware-based designs, custom hardware accelerators are designed to replace high-complexity software subroutines [19, 20]. These accelerators can be implemented on FPGAs, and they process data in conjunction with software-programmable cores. This increases the throughput of the video application while still maintaining the flexibility offered by the software solutions. Current industrial trends point towards a joint future of software and hardware implementation to offer both flexibility and performance, like Intel's acquisition of Altera and the release of a chip (Intel Atom E600C) containing Intel's x86 core with Altera's FPGA. An additional step in the same direction is to use reconfigurable fabric for using different types of hardware accelerators of the applications, in a time-multiplexed manner [21, 22]. However, the complexity of the design increases accordingly.

1.2 Fundamentals of Video Processing

A video is a concatenated set of temporally captured images or frames of the scene under consideration. A video frame of width w and height h consists of $w \times h$ pixels or addresses, and each pixels stores a video frame sample (the data used for displaying). For display, a video frame sample can be represented by 8-bits or up to 24-bits. Video algorithms perform either pixel-wise or block-based processing on samples of the video frames. Pixel-wise processing only process a single pixel at a time and may process this pixel using information from the neighboring pixels (exploiting spatial correlation) or using the same pixel in the previous frames (exploiting temporal correlation). The pixel(s) at the same location in the previous frame(s) is termed as the collocated pixel of the current pixel under consideration. Block-based processing considers a group of pixels at a single time instance and employs the same principles of spatial and temporal correlation.

Broadly, video processing can be classified as:

1. Online (or real-time) video processing, by capturing video samples directly from video cameras, or after decrypting and decoding the video packets received via wired or wireless links. Such video applications are also termed as streaming applications. In this situation, both the video processing algorithm and the system's architecture should be capable of handling the throughput requirement (i.e., FPS) in real time. Hence, such systems may require high compute power. Such applications might use TCP/UDP ports and shared memory, to interact with other applications.
2. Offline (batch processing) video processing, by reading a video file from the disk. Here, the throughput requirement may or may not be imposed. However, a highly efficient system both in computation and power domains is still desirable as it will increase the user satisfaction and lower the costs. For example, YouTube receives 300 h of videos per minute (statistics from October 2016), and a faster upload and process time at YouTube servers is desirable. However, about four billion videos are viewed per day on YouTube [23], and YouTube must ensure that the video content is provided to the end-users with high enough throughput to increase their viewing experience.

In both cases, the video frames are usually first stored in the external memory (DRAM) by the OS or the application, and parts of the frame are brought to the on-chip memory (generally implemented as caches or scratchpads using SRAM technology) for processing. External memory is larger (in GBs) but slower, while on-chip memories are usually smaller (few MBs or even KBs) but faster. However, note that the external and on-chip memories do not need to be DRAM/SRAM, as they can also be replaced with nonvolatile memories (NVM) [24, 25]. Additionally, the data transfer from the external to the on-chip memory depends upon the bandwidth supported by the external memory device. The on-chip computing device (e.g., a multi/many-core system, ASIC or FPGA) reads the data from the on-chip memory, processes it, and transmits it back to the external memory.

1.2.1 Video Compression

One of the chief video applications is video compression (encoding or just coding) and decompression (decoding). As noticed from Fig. 1.1b, more than 50% of the data over the Internet is compressed video. This suggests that a video encoder (at the transmission side) and video decoder (at the receiver side) are involved in video communication. For real-time, mobile, battery-driven implementation of video encoder/decoder (jointly called CODEC) systems, different dimensions of the underlying system and algorithms must be considered. For example, the throughput requirements should be reasonable, the compute power required to sustain the throughput must be available, and the algorithms and architecture of the video compression system must not drain the battery at a high rate.

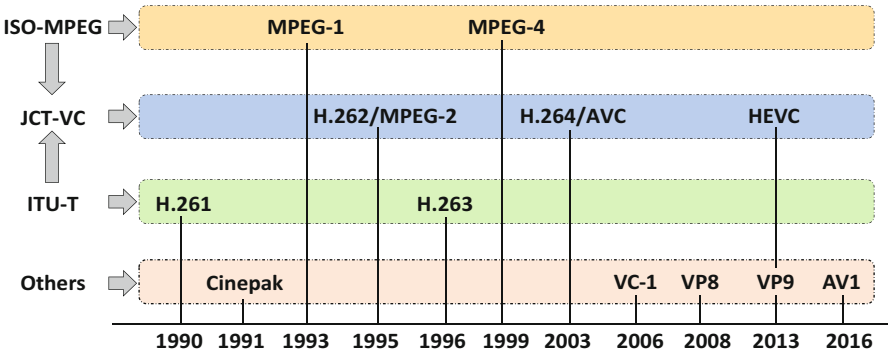


Fig. 1.3 Video CODEC design history of some noteable CODECs

Historically, when the processing power of compute nodes was low (see Fig. 1.2), the throughput demands were comparatively lower, and hence, the complexity of video encoders was small. This is because usually, more effort is required for better/finer compression of the video compared to a coarser compression. In Fig. 1.3, the developmental history of some video encoders is shown. The most popular video encoding standards were released by the ISO-Moving Picture Expert Group (ISO-MPEG) and International Telecommunication Union (ITU) and their joint effort via the Joint Collaborative Team on Video Coding (JCT-VC) [26]. As seen from the figure, numerous video encoding standards were developed, considering the throughput demands (i.e., frame resolution and FPS) of that era. However, as the resolution of display devices improved and the power consumed by these devices reduced, the video resolution and FPS requirements have been steadily increasing. That is, historically, from QCIF video frames (176×144 pixels) at 30 FPS, now we are witnessing demands of 4K video frames (3840×2160) at more than 60 FPS. This increased demand is also motivated by the increased processing capabilities offered by the new fabrication technologies. Therefore, every new generation of video CODECs incorporates new tools and new video processing modules, to enable better compression. In other words, newer encoders try to consume approximately the same channel bandwidth (or bitrate) for increased user demands of QoS. This has steadily led to increase in the computational complexity and power consumption of these video CODECs.

In the past decade, the current industry standard H.264/AVC [27] and next-generation CODEC tipped to overtake H.264/AVC, high efficiency video coding (HEVC, also sometimes called H.265) [1], have emerged as the most prominent video CODECs. These CODECs outperform other parallel video coding efforts (like Google's VP8 and VP9, Daala) [28]. Although, both software and hardware device vendors almost universally support H.264/AVC, HEVC is steadily finding its market. Both these CODECs use block-based frame processing and exploit sample redundancies at both spatial and temporal granularity (details in Sect. 2.2) for compression purpose. These video CODECs employ predictive video coding (PVC) algorithms at the encoder side, and the decoder is a slave of encoder, i.e., the

decoder follows every command of the encoder. For minimal utilization of bandwidth, the encoder searches for the best possible way to compress the video streams. This means the encoder is a high-complexity device while decoder has a relatively low complexity. This makes sense because the number of videos downloaded/played is more than videos uploaded, and small decoders (like tablets and smartphones) can take advantage of a lower complexity decoder. A new paradigm for video coding has also emerged which utilizes the principles of distributed source coding and is termed as distributed video coding (DVC) [29]. In these scenarios, the decoder exploits the correlation between decoded samples to reproduce missing (unprocessed and not transmitted) samples. Such systems are helpful in case of constraint video encoder, where the encoding pressure can be relaxed to save power at the encoder. To combine the advantage of PVC and DVC, hybrid DVC (HDVC) is also proposed [30].

1.3 Design Complexity of a Video System

Advancements in fabrication technologies are steadily increasing the number of transistors per chip. With reducing area of the cores as shown in Fig. 1.4 [31], tens/hundreds of cores and multitude of computational resources on chip are now available for the application designer to implement high-complexity systems. Each new fabrication technology is expected to reduce the minimum feature size by ~0.7×, which roughly translates to a transistor density increase of ~2× [32]. Further, multimedia systems have also advanced due to camera and ADC developments.

As discussed, next-generation video encoders are enhanced to meet the increasing throughput requirements while providing roughly the same video quality. Further, the increasing trend of 3D video acquisition and display has also increased the complexity of video processing systems, because more than one camera is used to capture a scene. The increased resolution and FPS supportable by video cameras and displays put huge pressure on the real-time video encoding system and introduce design complexity challenges. Examples are super-slow motion videos,

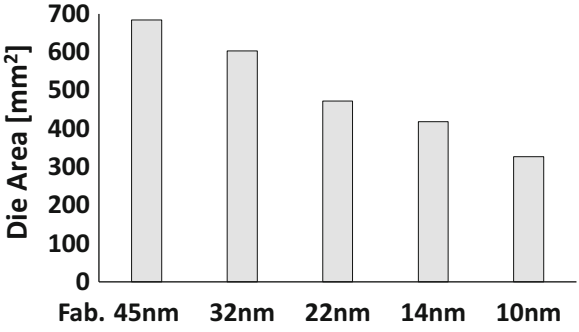


Fig. 1.4 Area of a die for different fabrication technologies [31]

captured at high resolution. Some of these challenges posed by the next-generation multimedia systems are given in Fig. 1.7. Similarly, new video processing algorithms have a higher complexity than their predecessors. For example, HEVC is ~2.65× more complex than H.264/AVC in order to provide ~35% more compression compared to H.264/AVC [33]. Similarly, for HEVC, memory accesses are more than 2× compared to H.264/AVC [34]. The power demands of these applications have also increased owing to fast processing requirements and increased memory accesses. Additionally, the fabrication technology scaling introduces numerous additional challenges for the system designer, some of which are detailed in the next sections.

1.3.1 The Dark Silicon Problem

For new fabrication technologies, the dissipated power is not shrinking at the same rate as area (as seen in Fig. 1.4) due to the failure of Dennard's scaling [35]. Dennard's scaling suggested that the power density would approximately remain constant if the size of the transistors is reduced, i.e., power and area will reduce at the same rate to keep the ratio of power to area almost constant. However, circuit designers are facing challenges to keep up with the voltage scaling to have constant electric field for constant power density and reliability [32]. Moreover, Dennard's scaling assumes increased channel doping to enable shorter channels (for appropriate threshold voltages). On the contrary, this causes an increase in the leakage current. Since Dennard's predictions are no longer valid, therefore, the power density (i.e., power per unit area) is increasing with every fabrication technology, as shown in Fig. 1.5 [31]. Hence, the temperature of the chip might be elevated to a level, which may not be supportable by the silicon and the designed cooling solution. Thus, advanced cooling mechanisms may be required to bring the temperature within safe limits, or the transistors of the chip must be adaptively turned *on* and *off*. In other words, not all of the chip's transistors should be kept *on* at maximum capacity, for 100% of the time [2]. This underutilization of the chip's real estate is generally referred to as dark silicon, which refers to the inefficient consumption of on-chip resources. Therefore, high-throughput applications, like

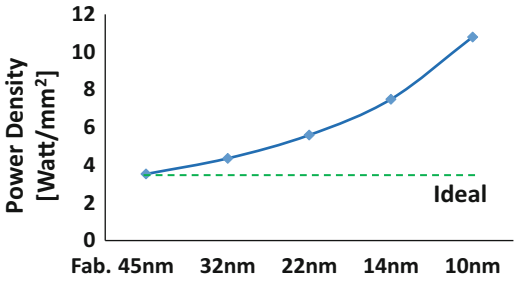


Fig. 1.5 Power density (in Watts per mm^2) for different fabrication technologies [31]

video processing, may not sustain their real-time throughput constraints if implications of dark silicon are not considered. Even a solution designed for a particular system may not be readily applicable to other systems (or for the same system) if there are parallel running workloads. Same arguments also apply for applications exhibiting high workload variations over time.

Moreover, every chip has a thermal design power (TDP) [36], which is the maximum power that can be pumped into the chip. TDP is limited by the physics of the device and the cooling mechanism. More TDP suggests that the cores can run faster and process their assigned workloads quicker. In addition, every core is also assigned a TDP. This tells that video applications cannot arbitrarily run faster and must consider the parallel running applications, as the TDP budget is divided among all parallel applications.

1.3.2 SRAM Aging

Some multimedia systems are in use for long durations, e.g., video processing servers like YouTube using Google File System (GFS) [37], video system aboard space missions, and security/traffic cameras to provide live streams. Such systems have a service lifetime after which they no longer function reliably. One of the reliability concerns for modern systems is the SRAM aging, whereby the sensitivity of the SRAM cells to the noise increases, resulting in increased rate of spurious bit-flips. This is also a direct implication of elevated temperatures of the chip, where, in addition to the dark silicon dilemma, the elevated temperature also reduces the reliability of the system. Phenomenon such as bias temperature instability (BTI) becomes more prominent in the new fabrication technologies. As will be discussed later in the book, video processing algorithms are generally memory intensive and require large on-chip buffers for high performance; hence, such memory phenomenon becomes important for video processing systems.

Negative-bias temperature instability (NBTI) is one of the foremost reliability concerns for SRAMs [38]. NBTI-induced stress on the constituent SRAM transistors results in a higher aging rate, which manifests into an increased delay and increased read errors. It reduces the static noise margin (SNM) of SRAM cells, and thus, the impact of noise increases, which means that the maximum supportable frequency of the system decreases. This trend is shown in Fig. 1.6 where we notice that the circuit delay increases with time and has little impact of operating temperature. However, over the longer run, temperature plays its role and the delay depends mainly upon the temperature. For video processing systems, mechanisms must be devised to either reduce the impact of NBTI or eliminate conditions that increase NBTI. More information about NBTI and its characteristics will be detailed in Sect. 2.3.2.

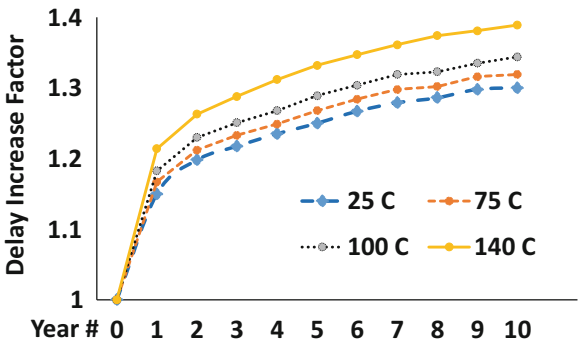


Fig. 1.6 Circuit delay increase factor plotted against time in years [39]

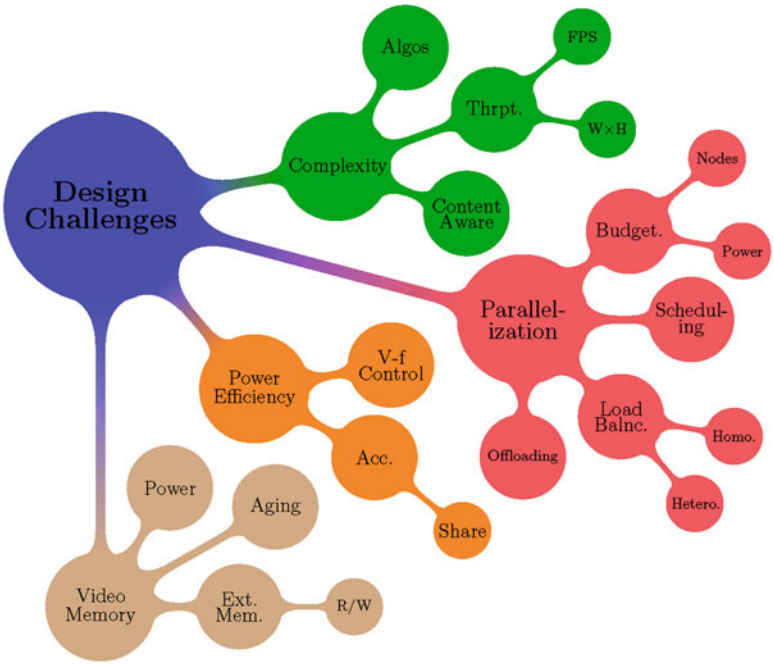


Fig. 1.7 Video system design challenges discussed in this book. In this figure: *Acc* accelerator, *Algos* algorithms, *Arch* architecture, *Hetero* heterogeneous, *Homo* homogeneous, *Pow* power, *Req* requirement, *Thrpt* throughput, *Wrkld* workload

1.4 Video System Design Challenges

The video system design challenges addressed in this book are outlined in Fig. 1.7. These challenges arise due to the throughput requirements fusing with the problems originated by new fabrication technologies. The major design challenge is to maximize the throughput-per-watt metric. For this purpose, both hardware

(architectural) and software (algorithmic) layers of a video system need to be designed while considering the abovementioned paradigms, like dark silicon and NBTI-induced aging. In short, the design of architectural and algorithmic layers of video systems must result in a video system consuming low energy/power, have high throughput, and can reliably operate. These challenges can be classified under the power-efficiency umbrella that can be summarized as:

- Either the power consumption of the system is minimized while sustaining throughput constraints.
- The throughput of the system is maximized under a fixed power budget.

One of the major challenges to address for video processing systems is to reduce their complexity such that real-time, online operations of these video systems can be carried out. This necessitates addressing the throughput requirements by designing efficient processing algorithms with minimal output-quality penalty. To maximally achieve the complexity reduction potential, these algorithms need to leverage the application knowledge. Further, one can achieve additional complexity reduction potential if the video content properties are analyzed (either online or offline). The challenge remains how to determine these properties and accordingly tune the system parameters to increase system's performance. Some of these design challenges are discussed below.

Power Budgeting and Parallelization Almost every complex video processing algorithm allows for parallel processing, to exploit the increasing number of cores on the chip. The challenge is to maximize throughput-per-watt metric by designing appropriate parallelization techniques, to utilize the available number of cores and their characteristics (e.g., maximum frequency). For systems with many cores, it is possible that the workload of the cores differ and (abruptly) vary at runtime. The cores themselves can be heterogeneous. Thus, an additional challenge is to balance the workload among the cores while encountering this variation, to maximally utilize the available hardware and, thus, increase the throughput. Further, the processing tasks must be appropriately packed and scheduled on the cores. For multiple, multithreaded video applications (like multicasting scenarios), the on-chip resources (i.e., the number of cores and the TDP budget) must be distributed among the competing applications with the objective to increase the throughput-per-watt. Otherwise, some applications might lose performance and miss their deadlines.

Power and Computational Efficiency The goal of a video system is to be power and computationally efficient. Thus, for a many-core system, the voltage-frequency settings of the cores cannot be free variables; rather, they must be dependent upon the throughput. One way to achieve power efficiency is to utilize the dark/gray physical areas of the chip by implementing customized hardware logic blocks (like high-throughput accelerators) to off-load workload from the cores. Such a design needs careful regulation. The accelerator must be scheduled among competing cores or applications depending upon the processing requirements of the cores and the throughput requirements of the threads/applications running on these

cores. A failure to do so may result in unfair accelerator distribution and deadline misses. In addition, parts of the systems can be power gated to save power when the hardware accelerators are being utilized. However, the power gating must not hurt the performance of the system because components of the system require a wake-up (or warm-up) time before they can be used again after gating.

Memory Subsystem Design An intelligent memory subsystem uses the external and on-chip memory synergistically such that the accesses to the external memory are reduced and high power efficiency is achieved. In order to reduce the power consumption, the memory subsystem can also employ hybrid memories (uniting volatile memories like SRAM with NVMs) and exploit advantages of different memory technologies. Further, the on-chip SRAM systems will age and reduce the SNM, and the challenge is to design the SRAM subsystem in a manner to lessen the aging rate.

The challenges mentioned above can be broadly classified within different video system layers, as shown in Fig. 1.8. Software layer corresponds to the code maintained by the system designer, OS/kernel, and it handles the complexity of the video system software, parallelization, and its data structures. The user/data layer is responsible for handling the input to the video system. The kernel layer provides hardware interface for the user layer applications. The hardware layer corresponds to the system architecture like memory, network on chips (NoCs), accelerators, and processors. Amalgamation of software and hardware layer relates to hardware accelerators, GPUs, etc., whereby the software and hardware share information without involving the kernel layer and thus may require a codesign.

The challenges mentioned above can be mapped to the software and hardware layers of the multimedia systems as given below and discussed in this book.

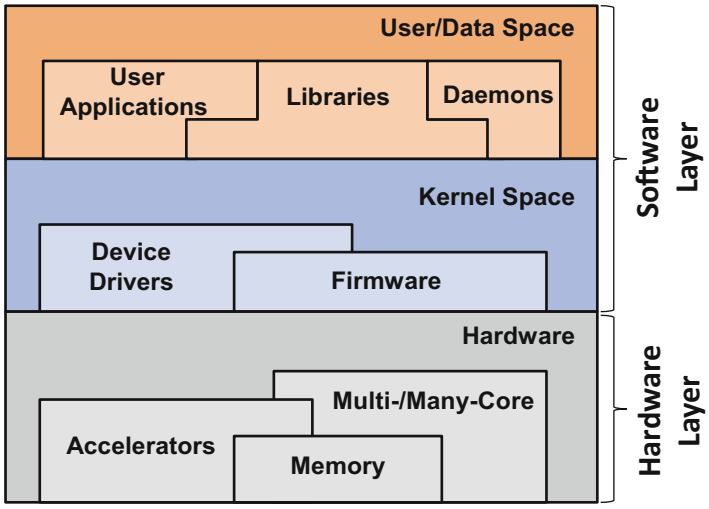


Fig. 1.8 Video system layers. The functionality of a component depends upon the components below it

1.4.1 Software Layer Challenges

The system designer must address the following challenges at the software layer:

- Selecting the appropriate number of parallel computing cores to use on possibly heterogeneous systems, for best output video quality and maximum throughput-per-watt
- Video processing algorithmic improvements by exploiting application- and content-specific properties, to reduce the computational complexity of the systems
- Distributing the TDP among multiple, multithreaded video applications (power budgeting) for fulfilling their QoS goals
- Runtime workload distribution and balancing of parallelized video applications, for maximum utilization of available resources

1.4.2 Hardware Layer Challenges

Following challenges need to be addressed at the hardware layer:

- Selecting appropriate voltage-frequency settings of the cores to maximize the throughput-per-watt metric, while maintaining insignificant output video quality degradation
- Power-efficient application thread scheduling, on the competing cores and the shared hardware accelerators
- Designing hardware accelerators and their efficient integration with the multi- or many-core system
- Power-efficient designs of accelerators and the memory subsystem
- SRAM design with aging rate reduction, for long-term video system deployment

1.5 Limitations of State-of-the-Art

The state-of-the-art techniques for increasing the computational/power efficiency of general and video processing applications will be discussed in detail in Chap. 2. Here, a brief synopsis of their limitations is presented to the reader for quick reference.

Most of the state-of-the-art video system design techniques do not exploit the co-optimization and codesign space of software and hardware layers of the system. Usually, both layers are developed orthogonal to each other and are optimized irrespective of the other's state. These techniques do have an advantage of high applicability and portability, because the system designer is only concerned with optimization of a single layer. Such techniques can reduce the power-efficiency

potential of the system, and fully exploiting this codesign space might lead to much better power efficiency.

At the software layer, mostly, complexity reduction techniques are proposed, which do not consider the underlying hardware attributes. At the hardware layer, different architectural solutions are given. Special purpose hardware like VLIW processors, digital signal processors (DSPs), and graphics processing units (GPUs) are proposed, which can be programmed by the designer. Usually, these components of the system are power hungry. Customized hardware (like ASICs and FPGAs) do present an opportunity for high performance under low power consumption. Typically, these designs ignore throughput adaptation mechanisms or software level control and do not consider the application and/or content knowledge. Similarly, the joint scheduling of cores and hardware accelerators ignores the throughput demands of the applications under consideration. Further, an important aspect of a video system's power efficiency is runtime workload balancing among possibly heterogeneous compute nodes, usually not considered by state-of-the-art.

Additionally, state-of-the-art video systems generally do not consider the effects of dark silicon. Little consideration is given to application-specific optimizations, which may reduce the power saving potential of the video applications. The power-efficiency potential is further reduced by ignoring the unforeseen workload variations of the application(s) or threads of the application(s). Generally, only TDP budgeting or assignment of cores is presented, and workload balancing does not consider the hardware platform properties and power awareness of the system.

Moreover, the aging characteristics of video systems (specifically its memory subsystem) are usually not considered. General SRAM aging reduction techniques provided in the literature involve high power overhead, multiple redundant memory read/writes, inefficient aging reduction techniques, and custom SRAM designs. In addition, these techniques are generally not applicable to on-chip scratchpad memories (memories in control of the programmer).

Summarizing, the complete video system codesign is usually not entertained. The state-of-the-art techniques usually limit themselves to either one of these layers and do not address the challenges imposed by new fabrication technologies. If the software and hardware layers are tuned synergistically, a larger potential of power efficiency is possible, because the designer can employ the application-specific optimization.

1.6 Design and Optimization Methods Discussed in This Book

This book contributes to the design of a video system, by jointly accessing the codesign space of software and hardware layers. The major design and optimization methods are outlined in Table 1.1. The key challenge addressed in this work is to synergistically optimize both the software and hardware layers to maximize the

throughput-per-watt metric of the video system. This way, the software considers the maximum throughput supported by the hardware layer, and the hardware layer sends feedback information to help tune the software layer. Both layers consider the implications of dark silicon and SRAM aging. Effectively, the power and complexity management is performed at a higher abstraction level (i.e., at software layer), and power configuration knobs are provided by the hardware layer. The software layer tunes these knobs selectively and objectively, both at design time and at runtime.

For video system deployment, many-core systems with hardware accelerators and application-specific custom platforms are considered in this book. Different software and hardware layer optimization techniques are presented, by designing algorithms and application-specific hardware accelerators, and their interdependencies are established. For power efficiency, the determination of appropriate number of parallel computing threads (i.e., parallelization) and workload balancing of a video application are presented, which considers the hardware platform properties and throughput demands of the application. This technique is extended to allot the resources on the many-core system (i.e., the compute cores and the hardware accelerators) to the parallel threads. Also, extending this technique, the TDP budget among the competing, multiple multithreaded video applications is distributed. In these scenarios, the voltage-frequency levels of the cores are

Table 1.1 A brief summary of the design/optimization methods in this book

Software layer	Runtime video application parallelization (Sect. 4.1)
	Workload balancing via Dynamic Voltage and Frequency Scaling (DVFS) (Sect. 4.1.1)
	Uniform and non-uniform workload distribution (Sects. 4.2.1, 4.2.2 and 4.2.2.1)
	Self-regulated frequency modeling (Sect. 4.2.5)
	Power-efficient application configuration selection (Sect. 4.3)
	Configuration tuning and mapping for HEVC (Sects. 4.3.1, 4.3.2 and 4.3.3)
	Number of cores and frequency allocation for heterogeneous nodes (Sect. 4.4)
	Resource budgeting for mixed multithreaded video applications (Sect. 4.5)
Hardware layer	Video system I/O and communication (Sect. 5.1)
	Video input pipeline (VIP) (Sect. 5.1.2)
	Custom communication interface (Sect. 5.1.4)
	Accelerator allocation and scheduling (Sect. 5.2)
	Sharing hardware accelerator among multiple applications/threads (Sect. 5.2.1)
	Hardware accelerator sharing for multicasting (Sect. 5.2.2)
	Accelerator architectures for H.264/AVC and HEVC (Sect. 5.3)
	Distributed hardware accelerator architecture (Sect. 5.3.2)
	Hybrid memory architecture (DRAM + SRAM + MRAM) (Sect. 5.4)
	SRAM anti-aging circuits (Sect. 5.5)

adjusted. Additionally, the workload of parallel running applications/threads on a many-core system is adaptively off-loaded to a shared hardware accelerator. The off-loading amount and scheduling is used for both power and throughput optimization. Since video processing can be performed at both software cores and hardware accelerators, we collectively refer to them as compute nodes.

At the hardware layer, this book discusses efficient hardware architectures for video data I/O and communication among compute nodes. The architecture of numerous hardware accelerators is detailed, especially for video coding applications (HEVC and H.264/AVC), whereby the design goal is to target high throughput at little power consumption. Moreover, a hybrid architecture for video scratchpad memories is presented, which uses NVM in conjunction with the SRAM, to reduce memory subsystem power with little throughput penalty. A power-efficient technique is also presented to reduce the aging rate of SRAMs for long-term system deployment.

In the following, we discuss the above techniques in more detail with reference to Table 1.1.

1.6.1 Software Layer Design

1.6.1.1 Power-Efficient Resource Budgeting/Parallelization

Power-Efficient Parallelization Here, the objective is to minimize the power consumption of a video application while meeting the throughput constraints imposed due to the resolution of the video frame and FPS requirements [40]. The power minimization is achieved by minimizing the number of cores used and reducing their frequencies as much as possible. Application's workload and hardware characteristics are used for parallelization and selecting the frequency of the cores. In addition, the video frame is adaptively divided into tiles (disjoint video frame regions), and a thread is used to process each tile. More on tiles will follow in the coming text. The technique presented here tries to balance the workload of the tiles among all the running cores. At runtime, the application-specific workload is fine-tuned using a closed loop for further frequency (and hence power) reduction. This technique not only accounts for the hardware characteristics but also for the system-load variations (due to parallel running applications) and content-dependent complexity. This mechanism enhances the portability of the technique, as it adjusts the number of cores and frequency of the cores at runtime. More information about this method can be found in Sect. 4.2. Similarly, load balancing technique on heterogeneous computing nodes is also discussed [41].

Resource Efficiency To reduce the number of processing cores (or, more generally, processing nodes) and balance the workload, video processing jobs are packed into subtasks, structured in a manner that multiple subtasks can be packed and dispatched to a single compute node, which can process these subtasks in a time-

multiplexed manner [42]. This distribution process accounts for the compute capabilities of the underlying nodes. A job queue is populated with subtask threads and the available cores fetch the subtasks from this queue. Additional information can be looked up in Sect. 4.2.2. Moreover, the same method is extended to distribute processing jobs and balance the workload among heterogeneous computing nodes. For details, refer to Sect. 4.4.

Resource and Power Budgeting This book also discussed the resource and power budgeting of multiple, multithreaded video applications concurrently executed on a many-core platform [43]. Depending upon the throughput requirement, and considering the resource and power utilization history, a hierarchical technique distributes the resources and power among the applications. At runtime, the resources and power allocated to each application are tuned to provide fairness among the applications. For more information, reader is directed to Sect. 4.5.

1.6.1.2 Power-Efficient Software Design

The software layer is also used for software-guided power management. Deadline conscious and high-complexity application (like HEVC and H.264/AVC [44, 33]) are considered for application-specific algorithm design on resource-constrained devices. In these techiques, the number of modes (or searches/trials) performed for generating the best output video quality is adapted, such that the complexity is considerably reduced with minimal video quality degradation. If the complexity of the algorithm is reduced, one can reduce the frequency of the hardware, and thus power reduction will follow. Thus, application-specific properties are leveraged to tune both complexity and power knobs, according to the design's or deployment needs. The complexity reduction techniques presented here are also utilized with workload tuning (see Sect. 1.6.2.1). More information about complexity management can be seen in Sect. 4.3.

1.6.2 Hardware Layer Design

1.6.2.1 Power-Efficient Accelerator Design

Video I/O and Communication Among Heterogeneous Nodes This book presents a communication infrastructure and hardware architecture of a low-latency video I/O. Similarly, a custom interface (communication architecture among the compute nodes) is presented which can be used for processing multiple tasks of the same video-based workload on custom and multi- or many-core hardware platform. The architecture is designed to support the workload distribution and balancing decisions on the multi−/many-core platform. A master core assigns the workload of all the tiles to the secondary cores, using the custom interface. The architecture is

functionally verified on a real FPGA, with multiple, embedded soft cores (i.e., programmable, compute cores on FPGA). More information is given in Sect. 5.1.

Accelerator Design Multiple hardware accelerators are discussed in this book, especially for video coding applications [45, 46]. A distributed, hardware accelerator design methodology is presented which can be used in conjunction with clock-gating (or power-gating) circuits to increase the power efficiency. These accelerators target the most computationally intensive part of HEVC and H.264/AVC, i.e., the evaluation of different modes to determine the right compression technique. For further information, refer to Sect. 5.3 in this book.

1.6.2.2 Shared Hardware Accelerator Scheduling

Power-Efficient Hardware Accelerator Sharing The target of the techniques presented in this domain is power efficiency of a system with multiple cores sharing a hardware accelerator [47]. As will be discussed, the hardware layer provides support to the off-loaded subtasks from the cores to the custom hardware accelerator, thereby reducing computational complexity and power consumption of the complete system. In addition, this method provides opportunities to exploit the dark silicon. Moreover, to fulfill the throughput constraints and minimize the power consumption of the system, off-load-scheduling mechanisms determine the subtasks assigned to the cores and the accelerator. To maximize the throughput, the slack of each parallel running thread/application is minimized. The technique basically formulates an optimization problem and proposes a heuristic to find the optimal percentage of workload off-loaded to the accelerator. The goal is to maximally utilize the accelerator while sustaining the throughput demands of parallel running threads/applications. The same method can be extended to process multiple videos processing workloads. More information is presented in Sect. 5.2.1.

Multicast H.264/AVC Encoder Design In addition, the design of a multicasting and fully customized H.264/AVC video encoder architecture using area- and power-efficient building modules is presented [15, 45]. Hardware is shared among different video encoders (using a hardware scheduler and a rescheduler) such that little or no penalty is incurred. The complete system integration (along with camera input, memory access, and Ethernet output) is presented in Sect. 5.2.2 and Appendix C.

1.6.2.3 Memory Subsystem Design

Hybrid Memory Architecture In this book, a hybrid memory subsystem for video processing is presented, which uses a NVM (specifically, magnetoresistive random-access memory, MRAM) in conjunction with an SRAM [34]. This subsystem exploits the characteristics of both NVM and SRAM such that maximum power efficiency is achieved at minimal or no latency penalty. Large MRAM

buffers are used for storing video frames. The MRAM frame buffer memory is sectored, and these sectors are normally *off*. These sectors are powered *on* only in the case their data is required by the application. An unsupervised learner (self-organizing map, SOM) is used for predicting the next sector that needs to be powered *on*. Thus, this saves leakage power and accesses to the external memory. SRAM FIFOs are used for input/output from MRAM buffers, in order to reduce the latency penalty. For further information, a reader is referred to Sect. 5.4.

SRAM Anti-Aging Architecture This book also presents a design to reduce the aging of SRAM cells, deteriorated by NBTI-induced stress cycles, by adapting the data written to and read from the SRAM [48, 49]. Memory read transducers (MRTs) and memory write transducers (MWTs) are discussed, which adapt the video data bits on the fly, read from and written to the SRAM. MRT and MWT incur minimal latency and avoid extra read/write of the SRAM. More information about SRAM anti-aging is given in Sect. 5.5.

1.6.3 Open-Source Tools

We also discuss some open-source video processing tools, which will be helpful to the research and technical community and give them a head start in designing power-efficient video systems. The intention is that these tools can be used for testing and extending the techniques discussed in this book or for testing new design paradigms.

Parallel HEVC Encoder For testing the parallelization, workload balancing, and resource allocation methods, a C++ − based, multithreaded, open-source, HEVC video encoder called ces265 [40] is presented. This encoder is lightweight and highly flexible, e.g., the designer can introduce different parallelization and tile structure settings. In addition, a thread of ces265 is about ~13× faster than the reference encoding software (HM-9.2). Further, the NIOS-II multi-core Altera FPGA implementation for this encoder is also provided. More information about this tool is available in Appendix B.

SRAM Aging Analysis GUI For video memory aging analysis and visualization, a GUI-based tool is presented. Written in C#, the tool can display videos and one can implement and evaluate multiple anti-aging circuits. This tool detects memory regions under high stress and automatically plots the stress regions, histograms, and other statistics. More information and download instructions for this tool can be found at: http://ces.itec.kit.edu/EnAAM.php

1.7 Book Outline

This book is structured in the following manner (more information can be found in Table 1.1).

- Chapter 2 provides detailed background and state-of-the-art on the design and optimization challenges. Essential background knowledge of video applications is also given to get the reader accustomed with the jargon and the methodologies discussed in this book. The dark silicon paradigm is presented in conjunction with video systems, when implemented on many-core and customized hardware platforms.
- Chapter 3 provides a comprehensive video system design by integrating different components of the system, which can result in high throughput-per-watt. The communication flow between the software and hardware layers is discussed, and the architectural design and its assumptions are outlined. Several motivational analyses are performed for parallelization, workload variation, and memory subsystem's power and aging.
- Chapter 4 discusses the video system software layer of Chap. 3 in detail. For mixed, multithread video workloads, power-efficient parallelization and resource budgeting are presented. Various complexity reduction methods are discussed.
- Chapter 5 relates to the video system hardware layer of Chap. 3. It discusses the shared hardware scheduling among competing cores and for multicasting scenario. The hybrid memory architecture design is also given, along with power-efficient, SRAM anti-aging circuits.
- Chapter 6 presents experimental evaluations of the techniques discussed in Chap. 4 and 5, while Chap. 7 concludes the book with an outlook of the future extensions.
- The algorithms used in the design and deployment methodologies presented in this book are given in Appendix A. The inner workings of the ces265 video encoder are presented in Appendix B, while Appendix C discusses the HDL of proposed multicasting H.264/AVC video encoder.

References

1. Sullivan, G. J., Ohm, J., Han, W., & Wiegand, T. (2012). Overview of high efficiency video coding. *IEEE Transactions on Circuits and Systems for Video Technology, 22*(12), 1649–1668.
2. Esmaeilzadeh, H., Blem, E., Amant, R., Sankaralingam, K., & Burger, D. (2011). Dark silicon and the end of multicore scaling. In *International Symposium on Computer Architecture (ISCA)*.
3. Henkel, J., Bauer, L., Dutt, N., Gupta, P., Nassif, S., Shafique, M., Tahoori, M., & Wehn, N. (2013). Reliable on-chip systems in the nano-era: Lessons learnt and future trends. In *Design Automation Conference (DAC)*.

4. ARTEMIS, [Online]. Available: http://www.artemis-ju.eu/embedded_systems. Accessed 20 Oct 2015.
5. Shafique, M. (2011). Architectures for adaptive low-power embedded multimedia systems. Karlsruhe Institute of Technology (KIT).
6. Nister, D. (2004). Automatic passive recovery of 3D from images and video. In *International Symposium on 3D Data Processing, Visualization and Transmission*.
7. Cutrona, L., Leith, E., Porcello, L., & Vivian, W. (1966). On the application of coherent optical processing techniques to synthetic-aperture radar. *Proceedings of the IEEE, 54*(8), 1026–1032.
8. WebRTC. [Online]. Available: http://www.webrtc.org/. Accessed 26 Aug 2015.
9. AdReaction. *Marketing in a multiscreen world* [Online]. Available: https://www.millwardbrown.com/adreaction/2014/report/Millward-Brown_AdReaction-2014_Global.pdf. Accessed 12 July 2017.
10. Intel chips through the years. (2015, September 12). [Online]. Available: http://interestingengineering.com/intel-chips-timeline/. Accessed 5 Oct 2015.
11. Held, J. (2010). Introducing the single-chip cloud computer: exploring the future of many-core processors. In *Intel White Paper*.
12. Pathania, A., Pagani, S., Shafique, M., & Henkel, J. (2015). Power management for mobile games on asymmetric multi-cores. In *Low Power Electronics and Design (ISLPED)*.
13. Momcilovic, S., Ilic, A., Roma, N., & Sousa, L. (2014). Dynamic load balancing for real-time video encoding on heterogeneous CPU+GPU systems. *IEEE Transactions on Multimedia, 16*(1), 108–121.
14. Xiao, W., Li, B., Xu, J., Shi, G., & Wu, F. (2015). HEVC encoding optimization using multi-core CPUs and GPUs. *IEEE Transactions on Circuits and Systems for Video Technology (TCSVT), 9*(99), 1–14.
15. Khan, M., Shafique, M., Bauer, L., & Henkel, J. (2015). Multicast FullHD H.264 intra video encoder architecture. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* pp. 1–5.
16. Benkrid, K., Crookes, D., Smith, J., & Benkrid, A. (2000). High level programming for real time FPGA based video processing. In *Acoustics, Speech, and Signal Processing (ICASSP)*.
17. Iwata, K., Mochizuki, S., Kimura, M., Shibayama, T., Izuhara, F., Ueda, H., Hosogi, K., Nakata, H., Ehama, M., Kengaku, T., Nakazawa, T., & Watanabe, H. (2009). A 256 mW 40 Mbps Full-HD H.264 high-profile codec featuring a dual-macroblock pipeline architecture in 65 nm CMOS. *IEEE Journal of Solid-State Circuits, 44*(4), 1184–1191.
18. Zhou, J., Zhou, D., Fei, W., & Goto, S. (2013). A high-performance CABAC encoder architecture for HEVC and H.264/AVC. In *International Conference on Image Processing (ICIP)*.
19. Henkel, J., & Yanbing, L. (1998). Energy-conscious HW/SW-partitioning of embedded systems: a case study on an MPEG-2 encoder. In *International Workshop on Hardware/Software Codesign*.
20. Zuluaga, M., & Topham, N. (2009). Design-space exploration of resource-sharing solutions for custom instruction set extensions. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), 28*(12), 1788–1801.
21. Grudnitsky, A., Bauer, L., & Henkel, J. (2014). COREFAB: Concurrent reconfigurable fabric utilization in heterogeneous multi-core systems. In *International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*.
22. Majer, M., Teich, J., Ahmadinia, A., & Bobda, C. (2007). The erlangen slot machine: A dynamically reconfigurable FPGA-based computer. *VLSI Signal Processing Systems for Signal, Image, and Video Technology, 47*(1), 15–31.
23. Smith, C. 120 Amazing YouTube statistics. [Online]. .Available: http://expandedramblings.com/index.php/youtube-statistics/. Accessed 5 Oct 2015.
24. Dong, X., Wu, X., Sun, G., Xie, Y., Li, H., & Chen, Y. (2008). Circuit and microarchitecture evaluation of 3D stacking magnetic RAM (MRAM) as a universal memory replacement. In *Design Automation Conference (DAC)*.

25. Wu, X., Li, J., Zhang, L., Speight, E., Rajamony, R., & Xie, Y. (2009). Hybrid cache architecture with disparate memory technologies. In *International Symposium on Computer Architecture (ISCA)*.
26. Joint Collaborative Team on Video Coding (JCT-VC), ITU, [Online]. Available: http://www.itu.int/en/ITU-T/studygroups/2013-2016/16/Pages/video/jctvc.aspx. Accessed 7 Oct 2015.
27. Ostermann, J., Bormans, J., List, P., Marpe, D., Narroschke, M., Pereira, F., Stockhammer, T., & Wedi, T. (2004). Video coding with H.264/AVC: Tools, performance, and complexity. *IEEE Circuits and Systems Magazine, 4*(1), 7–28.
28. Grois, D., Marpe, D., Mulayoff, A., Itzhaky, B., & Hadar, O. (2013). Performance comparison of H.265/MPEG-HEVC, VP9, and H.264/MPEG-AVC encoders. In *Picture Coding Symposium (PCS)*.
29. Girod, B., Aaron, A. M., Rane, S., & Rebollo-Monedero, D. (2005). Distributed Video Coding. *Proceedings of the IEEE, 93*(1), 71–83.
30. Dufaux, F., & Ebrahimi, T. (2010). Encoder and decoder side global and local motion estimation for Distributed Video Coding. In *International Workshop on Multimedia Signal Pro-cessing*.
31. Huang, W., Rajamani, K., Stan, M., & Skadron, K. (2011). Scaling with design constraints: Predicting the future of big chips. *IEEE Micro, 31*(4), 16–29.
32. Bohr, M. (2007). A 30 year retrospective on dennard's mosfet scaling paper. *IEEE Solid-State Circuits Society Newsletter, 12*(1), 11–13.
33. Khan, M., Shafique, M., & Henkel, J. (2013). An adaptive complexity reduction scheme with fast prediction unit decision for HEVC intra encoding. In *International Conference on Image Processing (ICIP)*.
34. Khan, M., Shafique, M., & Henkel, J. (2013). AMBER: Adaptive energy management for on-chip hybrid video memories. In *International Conference on Computer-Aided Design (ICCAD)*.
35. Dennard, R., Rideout, V., Bassous, E., & LeBlanc, A. (1974). Design of ion-implanted MOSFET's with very small physical dimensions. *IEEE Journal of Solid-State Circuits, 9*(5), 256–268.
36. Huck, S. (2011). Measuring processor power, TDP vs. ACP. Intel.
37. Ghemawat, S., Gobioff, H., & Leung, S.-T. (2003). The google file system. In *ACM Symposium on Operating Systems Principles*.
38. Kumar, S., Kim, C., & Sapatnekar, S. (2006). Impact of NBTI on SRAM read stability and design for reliability. In *International Symposium on Quality Electronic Design (ISQED)*.
39. Gnad, D., Shafique, M., Kriebel, F., Rehman, S., Sun, D., & Henkel, J. (2015). Hayat: Harnessing dark silicon and variability for aging deceleration and balancing. In *52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*.
40. Khan, M. U. K., Shafique, M., & Henkel, J. (2014). Software architecture of high efficiency video coding for many-core systems with power-efficient workload balancing. In *Design, Automation and Test in Europe*.
41. Khan, M. U. K., Shafique, M., Gupta, A., Schumann, T., & Henkel J. (2016). Power-efficient load-balancing on heterogeneous computing platforms. In *IEEE/ACM 19th Design, Automation and Test in Europe Conference (DATE)*.
42. Shafique, M., Khan, M. U. K., & Henkel, J. (2014). Power efficient and workload balanced tiling for parallelized high efficiency video coding. In *International Conference on Image Processing*.
43. Khan, M. U. K., Shafique, M., & Henkel, J. (2015). Hierarchical power budgeting for dark silicon chips. In *International Symposium on Low Power Electronics and Design*.
44. Khan, M. U. K., Shafique, M., & Henkel, J. (2014). Fast hierarchical intra angular mode selection for high efficiency video coding. In *International Conference on Image Processing*.
45. Khan, M. U. K., Borrmann, J. M., Bauer, L., Shafique, M., & Henkel, J. (2013). An H.264 Quad-FullHD low-latency intra video encoder. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*.

46. Khan, M. U. K., Shafique, M., & Henkel, J. (2013). Hardware-software collaborative complexity reduction scheme for the emerging HEVC intra encoder. In *Design, Automation and Test in Europe (DATE)*.
47. Khan, M. U. K., Shafique, M., & Henkel, J. (2015). Power-efficient accelerator allocation in adaptive dark silicon many-core systems. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*.
48. Shafique, M., Khan, M. U. K., Tüfek, O., & Henkel, J. (2015). EnAAM: Energy-efficient anti-aging for on-chip video memories. In *Design Automation Conference (DAC)*.
49. Shafique, M., Khan, M. U. K., & Henkel, J. (2016). Content-aware low-power configurable aging mitigation for SRAM memories. *IEEE Transactions on Computers (TC), 65*(*12*), *3617–3630*.

Chapter 2
Background and Related Work

This chapter discusses the basics of video processing in general, while specifically targeting the video coding applications. General video system design and its memory access patterns and resource utilization are deliberated. Fundamentals of HEVC and H.264/AVC video encoding are followed by their associated challenges when designing computationally efficient video processing systems. Modern technological challenges that arise in deploying video systems are also presented in this chapter. Afterwards, the state-of-the-art techniques to meet these design challenges are discussed, with details targeting video processing system's software and hardware layers.

2.1 Overview of Video Processing

The working of a video processing system largely depends upon its complexity. The complexity of a video system can be roughly correlated to two characteristics, processing algorithm and throughput constraint. As discussed in Sect. 1.2, the computational complexity of video processing depends upon the type of the algorithm, along with the video frame dimensions and the FPS requirements (given by f_p). From FPS requirements, the maximum time that can be spent on processing a single video frame is given by $t_{frm}=1/f_p$. A general metric to present the throughput requirements is given by $w \times h \times f_p$, which denotes the number of pixels that must be processed in one second, for a frame of size $w \times h$ pixels. However, most video algorithms process a block of pixels in a time unit, i.e., all the pixels within the block correspond to a particular computational mode. An example of dividing a video frame into blocks of size $b_w \times b_h$ is shown in Fig. 2.1.

The authors would like to point out that this work was carried out when all the authors were in Department of Computer Science, Karlsruhe Institute of Technology, Karlsruhe, Germany.

M.U.K. Khan et al., *Energy Efficient Embedded Video Processing Systems*,
DOI 10.1007/978-3-319-61455-7_2

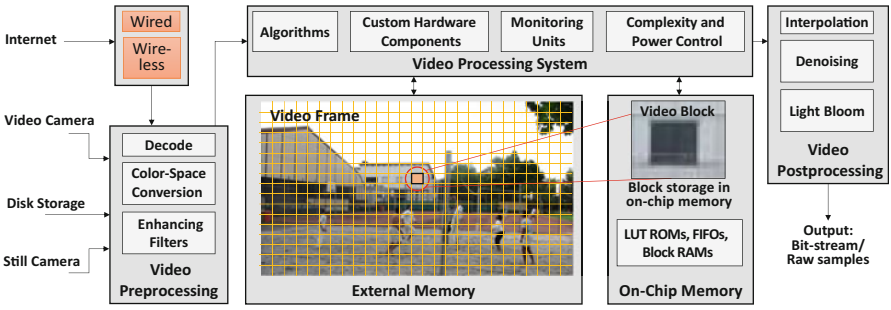


Fig. 2.1 Fundamentals of a video processing system. The video frame is divided into blocks and logically stored in the external memory as shown in the figure. A block of size $b_w \times b_h$ is read at a time from external memory and brought to the on-chip memory for processing

Generally, the video is processed online or offline. Further, the frames themselves are stored in the external memory due to the size of the video frame. A block (or a group of blocks) is transferred from the external memory to the on-chip memory for processing, because accessing the on-chip memory is faster. Once processed, the output of the processed block is concatenated to the output of other blocks.

Figure 2.1 presents an example of video processing system, with pre- and post-processing modules. For video system design discussion, naïve online gaming scenario can be considered. The video application reads the data (possibly compressed video) from the gaming server, decodes it, and then applies different image enhancement techniques before finally forwarding it to the main processing engine. The processing engine may also use data on the local disk and video content captured using the camera(s) at the user's end for processing. Afterwards, this data is displayed on the display device, using post-processing filters like light blooming, denoising, and interpolation. Further, the data/video formed at the user's end is also encoded and encrypted and finally sent back to the gaming server. Note that pre- and post-processing modules themselves can be separate video processing systems, employing the same principles as given in this figure. Further, these modules can be deployed on independent hardware resources, e.g., video preprocessing on CPUs and post-processing on GPU.

For block-based video processing, the throughput requirement can be written as:

$$n_{\text{sec}} = n_{frm} \times f_p = \frac{w \times h}{b_w \times b_h} \times f_p \tag{2.1}$$

Here, n_{sec} presents the number of blocks that must be processed per second, and n_{frm} is the number of blocks within a frame. A larger n_{sec} corresponds to more computational and power requirements and vice versa.

A video sample can be stored in multiple formats. Usually, a sample is stored as a union of three colors, red (R), green (G), and blue (B). This color space format is termed as RGB format. If intensity of each color of video samples is presented by 8-bits, then a sample is stored as a 24-bit value. Usually, R, G, and B "planes" of the

frame are stored separately. Therefore, the number of addresses (or pixels) is three times more. Therefore, the size of the frame can be presented in number of bits b_{frm} by:

$$b_{frm} = w \times h \times 3 \times \text{bits\_per\_sample} \tag{2.2}$$

The RGB color format carries a lot of redundancy and therefore burdens the storage/communication. By focusing predominately on the features which are visually more distinguishable to the human user, this redundancy can be removed. One prime method is to transform RGB to a different space. For example, a large number of video algorithms work with the YCbCr format (also sometimes called YUV format), where Y presents the luminance component of the sample, and Cb and Cr collectively determine the chrominance of the sample. YCbCr 4:4:4 format means that for every luminance pixel, there are two chrominance pixels, whereas YCbCr 4:2:0 format means that the two chrominance pixels are associated with four luminance pixels. Hence, both the chrominance planes are downsampled by two, in horizontal and vertical directions. This still results in high visual quality because the human eye is more susceptible to the luminance component than the chrominance components. For YCbCr 4:2:0 case, the size of the frame (in bits) is denoted by:

$$b_{frm} = w \times h \times (1 + 0.25 + 0.25) \times \text{bits\_per\_sample} \tag{2.3}$$

In Fig. 2.2, an example baseline architecture of a real-time camera-based image/video processing system is illustrated. The camera captures videos in real time at a certain frame acquisition rate in terms of frames per second (typical values are 30, 60, etc.). The raw data (i.e., video samples) is preprocessed via a video input pipeline (VIP). The output of VIP is the streaming data, containing YCbCr 4:2:0 samples, which is converted into words of size ≥ 8 bits using a combination of a FIFO and a shift register. Each frame is stored in a separate location within the main memory. The memory is composed of multiple frame memory partitions in order to simultaneously store multiple video frames and support double or triple buffering. Write address generation unit (AGU) is used to write video frames to the memory partitions, and frames are read via Read AGU. The application requests a new

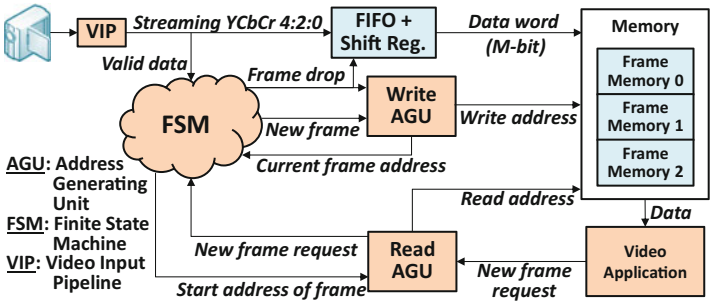


Fig. 2.2 Video capture, store and access management of a video processing system

frame once it has processed the previous frame. Frame drop mechanism (i.e., previously stored frames are overwritten, although they are not processed) is also supported, in case the video processing application has high complexity and it cannot cope with the high frame capture rate of the camera. Other factors that might induce frame-dropping are e.g., constrained underlying hardware with lower clock frequency, lesser CPUs, smaller caches, low bandwidth of wired/wireless connections etc. Frame drops can also occur if the output of the video system is stalled (e.g., scenarios like high congestion and packet loss in wired/wireless output scenarios, the output disk is full).

Generally, small parts of the video processing code (called kernels) take a hefty portion of computational complexity and power. In order to reduce the complexity, these kernels are optimized by developing past-predicts-future paradigms and by tuning the complexity knobs. For example, the number of modes computed to search for the best mode can be reduced, which increases the throughput-per-watt metric, but may reduce the output video quality. These kernels can also be implemented as custom hardware accelerators.

In addition to video applications, other applications are also sometimes referred to as "frame-based" applications. The term "frame-based" application broadly denotes an application that needs to process a set of data periodically, within a specific interval of time.

2.1.1 Intra- and Inter-frame Processing

Broadly, there are two distinct types of video processing algorithms employed by video systems:

1. Intra-frame processing, where the spatial neighborhood of the pixel (or block) under consideration is used for computations. That is, the same frame is used for processing the current pixel/block.
2. Inter-frame processing, where the temporal neighborhood of the pixel (or block) under process is used for computations. That is, the frame(s) processed in the past or future is used in algorithms involving the current pixels/block processing. The frames used for interframe processing are also called reference frames.

The neighborhood concept is illustrated in Fig. 2.3, where a block under consideration has spatial neighbors within the frame and temporal neighbors across the frames. The same concept is applicable to pixel-based processing; however, our main focus in this book will be block-based processing. The spatial and temporal neighbors are exploited for multiple purposes, e.g., noise filtering and motion tracking. The correlation of the current pixel/block with its neighbors plays an important role in determining the video quality, as will be discussed in the coming text. High correlation in the neighborhood usually translates to higher video quality (e.g., better noise removal if neighbors correlate highly with the current block). Mostly, the spatial neighbors are not as highly correlated with the current block; therefore, the video output quality is not high for intra-frame processing, compared

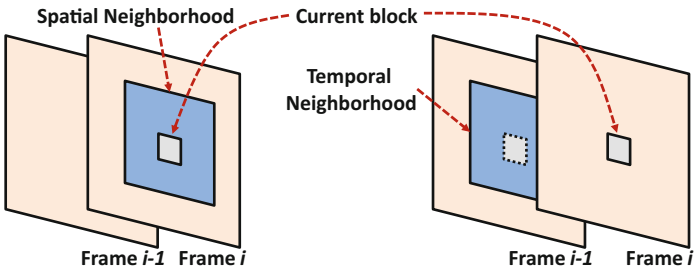


Fig. 2.3 Spatial and temporal neighborhood of current block under consideration

to inter-frame processing. However, the computational requirement of inter-frame processing is higher due to a large amount of data being transferred between external and on-chip memory. This increases the data processing pressure and the power overhead. On the other hand, a larger complexity corresponds to searching for the best mode of computations and hence increases the output video quality. The output video quality can be compared against an ideal output using peak signal-to-noise ratio (PSNR) and Bjøntegaard delta peak signal-to-noise ratio (BD-PSNR) or bitrate (BD-Rate) [1]. This will become clear when we discuss video coding overview in Sect. 2.2.

Usually, the on-chip memory shown in Fig. 2.1 is large enough to hold additional data, and therefore, in addition to the video frame block, its spatial or temporal neighbors can also be stored on-chip. This further reduces the stress on the memory subsystem.

2.2 Overview of Video Coding

Here, we discuss video coding algorithms with reference to the principles mentioned in Sect. 2.1. This section introduces the working principles of H.264/AVC [2] and HEVC [3] video encoders. The working principles of a video encoder are diagrammatically shown in Fig. 2.4. A video frame block (of size $b_w \times b_h$, with samples s) is brought from the external memory to the on-chip memory and is compressed by searching for the best compression mode/configuration, by iteratively testing the intra- and inter-compression modes (also referred to as intra- and inter-predictions). Afterwards, the best compression mode is forwarded to the additional video compression modules. The purpose of both intra- and inter-prediction modes at the encoder is to generate a resembling representation of the current block under test, called prediction (with samples s'). Therefore, the module in Fig. 2.4 is referred to as prediction generator. This prediction is generated using the neighbors of the block. The idea is that since the neighbors will already be present at the decoder, the decoder can recreate the current block using only the information of the current block sent by the encoder. Usually, number of bits for representing the current block is smaller than the number of information bits send

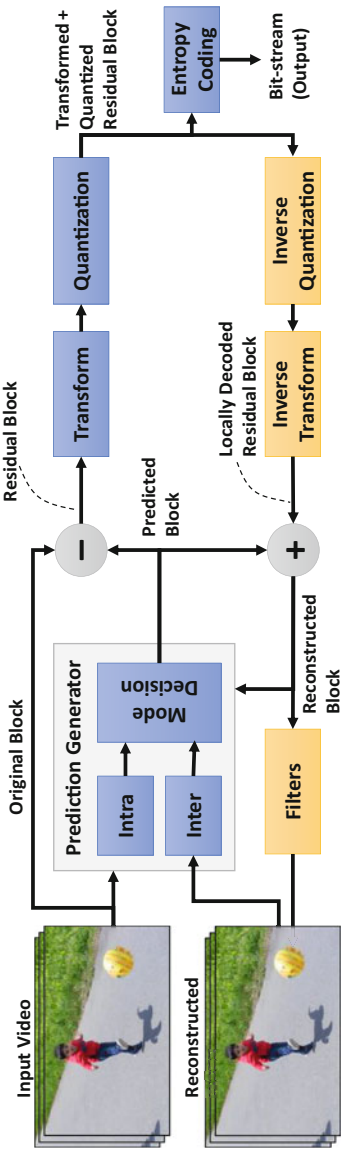


Fig. 2.4 Basic modules of a video encoder. The *blue* boxes are used for encoding purposes, and the *orange* boxes denote the local decoder modules used inside the encoder

by the encoder (and hence, compression is achieved). However, this technique only works with lossless video encoders, which do not provide higher compression rates compared to lossy compression. Therefore, in case of lossy compression, the encoder compresses the current block by using the neighborhood of the block in the reconstructed video frame samples. A reconstructed video frame is a video frame generated by decoding the information sent by the encoder. The encoder can also locally decode and use the locally decoded video frame for generating the predictons, i.e., the video frame samples which will be generated at the decoder, to make sure that there is no error accumulation at the decoder. Thus, the encoder also implements a local decoder (shown by the lower modules in Fig. 2.4). The generation of prediction either uses the spatially neighboring pixels (intra-frame processing) or temporally neighboring pixels (inter-frame processing). Thus, if the quality of prediction (its resemblance with the original block) is high, the residual block, i.e., the difference between the original and the prediction, has low pixel energy (samples with low values). The quality of a prediction is usually tested using the sum of absolute differences (SAD) metric:

$$SAD = \sum_{y=0}^{bh-1} \sum_{x=0}^{bw-1} \left| s(x,y) - s'(x,y) \right| \tag{2.4}$$

Here, (x, y) is the pixel location within the original and predicted blocks. Every intra- and inter-mode has its associated prediction and, therefore, a SAD value. It is evident that, if the number of predictions that are tested is high (i.e., a more complex encoder), probability of finding a prediction with a low SAD/pixel energy is high, which results in higher compression.

Afterwards, transformation of residue and its quantization take place. The purpose of transformation is to separate low- and high-frequency components of the residual bock and transmit a part of these components (hence, lossy compression). If high compression is required, the higher frequency components are not sent to the decoder and vice versa. This is because the human eye is more susceptible to the low-frequency image components than high-frequency components. Quantization further reduces the scale of the transformed values before finally the bitstream generation (i.e., the entropy coding module) forms a bitstream which is sent to the video decoder. The size of the bitstream is proportional to the pixel energy in the residual block, and more energy in the residual block will increase the size of the bitstream.

This bitstream consists of the transformed and quantized residue of the current block, along with the prediction mode selected for compressing the block. The decoder can reverse the process, by generating the transformed and quantized residual block (approximately) from the bitstream. The steps to generate the block at decoder are (a) decoding the bitstream to get quantized blocks and prediction mode, (b) inverse quantizing the received block, (c) inverse transforming the block, (d) generating the prediction using the prediction mode, and, finally, (e) adding the prediction to generate a representation of the original block.

In addition to video encoding/decoding algorithms (which attract high attention from research community), algorithmic and architectural designs of video input and

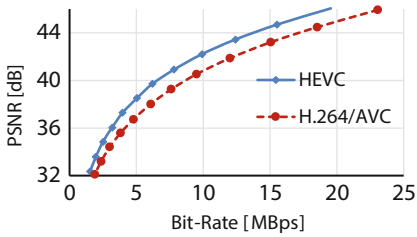


Fig. 2.5 Video quality comparison for H.264/AVC and HEVC

output pipelines are also a critical design step, since if a module within these pipelines becomes the bottleneck, it will hurt the performance of the complete system. Video input pipeline involves color space conversion, deinterlacing, downsampling, etc. More on this is covered in Sect. 5.1.2. Video output pipeline usually incorporates noise reduction algorithms, color space conversion, and content enhancement (like increasing the sharpness, correcting brightness, etc.).

2.2.1 H.264/AVC and HEVC

H.264 or MPEG-4 Part 10 or Advanced Video Coding (MPEG-4 AVC) is the current video compression industry standard developed by JCT-VC [2, 4]. In this book, this standard will be referred to as H.264/AVC. Here, a video frame is divided into blocks of 16×16 pixels called macroblocks (MBs), and each MB is treated as a separate entity for compression. A block is tested with intra-prediction modes using angular directions and inter-prediction via block matching (also called motion estimation (ME)) [5, 6]. However, due to increasing video resolutions and frame rates (e.g., 8K Ultra-HD, 7680×4320 pixels, at 120 fps), JCT-VC has also recently developed the next-generation High Efficiency Video Coding (HEVC) standard. HEVC aims at increasing the compression by 50% compared to the H.264/AVC. From Fig. 2.5, we notice that the bitrate of the video encoded via HEVC is lower than H.264/AVC, for the same video quality (PSNR). These curves are also called rate-distortion (RD) curves. For these curves, higher is better.

On the other hand, HEVC time consumption is substantially higher and our experiments show that HEVC is ~2.2× more complex than H.264/AVC. Compared to H.264/AVC, HEVC brings two major innovations [3, 7]. Firstly, unlike the concept of MB of its predecessor coding standard H.264/AVC, the HEVC introduces the concept of the coding tree unit (CTU) as a coding structure. A CTU is a square block of size 16×16, 32×32, or 64×64. The CTU is recursively subdivided into multiple coding units (CUs) and prediction units (PUs). Each PU serves as an independent, basic entity for compression carrying individual header data. Secondly, a prediction for a PU can be generated using one out of many standard-defined prediction modes.

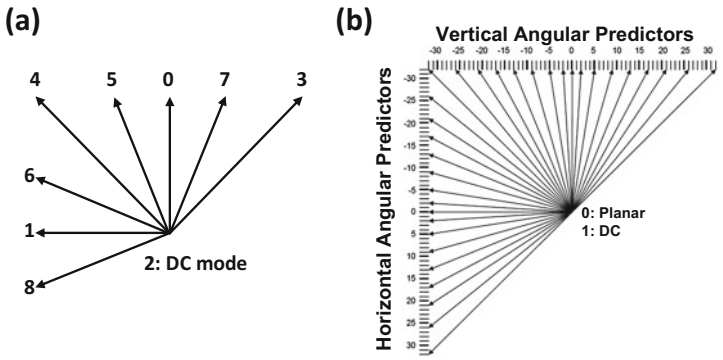


Fig. 2.6 (**a**) H.264/AVC and (**b**) HEVC intra-angular modes

Table 2.1 Comparing HEVC intra-prediction modes for 64×64 CTU with the H.264/AVC intra-modes for a 64×64 image region

	Total intra-angular modes n_{ang}	
Prediction size	HEVC/ H.265 (64×64)	H.264/AVC (16×16)
64×64	4	NA
32×32	35	NA
16×16	35	4
8×8	35	9
4×4	19	9
Total modes	$\psi = 7808$	16×(16×9+4×9+4) = 2944

2.2.1.1 Intra-prediction Modes

As discussed, H.264/AVC and HEVC intra-compression modes generate the prediction block for the current block using the spatial neighboring reconstructed video samples. This is because spatially, the texture flow of the video samples is expected to continue from the surrounding blocks into the current block under consideration. However, the proper direction of texture flow must be estimated for best representation of the original block, which will result in a residual block with low pixel energy. As shown in Fig. 2.6 (a, b), both H.264/AVC and HEVC employ different spatial directional (or angular) modes to determine the best texture flow direction. The encoders employ a brute-force (also called full-search) algorithm to get the best prediction mode, whereby all the predictors are tested to get the best predictor. The best prediction mode is usually chosen as the one which corresponds to the lowest SAD (Eq. 2.4). SAD value is computed between the current block and the prediction block.

Table 2.1 tabulates a comparison of intra-prediction modes for both HEVC and H.264/AVC. It is notable that the total number of prediction modes for the HEVC is significantly higher compared to H.264/AVC. Furthermore, the selection of

RD-wise best PU size and prediction mode is determined by a RD optimization (RDO) process. Therefore, due to the recursive nature of best PU size selection and RDO process, the total number of mode decisions in HEVC (computed using Eq. (2.5) for a CTU of size $b_w \times b_h$=64×64) is ~2.65× more than that in H.264/AVC. n_{ang} denotes the number of angular modes for a particular PU size (column 2 of Table 2.1).

$$\psi = \sum_{i=0}^{\log_2 b_w - 2} \left(2^{2i} \times n_{ang}\right) \tag{2.5}$$

2.2.1.2 HEVC Inter-prediction Modes

Similarly, for generating the inter-predictions, both H.264/AVC and HEVC encoders search for a similar block in temporal neighborhood (i.e., the previously encoded frames called reference frames) using ME. In HEVC, ME is repeated for every PU as shown in Fig. 2.7. This search is pixel-wise and uses the SAD metric. The ME process is shown in Fig. 2.8 along with the hardware architecture of computing the SAD. In the inter-encoding case, a predictor is a block of pixels of the previous frame. Usually, only parts of reference frames, called the search

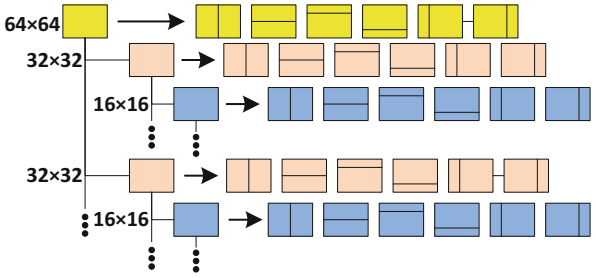


Fig. 2.7 HEVC inter-prediction modes

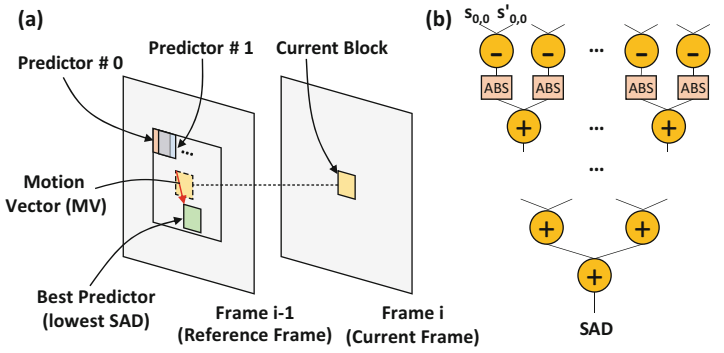


Fig. 2.8 Motion estimation (ME) and SAD computation architecture

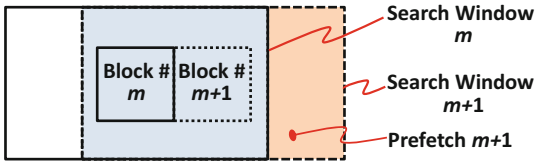


Fig. 2.9 Search window structure and window prefetching

window, are tested for the best predictor. The search window is brought from the external memory to the on-chip memory, and the predictors are drawn from this search window for computing SAD. Search window is a rectangular region of size $s_w \times s_h$. For fast prediction selection, ideally the search window should be loaded into the on-chip memory. Furthermore, note that blocks are processed in a raster scan order, and two neighboring blocks will have most of their search windows overlapped [5]. This concept is outlined in Fig. 2.9. Thus, search window "prefetching" of new reference samples can be done in parallel to the ME process. Further, a larger search window increases the probability of finding a predictor with low SAD value (and thus increases the video quality). However, this also increases the on-chip memory required to store the search window, which increases the power consumption of the system. Additionally, the external memory accesses also increase, resulting in higher power consumption [8, 9] and access latencies.

From the discussion above related to the search window, it is simple to infer that a single reference pixel is written multiple times to the search window, depending upon the height of the window. This is because the search window overlaps for the blocks in adjacent rows. A larger search window height (s_h) denotes that a single pixel will be written more times into the search window storage than having a smaller search window height. A read factor r_f is defined which denotes the total number of times a pixel in the reference frame is read and then stored in the search window. Typical value of this factor is between 3 and 12. For the technique given in Fig. 2.8, where the search window is shifted by b_h for every new row of blocks with size $b_w \times b_h$:

$$r_f = s_h / b_h \tag{2.6}$$

There are multiple ways to determine the best predictor within the search window. One method is to search every predictor in the search window. This scheme employs a brute-force or full-search algorithm and results in the best predictor with the lowest SAD value. However, this scheme is excessively complex and almost never employed in practice. Usually, fast prediction search algorithms (like EPZS, TZ [10]) are used, which result in considerably lower power consumption and complexity with little video quality loss, compared to the brute-force search. In these fast algorithms, not every possible predictor of the search window is tested (i.e., not all pixels of the search window are utilized). An additional step iteratively determines the next most probable predictors which will result in the lowest SAD value. However, this does not suggest that the number of pixels "prefetched" will reduce. The number of pixels fetched from external memory

will remain the same, because the pixels that will be utilized by the iterative step for ME are unknown. Moreover, if a part of the search window is not fully utilized for the current block, it might be utilized for the next adjacent block.

In HEVC, each CU is split into 13 PUs (see Fig. 2.7), and ME is carried out for these PUs. Mathematically, the total number of block-matching iterations for a square CTU of width b_w is given by:

$$\psi = 13 \times \sum_{i=0}^{\log_2 b_w - 3} 2^{2i} \tag{2.7}$$

Simulations show that block matching in HEVC takes around 80% of the total encoding time, and it is ~2.2× more complex than compared to H.264/AVC. In addition, normally the block matching takes about 60% of the total energy in video encoders [11]. Moreover, there can be multiple reference frames (e.g., bi-prediction in HEVC and multiview video coding (MVC)), and the best predictor is searched in a search window for each of these frames.

Many algorithms used in both H.264/AVC and HEVC can be applied in other video processing algorithms, as they exercise the same principles of computations and memory access. Many mainstream video encoders use similar principles of intra- and inter-video encoding (e.g., Google's VP8 and VP9, Microsoft's SMPTE, Cisco's Thor, MJPEG). Moreover, ME algorithm is also used in super-resolution techniques (increasing the resolution of a video frame by concatenating multiple, temporally neighboring frames), temporal frame interpolations (for reducing flicker and algorithmically increasing FPS), motion tracking, corner detection, etc.

Further, almost all video processing algorithms have configuration knobs, which can be tweaked to leverage the computational complexity against video quality. For example, the search window for motion estimation in video coding, denoising, frame interpolation, and super-resolution algorithms can be enlarged or contracted. The number of intra-angular modes tested for HEVC can be increased or reduced. This suggests video processing applications provide prospects for runtime adaptation of workload. However, such adaptation for increasing the throughput might result in lower output quality, e.g., loss in PSNR or precision of tracking.

2.2.2 *Parallelization*

Like all compute-intensive video applications, H.264/AVC and HEVC standards allow for parallel operation. A system designer can utilize a multi- or many-core system and exploit the parallelism in order to gain computational advantages. To process a video sequence, a hierarchical approach for video partitioning is employed. Video frames are divided into sets of frames, called groups of pictures (GOPs). Each GOP can be processed independently of other GOPs, supporting GOP-level parallelization [12]. Within a GOP, video frames can also be processed in parallel on independent processing cores [13]. A frame is divided into slices or

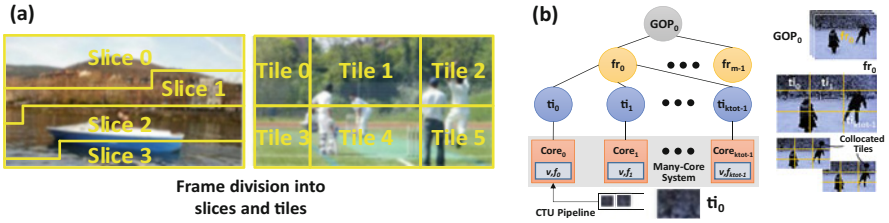


Fig. 2.10 (**a**) Frame division into slices and tiles for parallel processing, (**b**) video partitioning hierarchy. Here, *fr* frame, *ti* tile

tiles (see Fig. 2.10a), and each slice/tile can be processed in parallel [14–16]. As discussed before, usually the image and video processing algorithms are block-based algorithms, where a set of pixels in a rectangular region is considered as a basic processing entity. Each slice/tile is divided into blocks, and a single block is processed one at one time (the blocks within the slice/tile are processed sequentially). However, intra-angular predictions and inter-predictions can be tested in parallel. An example video partition hierarchy is given in Fig. 2.10b, where each frame in the GOP is divided into k_{tot} tiles. This figure also shows collocated tiles, which are the video tiles at the same location but in adjacent frames.

Video Workload Balancing Via parallelization, the workload of the whole application is divided into chunks, and thus the workload corresponds to the computational complexity and energy consumption of a processing core. That is, the larger the workload, the more computational complexity and energy consumption of the underlying hardware (processing resources like cores, FPGAs, GPUs, etc.) are expected. Generally, the workload of the complete application should be distributed to the physical resources in a manner that the hardware utilization is maximized. Therefore, every resource should ideally consume the same amount of time processing their assigned jobs or subtasks. Collectively, this process can be termed as workload or load balancing. Workload balancing will increase the throughput of the system along with increase of the throughput-per-watt metric.

Workload balancing can be achieved via centralized or distributed techniques [17, 18], which gather statistics and objectively distribute the subtasks among the resources. However, for fully distributed strategies, optimal scheduling decision is difficult to make due to rapidly changing environment, randomness, and unpredictability. The communication delays in fully distributed strategies can also invalidate a correct decision at a previous time, as the state of the system might change rapidly after the distributed load balancing decision. Further, distributed algorithms reduce the range of subtask migration from one core to another, because physically far apart resources, with highest and lightest workloads, cannot be balanced in a single iteration of load balancing. Also, the overhead of the technique increases with the system size due to increased message passing, control logic, scheduling algorithms, etc. Further, the response of distributed strategies saturates after tens of compute nodes. For the centralized strategy of load balancing, the overhead of the assignment algorithm is large and has a large communication

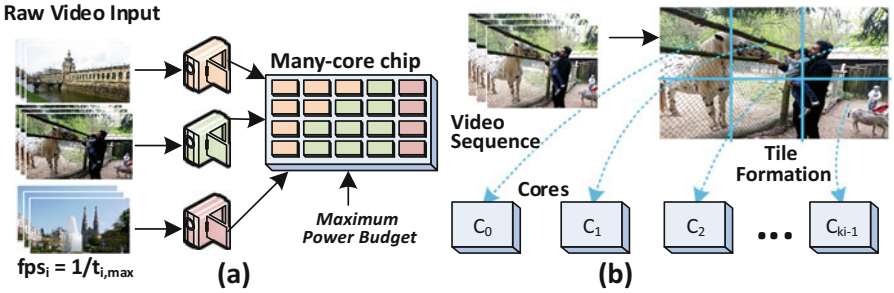


Fig. 2.11 (**a**) Multi-video capture and encoding on a many-core chip, (**b**) frame division into multiple tiles

and storage overhead. This is due to the chip-wide information gathering and storage of statistics. Further, centralized load balancing algorithms are vulnerable to faults, and a complete system failure will occur if the central node (running the subtask assignment algorithm) fails. Thus, both centralized and distributed strategies for load balancing do not always produce optimal performance. This discussion not only is applicable for video processing systems but also applies to any many-core system running parallel workloads.

Multicasting Multicasting refers to information transfer to a set of predefined destination nodes. With reference to video communication, a multicasting or multichannel video encoder should be capable of encoding concurrent [19, 20] video streams in parallel and generating appropriate compressed bitstreams for each of these videos, as shown in Fig. 2.11. Use cases of multicast video encoding include security, entertainment, video logging, etc. Each video can have its own resolution, texture/motion content-dependent workload, and throughput requirement like FPS. This system can be implemented using a many-core platform or by designing custom hardware. In case of a many-core system, it is a design challenge to efficiently distribute the processing cores and power among multiple, concurrently executing encoders, while fulfilling their throughput requirements. Moreover, the load balancing techniques must be applicable to such paradigms. For a custom hardware solution, some of the challenges for the multicast encoder are being area efficient (due to multiple encoders working in parallel) and being able to efficiently access video data.

2.2.3 DVC and HDVC

As discussed, the high compression efficiency of H.264/AVC and HEVC has enabled a wide range of applications under low-bandwidth constraints. They follow the predictive video coding (PVC) model, where the predictions of the block under consideration for compression are generated at the encoder side (using intra-angular

modes or ME). PVC is typically well suited for circumstances where encoding devices have large-enough computational power, while decoders are resource-/power-constrained devices, like mobile devices engaged in video streaming, battery driven hand-held phones etc. Similarly, if the video is encoded once and transmitted to numerous decoders multiple times, then PVC approach is most suitable.

2.2.3.1 Distributed Video Coding

It is possible that the computational complexity of a video application is too high to be feasible for a particular system. For example, the significantly increased computations at the HEVC encoder prohibits its use in constrained encoded scenarios. Distributed video coding (DVC) has been emerged as an attractive solution for scenarios where the encoding devices are resource constrained and must use low-complexity encoding (like infield wireless video sensor nodes, small autonomous flying robots, mobile devices with low processing capabilities, etc.), while the decoding devices have high computational power (like high-end servers) and can execute high-complexity decoding tasks. DVC paradigm provides means to shift/offload the computational workload from the video encoder to decoder [21–27].

A DVC encoder typically consumes only 7% of the total power consumed by a PVC H.264 encoder [28–30]. In DVC paradigm, the decoder performs the ME for frame interpolation, extrapolation, and upsampling. Only a subset of a GOP is transmitted by the encoder, and the decoder generates the complete GOP by exploiting the inter-frame correlation. At the DVC decoder side, Slepian-Wolf decoder and ME with interpolation contribute 90% towards the total decoding complexity. To improve the estimation quality, the DVC decoder requests auxiliary information from the encoder (i.e., parity bits generated by turbo coding), which results in a higher transmission power compared to the PVC case (see Fig. 2.12). A DVC encoder may require ~4× higher transmission energy compared to an H.264 PVC encoder for a video sensor node [28]. Increase in parity bits positively influences the video quality at the decoder side, but results in higher transmission energy at the encoder side.

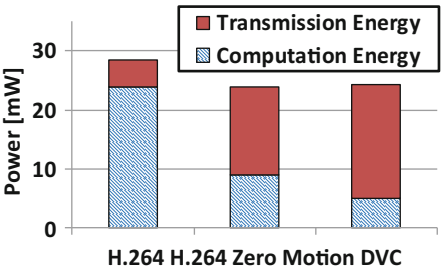


Fig. 2.12 Comparing the computational and transmission power for an ASIC-based video sensor [82]

2.2.3.2 Hybrid Distributed Video Coding

DVC completely offloads the ME from the encoder to the decoder. The major drawback of DVC is lower video quality compared to that provided by PVC. On the other hand, PVC results in best video quality at the expense of high computational complexity at the encoder side. PVC and DVC become power-/energy-wise inefficient in scenarios where both encoder- and decoder-side devices are resource constrained and/or subjected to runtime varying conditions of available energy levels and computational resources. In these cases, only one or neither of the encoder-/decoder-side devices has adequate computational and/or transmission power to deliver the required throughput and/or video quality. Examples of such scenarios are (1) collaborative distributed video sensor networks for smart energy-aware surveillance; (2) mobile devices on Internet of Things (IoT) – with varying battery levels – communicating with each other or other power-constrained devices; (3) heterogeneous devices from different vendors with distinct energy consumption properties, etc. Besides, DVC may not facilitate complete offloading in scenarios where multiple encoding devices concurrently offload their computational workload to a single, shared decoding device [31].

To cope with the energy-related issues for video coding in such dynamic scenarios, Hybrid Distributed Video Coding (HDVC) has emerged as an attractive solution which combines the positive aspects of both PVC and DVC, i.e., providing high video quality close to PVC and low computational power close to DVC. In HDVC, the decoder-side ME complexity is relaxed at the cost of partial ME at the encoder side. This means that the encoder also fully processes some of the video frame blocks and leave the rest to the decoder. The partial ME at the encoder side results in better reconstruction of frames at the decoder side that corresponds to a high video quality and low energy consumption at the decoder side.

Concisely, general workload offloading improves the performance if [32]:

- The throughput requirement of the (video) application is high and is not sustainable by the video processing device.
- The (DVC/HDVC) decoder/receiver is fast and results in computational/power benefit if the jobs are offloaded to the decoder.
- A small amount of auxiliary bits is transmitted.
- The bandwidth of the channel between the encoder and decoder is not a bottleneck.

2.3 Technological Challenges

In the previous chapter, we briefly discussed some of the technological challenges that might arise while implementing a system with high throughput-per-watt metric. Here, details about these challenges are given.

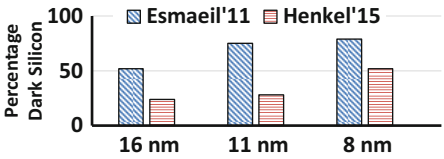


Fig. 2.13 Prediction/trends of dark silicon. See [8] for Esmaeil'11 and [164] for Henkel'15

2.3.1 Dark Silicon or Power Wall

Reduced transistor sizes in the modern fabrication technologies have led to new unforeseen challenges for system designers. Though the chip designers are living up to the Moore's law challenge, it is expected that most of the transistors etched on the chip will not be completely utilized due to the power wall problem. Specifically, the failure of Dennard scaling has resulted in the emergence of the Dark Silicon Age, where the chip's real estate cannot be utilized continuously, at full capacity. In fact, current estimates (as shown in Fig. 2.13) suggest that only 30–50% of the chip's usable resources will be bright (fully utilized) for 8 nm technology, while the rest will be kept dark (unutilized) or dim/gray (partially utilized or underutilized). This forced underutilization arises from the fact that power per unit of area is increasing monotonously with increasing transistor density. Therefore, the temperature of the chip may reach levels which will not be contained by the available state-of-the-art coolants and result in permanent damage of the chip. Thus, power-efficient designs are of prime importance for modern systems.

A digital circuit consumes two types of powers, dynamic (p_{dyn}) and static (p_{sta}) power. The dynamic power is due to switching the transistors *on* and *off*, whereas the leakage power is a result of subthreshold current through the transistor's channel and the leakage through the transistor's gate, when the transistor is *off*. The static power can be reduced by lowering the supply voltage v_{dd}. On the other hand, for a CMOS circuit, the dynamic power consumption can be written as:

$$p_{dyn} = \alpha \cdot c_p \cdot v_{dd}^2 \cdot f \tag{2.8}$$

Here, α is the switching activity level, c_p is the capacitance of the circuit, v_{dd} is the supply voltage, and f is the clock frequency at which the circuit is operated. This shows that dynamic power can be reduced by reducing the supply voltage. However, reducing v_{dd} will increase the time delay (t_d), and, therefore, the frequency must be reduced. In fact, t_d and v_{dd} are related by the following equation:

$$t_d \propto \frac{v_{dd}}{v_{dd} - v_{th}} \tag{2.9}$$

Here, v_{th} is the threshold voltage. Therefore, we can rewrite Eq. (2.8) as:

$$p_{dyn} \propto v_{dd}^3 \propto f^3 \tag{2.10}$$

Using this relationship, dynamic voltage frequency scaling (DVFS) can reduce the power of the circuit. However, note that reducing the frequency of the circuit also reduces the throughput and may result in deadline misses. Moreover, as discussed above, the voltage and frequency of a processor scale together. However, this relationship does not hold for the complete frequency range, since there is a certain threshold below which the voltage reduction results in unstable processor behavior [33]. Thus, only dynamic frequency scaling (DFS) can be employed in these scenarios.

For video applications, a strict throughput constraint usually exists that must be met. Current trends for user-demanded frame resolution and FPS suggest that this throughput demand is increasing (Equation (2.1) and hence puts pressure on the hardware to perform. However, new fabrication technologies – with about half of the cores turned OFF due to dark silicon constraints – require careful consideration of available TDP and its distribution among possibly multiple, multithreaded video applications competing for system resources. Moreover, memory may consume in excess of 40% of the total chip's power [8]. This power includes access to external memory, read/write energy consumption, and leakage/standby power. The memory power consumption is especially of concern for video applications that are memory intensive, and their memory requirements continue to grow with the growing throughput demands. Therefore, it might not be possible to achieve higher power efficiency without considering the memory power.

The discussion above suggests that the software and hardware must be power efficient to exercise the minimum amount of power, while fulfilling the throughput requirements. This will not only address the dark silicon issue but will provide supplementary power to other parallel running applications. Further, the parallelization potential of a video application can be exploited to meet the throughput requirements and ideally distribute the compute power along the complete chip. Also, hardware accelerators, running at a lower frequency/power but generating a higher throughput than its software counterpart, can be strategically placed on the die to reduce the chip's temperature. Additionally, the advantages of the new memory technologies (like MRAM) can be exploited to replace the on-chip SRAM, in order to reduce the leakage power of the memory subsystem and limit the access to the external memory due to their higher density/sizes.

2.3.2 NBTI-Induced SRAM Aging

To provide fast read/write accesses, application-specific architectures normally employ dedicated SRAM-based on-chip memories (like scratchpads instead of caches) for storing data, thus saving the extra power overhead of tags and other supportive circuitries. These on-chip memories are managed using specialized

address generation units and/or explicitly programmed to exploit applications' attributes (more on this later). However, due to continuous technology scaling resulting in small feature sizes, high-power densities (dark silicon paradigm), and resulting temperatures, SRAM-based on-chip memories are exposed to various reliability issues like transient errors (soft errors) and permanent errors (device aging). Memory-intensive video applications have flourished into various mission-critical domains like surveillance and security, automotive, satellite imaging and video transmissions, sensor-based image/video processing over long durations, etc. For these applications, reliable operation over their lifetime or an extended lifetime is an imperative system requirement.

This book considers SRAM aging due to NBTI, which has emerged as one of the most critical reliability threats for the new fabrication technology. NBTI occurs in PMOS transistors due to negative voltage at the gate (i.e., $v_{gs} = -v_{dd}$). This voltage results in interface traps because of the breakdown of the Si-H bond at the Si-SiO_2 interface. This manifests as a surge in threshold voltage and reduction in noise margin (i.e., short-term aging) that may lead to timing errors/delay faults and/or runtime performance degradation. To encounter this threshold voltage increase (more than 50 mV [34]), the clock frequency of the device must be reduced by more than 20% over its lifetime. However, due to rising NBTI issues and cost/power/performance constraints, the degradation of the cell stability can no longer be addressed by simply providing a design time delay margin [35]. This aging-based phenomenon is partially reversed in the so-called recovery mode (Si-H bond is reformed in a few cases) once the stress is removed from the PMOS gate, i.e., at v_{gs}=0. An abstract view of this process is shown in Fig. 2.14a for a PMOS transistor. Such a situation occurs when a "zero" overwrites the "one" stored in the SRAM cell and vice versa. However, 100% recovery is not possible and NBTI results in continuous degradation over years (i.e., long-term aging). The total aging throughout the lifetime depends upon the stress and recovery cycles; see Fig. 2.14b. For ease of discussion, this book defines the duty cycle (Δ) as the percentage of a cell's lifetime when the stored value is "one."

This book considers a memory composed of numerous 6T SRAM cells. Each cell is composed of two inverters to store a bit value (see Fig. 2.14c), and these inverters store complementary values at all times. The word line (WL) is enabled to write a value, while the bit line (BL) is used to deliver data to be stored in the cell.

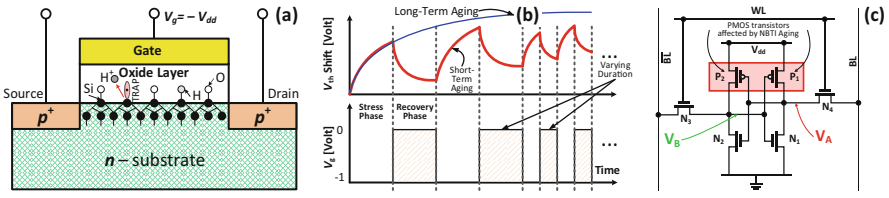


Fig. 2.14 (**a**) PMOS transistor demonstrating NBTI aging with breaking Si-H bond at the Si-SiO_2 interface, (**b**) example of stress and recovery phases for a PMOS transistor, and (**c**) a standard 6T SRAM cell

The data is retained in the cell by turning off access transistors. To read data, WL is set high and the BL value is retrieved. Both these transistors are in complementary states at all times. In case a "zero" or "one" value is stored in an SRAM cell, one of its PMOS transistors will be under stress and the other in the recovery phase. Since the aging of an SRAM cell is determined by the worst-case aging of one of the two PMOS transistors, the overall lowest aging is achieved when both PMOS transistors are stressed by the same amount of time during the whole lifetime. That is, an SRAM cell contains "one" value for 50% of its lifetime. This corresponds to a duty cycle (Δ) of 50%. In short, SRAM aging depends upon the duty cycle of the transistors in an SRAM cell. If the duty cycle is balanced (which denotes Δ=50%), the aging rate of SRAM cell will minimize.

Note that NBTI is not the only deteriorating mechanism active in SRAM cells. Hot carrier injection (HCI, which mainly degrades the NMOS transistor) is another aging mechanism which injects high-energy (hot) carriers inside the gate oxide. This causes interface traps and thus introduces threshold voltage shift [36], which is presented by the following equation:

$$\Delta v_{th} = \psi_{HCI} s_a f e^{\frac{v_{dd}-v_{th}}{d_{ox}\psi_1}} t^{0.5} \tag{2.11}$$

In this equation, ψ_{HCI} and ψ_1 are aging rate-dependent constants, s_a is the switching activity, and d_{ox} is the oxide thickness. This equation shows that a higher switching activity will increase the HCI-induced aging rate and vice versa. On the contrary, higher switching activity will reduce NBTI. Studies like [37, 38], however, emphasize that NBTI has a greater impact and is the dominating factor in limiting the life of a circuit.

2.3.3 Other Challenges

Some other challenges, which arise due to recent fabrication technology advancements, are enabling high instruction-level parallelism (ILP) and having a high memory bandwidth. Sometimes, they are referred to as ILP wall and memory wall. ILP wall makes it difficult to parallelize the instruction streams in order to keep the resources busy, and, thus, it prohibits the increase in throughput. Further, the reduced bandwidth between the caches/external memory and the computing resources (e.g., CPUs) is a performance-limiting factor contributed by the so-called memory wall. However, this book will only focus on power wall and SRAM aging.

2.4 Related Work

2.4.1 Video System Software

This section presents state-of-the-art techniques for addressing different challenges of a video system at the software layer. These issues include parallelization, complexity reduction, and power/resource budgeting.

2.4.1.1 Parallelization and Workload Balancing

As previously discussed, parallelization is a fundamental requisite of high-complexity video applications, which must be exploited on many-core systems, possibly having hardware accelerators and custom logic/interfaces. The general classification of parallelization and workload mapping practices on many-core systems is presented in the literature [39]. However, one of the objectives of this book is to present and analyze the application-specific properties for workload mapping on a many-core system and improving power efficiency of the video system. Numerous works are reported to enable parallel processing of video applications. These works include parallel video coding/decoding [40, 41], tracking [42], image/face recognition [43, 44], nonnegative matrix factorization [45], and others. However, these works generally do not consider resource management, hardware characteristics, and workload balancing.

Workload Balancing General load balancing of compute jobs among compute entities is presented in [46, 17, 18, 47]. For power efficiency (i.e., increased throughput-per-watt ratio), either the load of an application can be dispersed on a given platform (load balancing [48, 49]) or a platform can be synthesized for the given load/application (load-driven synthesis [50, 51]) under throughput and/or power constraints. Most of the load balancing techniques consider homogeneous many-core systems, jobs with almost equal complexity and do not consider load variation at runtime [46, 52]. For example, [47] considers load balancing for distributed stream processing applications in wide-area environment, under dynamic resource consumption. In [53], load balancing between mirror multimedia servers is discussed for both centralized and distribution load balancing techniques. Ref. [46] deals with assigning each resource (core) with equal number of subtasks and reaches an equilibrium state if no more jobs can be migrated from one core to its physical, homogeneous neighbors. However, the present architectural and physical challenges for homogeneous many-core system design are not considered. Smaller feature sizes result in physical variability of underlying transistors (also called process variation), which transforms into variable leakage power and maximum frequency achievable for the homogeneous cores on the same die [54]. Thus, compute cores can have different characteristics, even though they form a homogeneous many-core system. Research has also focused on combining the

distribution and balancing of load and DVFS and dynamic power management (DPM) of the underlying hardware resources [55]. Ref. [56] defines a single clock frequency for the entire chip for maximum efficiency, whereas [16, 57] independently determine the frequency of each core while distributing application load. Authors in [16] target minimizing the power consumption for a fixed deadline, while [57] tries to maximize the throughput of parallel running, multithreaded applications for a given power budget.

Workload Balancing on Heterogeneous Nodes Further, to increase the throughput-per-watt under modern system design challenges, heterogeneous multi-/many-core systems are becoming progressively popular [33, 58]. Using architectural heterogeneity, it is now possible for the designer to schedule a processing job on a compute node (e.g., a core, a hardware accelerator) that will increase the throughput-per-watt metric [59]. This way, maximum power efficiency is achieved. For example, ARM big.LITTLE architecture [60] integrates high-performance Cortex-A57 big cores with low-power Cortex-A53 little cores, in order to achieve maximum throughput-per-watt by capitalizing on adaptive application mapping techniques. Thus, general load balancing methods are not applicable in heterogeneous paradigms having cores/compute nodes with unequal compute capabilities. Parallelization and load balancing of H.264/AVC is carried out in [61], using heterogeneous CPU+GPU systems [62, 63], without considering the impact of power which is substantial when GPUs are used. In [33], authors target energy-efficient workload allocation and voltage-frequency tuning of the underlying single-ISA computing nodes. The goal is to minimize energy/power of the system. However, their approach does not consider fulfilling the required throughput of the application(s). In [64], authors propose to identify the program and cores' characteristics and then appropriately match them for scheduling. Reference [65] studies parallelized database on heterogeneous, single-ISA architectures. These proposed workload balancing approaches do not consider modern fabrication technological challenges like power budgeting, dark silicon, etc. Moreover, generally the complexity of each subtask is not equal. For example, ME can have considerable different complexity for different video frame blocks, depending upon the content properties and texture within the block [66, 11]. Further, the cache behavior of the application and the physical locality of the compute node (e.g., its distance from the external memory controller) also determine the complexity of a subtask. All these challenges must be addressed if an efficient workload mapping and balancing policy needs to be implemented. This is especially true for video systems under throughput constraints.

Parallelization of Video Systems Multiple parallelization methods for video systems are available in the literature. For example, a many-core based SIMD implementation of H.264/AVC is given in [67], but it does not consider workload mapping and balancing. Using partial frame-level parallelism, a 12-core system for parallel HEVC encoding is given in [40]. Ref. [41] discusses a technique to parallelize H.264/AVC on Cell multiprocessor. In [68], a hierarchical parallelization of H.264/AVC is presented for low-cost cluster of cores, by

combining multilevel parallelism. In [69], a parallel implementation of particle filter on shared memory architectures is given. In [70], a hardware/software partitioning is targeted for a heterogeneous processor, for MPEG-2 encoder. Ref. [71] proposes a parallel implementation of the nonlocal means filter (NLMF) on a GPU for denoising 3D data. Parallel video super-resolution methods are proposed in [72, 73]. In general, not all these parallelization techniques consider workload balancing on the many-core system, power reduction, and meeting the throughput requirements. In addition, these techniques might require access to multiple video frames (in the external memory) at the same time, increasing the latency of the application. Thus, they either increase the power consumption of the system by needlessly increasing the core frequency beyond requirement or reduce the throughput and increase latency by burdening each compute resource with divergent workloads.

H.264/AVC parallelization and workload balancing are also discussed in the literature. Authors in [14] present a history-based technique to allocate the number of slices per frame dynamically, for balancing workload among the multiple cores. In their technique, each slice is mapped to a single compute core. A similar history-based technique can be found in [15] where the skipped video frame blocks determine the slice boundaries for parallel encoding. A two-pass slice partitioning technique for workload balancing of H.264/AVC is given in [74], where each frame is preprocessed, prior to being assigned into slices. However, no adaptation of workload and frequency of the cores (and thus, of power consumption) is proposed for these techniques.

2.4.1.2 Power-Efficient Video Processing Algorithms

A multitude of works in reducing the complexity of computationally heavy image/video applications also exist in the literature. In a nutshell, by sacrificing a small amount of output quality (e.g., a reduced PSNR or accuracy of tracking, increased bitrate), the workload of the application is curtailed to meet the throughput.

Numerous complexity reduction techniques exist in the literature for HEVC encoding [75–78]. The work in [75] (basically inspired from [79, 80]) presents a gradient-based fast intra-mode decision for a given PU size and results in about 20% time savings. In [76], authors have also presented a fast PU size selection algorithm for inter frames (exploiting temporal correlations for frame compression, similar to open loop for H.264/AVC presented in [81]). A divide-and-conquer strategy for choosing the best intra-angular prediction is given in [77]. First, eight equally spaced modes (at a distance of 4 in both directions; see Fig. 2.6b) are tested. Afterwards, six best modes with a distance of 2 are tested. In the end, two best modes are left which are tested with a distance of 1 to select the best mode. However, the number of modes selected for RD-cost determination is static and fairly large. Similarly in [78], to reduce the total number of intra-prediction modes tested for selecting the best predictor, an open-loop technique is utilized. Using the current pixels instead of reconstructed pixels, the total number of predictors is

reduced from 35 to 9. These nine modes are used for computing the rate-distortion (RD) cost. In [82], the video frame block (CTU) is downsampled and then texture-complexity (via variance) is computed, to determine the appropriate PU sizes for best encoding. Reference [83] exploits the correlation of intra-prediction modes of the current block with the final predictors of the neighboring blocks for determining a highly probable intra-prediction mode for the current block. An edge-based intra-prediction candidate selection technique is given in [84] to reduce the total number of modes tested by 73% and a time reduction from ~8% to ~32%. The 4×4 pixels in each PU are treated for determining the principal direction, and a set of nine intra-predictors is used for testing. However, the selection and truncation of intra-prediction modes is not adaptive. Similarly, numerous works exist to reduce the complexity and energy consumption of the inter-prediction engine (i.e., ME engine). Techniques to reduce the total number of operations in ME are also widely studied and employed [85, 10, 11]. For example, in [86], authors have proposed a technique to reduce the off-chip memory accesses for ME, which results in 56% memory access reduction. However, their technique is tested for a very small search window size of 16, which is not recommended to be used for large resolution sequences.

Other power-efficient techniques for H.264/AVC (e.g., [87, 88]) may not be directly or efficiently applicable to new video encoders like HEVC, due to the novel CTU structure of HEVC and nature of its angular prediction modes. Moreover, these techniques usually do not jointly consider power efficiency and workload balancing and do not exploit the speedup achieved via parallel encoding on a many-core platform. Further, these techniques do not consider the underlying platform properties (i.e., do not exploit the opportunities provided by the hardware) and the new challenges introduced by the reduced feature sizes, while managing their workload.

In addition to video coding, there is a plethora of other video processing algorithms, where complexity knobs are tuned at the cost of output quality. For example, see [89, 90]. Libraries like "open-source computer vision" (OpenCV [91]), and standards like OpenVX [92], provide plentiful implementations of these video algorithms.

2.4.1.3 Mitigating Dark Silicon at Software Level

The purpose of these techniques is to limit the maximum temperature or maximum power (TDP) consumed by the system at the software level. The abovementioned software-level techniques (i.e., parallelization, complexity reduction, budgeting, and offloading) implicitly address the dark silicon challenges. For realizing these techniques, mainly two distinct approaches are employed:

- Dynamic thermal management (DTM), which involves adjusting the voltage-frequency or power of the cores (DVFS [93]) and even severing the power to the

compute resources (via power gating, also called dynamic power management (DPM))
- Tasks/thread/workload migration, which involves migrating the subtask from one compute resource to another, in case the former's temperature becomes critical [94]

The frame-based energy management technique for real-time systems [95] exploits workload variations and the interplay between DVFS and DPM, for a real-time embedded application. It determines the optimal voltage-frequency setting and power levels of the devices to minimize the system's energy. In [96], a trial-and-error-based centralized algorithm determines the appropriate DVFS settings for the many-core system, to maximize application speedup under chip's power constraint. A thread mapping methodology under the constraints of thermal safe power (TSP) is presented in [97]. The authors argue that TDP is very conservative, and power more than TDP can be pumped into the chip to speed up execution if intelligent task-mapping decisions are made. The work in [93] proposes PID controller-based mechanisms at runtime for efficient utilization of the TDP budget, in order to maximize performance of architecturally heterogeneous cores synthesized with different power and performance targets. However, [93] does not target power budgeting among multithreaded applications with thread-level workload variations. In [98], a two-level closed-loop power control technique is given. Using voltage/frequency islands on a chip, the power distribution is divided among the compute resources. However, fine-grained power distribution and configuration selection using this technique are not possible, which are important for multithreaded applications with varying workloads. Furthermore, the closed-loop control usually responds slowly, causing performance issues for applications with abrupt workload changes. Single thread-based power budgeting is discussed in [99], which is not applicable to modern multithreaded applications. PEPON [100] also presents a two-level power budgeting technique to maximize performance within an allocated power cap. The technique in [57] targets control-based chip-level and application-level power budgeting, while accounting for the critical threads of the application.

Most of these techniques do not consider assigning compute resources to the applications, and the varying workload of the applications at runtime, which might require readjustment of the resource allocation. Moreover, these techniques do not target power allocation to dependent applications or subtasks of a single application, where the critical application or subtask will reduce the throughput of the complete system. These works also ignore the tuning characteristics of the applications and the opportunity they might provide for increased throughput-per-watt. Further, applications with throughput constraints must meet their deadlines (e.g., multimedia applications having soft deadlines and mission-critical applications with hard deadlines), which is usually not addressed by these works; rather, speedup is the optimization target. This poses additional challenges if the system load (e.g., due to parallel running applications, delay in delivering Ethernet packets) may change at runtime.

In summary, most of these techniques do not exploit the opportunities provided by the following:

- Selecting appropriate operating modes (configuration of different variables) of the applications and dark silicon
- Determining the right compute resources at nominal frequency/voltage settings even for unadaptable applications

Collectively, the operating modes of the application and the dark silicon provide opportunities of having multiple power modes [101], for instance:

- Powering on more cores at lower frequency to facilitate more applications or applications with high thread-level parallelism
- Powering on less cores at higher frequency to facilitate high instruction-/data-level parallelism
- Choosing appropriate operating modes of the applications to enable higher throughput at the same power budget

Mostly, DTM only gathers system statistics and manages the system stack excluding the application layer. Since these techniques do not take the application-specific characteristics into consideration, therefore, they lack power efficiency in cases of abrupt workload variations and/or when multiple threads of different applications (especially with mixed workload characteristics) are competing for the power budget. Hence, fine-grained power distribution and configuration selection is not possible.

This book will only focus on DTM-based techniques for power management of video applications, and runtime workload migration techniques will not be considered.

2.4.2 Video Systems Hardware

This section introduces the state-of-the-art approaches that target design and implementation of video system hardware architecture to address multiple challenges.

2.4.2.1 Efficient Hardware Design and Architectures

Numerous state-of-the-art techniques exist for designing compute- and power-efficient video systems. For encoding HD videos, new methods and tools to administer the necessary data processing tasks and dependencies of video encoders are required. For example, [102] discuss an H.264/AVC intra-encoder chip operating at 54 MHz clock. However, it is only capable of handling a small throughput requirement (720 $\times$ 480 4:2:0 video at 30 fps). Additionally, the authors use a parallel structure for generating the predictors that increases silicon area footprint. In [103], a fast prediction selection preprocessor, based on spatial domain filtering,

is proposed. A four-stage pipeline for edge extraction increases the latency of the design, and processing one video block requires 416 cycles at a maximum possible clock rate of 66 MHz. The design in [104] presents a 1920 × 1080 (1080p) intra-encoder, providing 25 fps at 100 MHz. It takes about 440 cycles to compute the intra-predictions. In [105], a 4K UHD (3840 × 2160) resolution at 60 fps intra-prediction architecture is presented, which replicates hardware for high throughput and needs to run at 310 MHz to achieve 4K UHD while still using suboptimal prediction selection methods. In [79], authors propose a fast method of selecting the best prediction, based on the texture flow. Authors in [106] present a low-latency 1080p, 61 fps intra-encoder architecture operating at 150 MHz. However, the proposed design tests the predictions in parallel and takes about 300 cycles to encode one block. Reference [81] presents a so-called open-loop (OL) method to determine the most likely predictor based upon the original image pixels rather than the reconstructed pixels. Similarly, a multitude of novel architectural designs for HEVC are also available. In [107], the authors proposed an HEVC intra-prediction HW for only 4×4 blocks. In addition to video coding, several other multimedia processing systems are realized using efficient architectures, for example, deblocking filters, AES, and CRC [108, 109].

Motion Estimation Many approaches to reduce the energy consumption of the video processing systems target the ME engine for optimization, because ME (also called block matching) is the most time- and energy-consuming process of a video encoder. In [110], ME energy is reduced by dropping the supply voltage and then employing error resiliency features. This results in energy savings of up to 60% on 130 nm CMOS technology. However, this technique results in extra control and noise-tolerance circuitry and worsens the output video quality as well. Reference [111] lessens the search space for ME and thus avoids redundant memory accesses. However, an unnecessary brute-force full-search algorithm results in high computation effort. In addition to the technique's failure to account for sudden motion, their results are for CIF/QCIF videos which already require a small search space for ME. In [112], the on-chip memory is replaced with a cache. Further, they do not explore the opportunities by reducing the leakage energy (which is dominant for submicron technology [113]). The work in [114] reduces the external memory accesses by frame buffer compression, but requires additional computations. In [115, 116], different data reuse schemes for reducing the external memory accesses by video processing algorithms are categorized from levels A to D, with highest latency (smallest on-chip memory) to the lowest latency (largest on-chip memory). There are other extensions of the originally proposed levels, like level C+ [5]. Reducing the total number of predictions (ME operations [85, 10]) also decreases the latency of execution but has little improvement for memory energy consumption and external memory access (see details in Sect. 2.2.1.2).

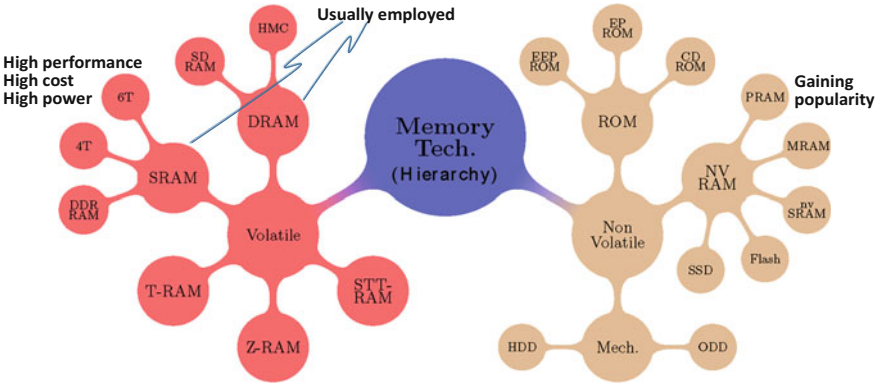


Fig. 2.15 Different memory technologies and their design hierarchy

Table 2.2 Abstract comparison between NVM and VM memory technologies

Technology	Access Speed		Power Consumption			Density	NVM
	Read	Write	Dynamic		Leakage		
			Read	Write			
SRAM	HH	HH	L	L	HH	LL	No
DRAM	H	H	H	H	H	H	No
MRAM	H	L	L	HH	L	H	Yes
PRAM	L	LL	H	HH	L	HH	Yes

H denotes high, *L* denotes low, *blue* color denotes advantage, *red* color denotes disadvantage

2.4.2.2 Memory Subsystem

The leakage energy of the memory is of more importance in the submicron era, as it surpasses the dynamic power of the memory [113]. Therefore, new memory technologies are evolving that address the issues of density and leakage power. A synopsis of some of the memory technologies and their hierarchy is given in Fig. 2.15. Next-generation NVMs like MRAM [117–119] and Phase-change RAMs (PRAM) [120, 121] have shown promising results towards leakage power reduction compared to SRAM or DRAM. Application designers are considering to exploit these memories by analyzing their advantages and disadvantages. Table 2.2 summarizes the main differences between the NVM and VM technologies. NVMs provide high capacity and low leakage power but their write latency and energy is substantially larger compared to that of SRAMs. However, the nonvolatility of the NVMs can be sacrificed, and NVMs like STT-RAM can be used as VMs.

In [122] and [123], a hybrid memory architecture comprising of PRAM and DRAM is proposed. A hybrid of scratchpad and NVMs for on-chip memories is

given in [124]. A technique which utilizes hybrid memory for video decoding is given in [125], where frame-level decisions for storing H.264 frames on hybrid-memories are used. However, H.264/AVC encoder presents a harder challenge as it is ~10× to 20× more complex than the decoder [2]. Reference [9] targets the HEVC application and limits the external memory access by storing the video samples (which are expected to be used later) in a hybrid combination of SRAM and STT-RAM. Other approaches utilizing MRAMs as replacements and augmentation of the traditional fast SRAMs are reported in [126, 127].

Most state-of-the-art memory subsystem designs do not jointly reduce the power consumption of the system in conjunction with meeting the throughput demands. Additionally, overall system characteristics (e.g., data transfer from external to on-chip memory) are not considered.

2.4.2.3 Accelerator Allocation/Scheduling

In order to combine the advantages of both programmable and application-specific custom architectures (e.g., ASICs), accelerator-based many-core systems are becoming increasingly popular in the industry [128, 129]. Accelerators are implemented as custom hardware for high-complexity parts of programs (called subtasks), and a programmable core can offload its assigned tasks to these accelerators. For examples of accelerators, refer to Sect. 2.4.2.1. Accelerators naturally lend themselves to occupy the underutilized chip's area, i.e., occupy dim/gray silicon. In addition to increasing the bright silicon, accelerators are designed to quickly process the assigned tasks. Therefore, accelerators are fundamental to high-complexity, deadline-conscious applications. Examples include video encoding and decoding [130] (also see Intel's Quick Sync technology), software-defined radios [131], etc.

For ease of discussion, we broadly classify accelerators into three categories, based upon their flexibility and access mechanisms. These categories are also shown in Fig. 2.16:

- First are the in-core accelerators, which are embedded as a part of the programmable core's computation pipeline (e.g., Nios II custom instructions [132, 133], vector instructions [134], application-specific instruction-set processors (ASIPs) [135, 136]). However, note that the corresponding core can only access these accelerators, and they are a part of the execution stage in the computational pipeline. Therefore, these accelerators exhibit the least flexibility, as their cores can only access these accelerators.
- The second category is clustered accelerators, where an accelerator can be accessed by only a specific set of cores and they reside in vicinity of these cores (e.g., within a computation tile). Such accelerators are also called tightly coupled accelerators [137, 138]. Techniques like [139, 140] allocate the accelerator to the corresponding cores by offloading the soft-core subtasks, using past-predicts-future paradigms and dynamic programming. However, these

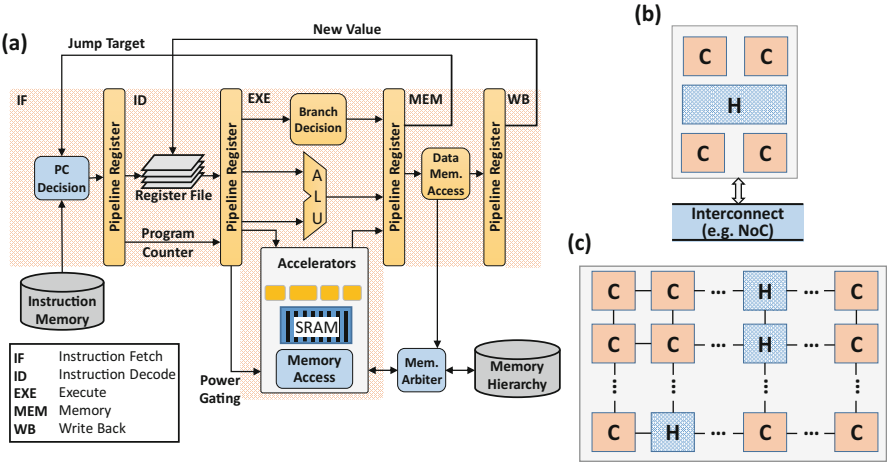


Fig. 2.16 Accelerator locality and access-based classification, (**a**) In-core, (**b**) tightly coupled, (**c**) loosely or decoupled accelerators

techniques do not consider the complete power consumption of the system, and neither do they account for the deadlines of the running applications.

- The third and the most flexible category of accelerators can be accessed by all the cores (e.g., via a network-on-chip (NoC) and PCIe) and are called decoupled accelerators or loosely coupled accelerators. It is obvious that the clustered and decoupled accelerators are the most versatile and offer maximum advantages. However, state-of-the-art accelerator allocation techniques presented in the literature [141–145] for decoupled accelerators usually try to reduce the resources used, maximize the processing speed, or reuse the accelerators' memory as cache or reconfigurable logic. No reference to the power consumption, frequency tuning of the cores, and deadlines of the applications is made.

Since the shared accelerator can only be allotted to a single compute resource at a given time, therefore some of the applications running on these cores might miss their deadlines, or these applications might change their workload at runtime. Further, it is likely that the accelerator is not continuously utilized, which defeats their purpose of providing power and complexity efficiency. In addition, it is also possible that to meet the deadlines, higher than required power is pumped to the cores. This will increase the power consumption of the system and, therefore, elevate the temperature of the chip.

2.4.2.4 SRAM Aging Rate Reduction Methods

In general, state-of-the-art techniques for aging mitigation mainly target aging optimization (i.e., aging rate reduction) for SRAM-based register files. However,

these techniques do not target large memories which have distinct access behavior and require different architectural support. The first category of techniques is based on the principle of bit rotations (i.e., moving LSB by one position with every write to the memory location) to improve duty cycle of registers [146, 147]. These techniques perform ineffectively for registers with successive zeros and are beneficial only when the bits inside registers are frequently modified, which is typically not the case for large-sized memories. Moreover, applying bit rotations requires barrel shifters at the read and write ports of the memory. We know that the total number of multiplexers required to implement an n-bit barrel shifters is $n\log_2 n$, and this is in addition to the control logic which is used to configure the barrel shifters. Therefore, the area overhead of such techniques might be high.

In [35], a redundancy-based SRAM mircoarchitecture extends the life of an SRAM cell. Similar to [148–150], this also requires architectural modification of the 6T SRAM cell. The register value inversion techniques result in additional reads/writes and power. The recovery boosting technique [148] adds dedicated inverters in the SRAM cells to improve the recovery process. However, this incurs significant power overhead, which may be infeasible for large-sized video memories, for instance, targeting image buffers for ME at high-definition (HD, 1920×1280 bytes) and 4K UHD (3840×2160 bytes) resolutions. Additionally, it requires an alteration to the SRAM 6T cell circuitry. Another category of work is based on bit flipping each bit at every write to the memory [147, 151, 148]. In [152], an algorithm is introduced for balancing the duty cycle of SRAM data caches by exploiting cache characteristics (i.e., tag bits). A similar technique is presented in [153, 154]. These architectures and techniques depend upon the inherent properties of caches (like flushing, cache hits, etc.) and are not directly pertinent to general on-chip SRAM memories. Moreover, some of the mentioned balancing policies are designed for capturing and exploiting the occurrence of a certain bit pattern and thus perform inefficiently for other content properties and varying stress patterns. Further, many of the reported works for aging balancing require multiple read/write of the same data in the memory, rendering themselves to be power hungry and increasing the latency of the application by halting their access to the memory.

Summarizing, state-of-the-art aging balancing techniques incur significant power and area overhead by employing bit flipping or rotation at every bit level, and reading and writing to the memories multiple times. These techniques do not explore the tradeoff between power consumption and aging balancing. Moreover, most of these techniques only provide elementary circuitry without exploring benefits of different aging balancing architectures and lack full architectural solution with power-aware aging control and adaptations.

2.4.2.5 Encountering the Power Wall at Hardware Level

Brief details about handling the power wall or dark silicon at the hardware layer of the video system is given in this section. However, the abovementioned state-of-the-art techniques (for designing efficient hardware accelerators, scheduling the

shared accelerator, power-efficient memories, etc.) implicitly address these challenges at the hardware layer.

At the hardware layer, different control knobs (e.g., for DVFS and DPM) are provided to throttle the chip's temperature within safe limits. Other techniques employ architectural heterogeneity to trade off performance and power. Via heterogeneity, the system supports runtime management of tasks by providing several degrees of freedom to the system designer. One can classify the different forms of heterogeneity as [155]:

- *Functional heterogeneity*, where compute nodes exist with varying functional behaviors and architectural details. Examples are application-specific hardware accelerators, superscalar cores and RISC processors, GPUs in conjunction with CPUs, reconfigurable architectures, etc. Thus, using task migrations and using scheduling, design challenges for modern fabrication technologies can be encountered.
- *Accelerator heterogeneity*, same as discussed in Sect. 2.4.2.3, i.e., in-core, clustered, and decoupled accelerators, providing different levels of performance and flexibility of usage. Further, approximate accelerators [156, 157] with controllable amount of approximations can also be used to increase the throughput-per-watt metric. This will not only increase the amount of bright silicon but also enable higher performance.
- *Microarchitectural heterogeneity*, whereby different cores on the same die have varying power and performance properties, but employ the same instruction set architecture (ISA). An example is ARM big.LITTLE architecture [60]. For example, [158] presents a methodology to design multi-core systems while considering the dark silicon paradigm. The purpose of their technique is to maximize the utilization of the silicon. In [50], depending upon the characteristics of parallel running applications, dark silicon-aware multiprocessors are synthesized using a library of available core types. In [159], special-purpose conservation cores (c-cores) are discussed, the goal of which is to reduce the energy consumption of the system rather than boasting performance. Device-level heterogeneous multi-cores and resource management are exploited in [160] to speed up the performance, as well as save energy.
- *On-chip interconnect heterogeneity*, whereby the network routers that connect the multiple cores of the chip are designed with heterogeneous architectures [161]. This provides diverse power and performance design points, available for the system designer.
- *Process heterogeneity*, where the nonideal fabrication process results in core-to-core and chip-to-chip variations in the maximum achievable frequency and leakage power. This variation can be exploited to adaptively grow the speedup of applications [162, 163] while meeting the TDP budgets of the chip.

2.5 Summary of Related Work

A plethora of techniques to tackle challenges imposed by video system software, hardware, and new fabrication technologies are presented in the state-of-the-art. Summarizing, the state-of-the-art does not exploit the complete design space concerning both hardware-software co-design and co-optimization. This is specifically important for multimedia systems, under dark silicon and reliability threats. For best throughput-per-watt ratio, the designer needs to consider the full-system stack, which involves the design of software layer, to the knowledge and exploitation of the hardware layer. This knowledge can be used to fully exploit complexity, power, and resource savings and reliability improvement potential for long-term system deployment.

Usually, the dark silicon mitigation techniques proposed in the state of the art do not consider the throughput constraints, and they do not exploit application-specific properties. As discussed, multimedia systems have deadline constraints, which require intelligent power budget distribution (i.e., frequency allocation) among the resources. Mostly, the state of the art does not consider the impact of deadlines and resource and power budgeting for shared accelerator-based systems. This results in suboptimal performance of the system and reduction in power efficiency.

Similarly, for SRAM aging-rate reduction, state-of-the-art techniques employ fixed aging balancing algorithms and architectures, with significant energy overhead. Therefore, these techniques are unable to explore the tradeoff between aging balancing and energy consumption. Moreover, due to the added power consumption, the state-of-the-art techniques might result in a higher temperature, which will increase the aging rate in a positive feedback cycle. Further, exploring the application-specific properties might result in high power efficiency and high reliability, which is mostly ignored by the state of the art.

References

1. Bjontegaard, G. (2001). Calculation of average PSNR differences between RD-curves. VCEG Contribution VCEG-M33.
2. Ostermann, J., Bormans, J., List, P., Marpe, D., Narroschke, M., Pereira, F., Stockhammer, T., & Wedi, T. (2004). Video coding with H.264/AVC: Tools, performance, and complexity. *IEEE Circuits and Systems Magazine, 4*(1), 7–28.
3. Sullivan, G. J., Ohm, J., Han, W., & Wiegand, T. (2012). Overview of high efficiency video coding. *IEEE Transactions on Circuits and Systems for Video Technology, 22*(12), 1649–1668.
4. Wiegand, T., Sullivan, G., Bjontegaard, G., & Luthra, A. (2003). Overview of the H.264/AVC video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology, 13*(7), 560–576.
5. Chen, C., Huang, C., Chen, Y., & Chen, L. (2006). Level C+ data reuse scheme for motion estimation with corresponding coding orders. *IEEE Transactions on Circuits and Systems for Video Technology, 16*(4), 553–558.

6. Zhu, S., & Ma, K.-K. (2000). A new diamond search algorithm for fast block-matching motion estimation. *IEEE Transactions on Image Processing, 9*(2), 287–290.
7. Bossen, F., Bross, B., Suhring, K., & Flynn, D. (2012). HEVC complexity and implementation analysis. *IEEE Transactions on Circuits and Systems for Video Technology, 22*(12), 1685–1696.
8. Sze, V., Finchelstein, D. F., Sinangil, M. E., & Chandraksan, A. P. (2009). A 0.7-V 1.8-mW H.264/AVC 720p video decoder. *IEEE Journal of Solid-Sate Circuits, 44*(11), 2943–2956.
9. Sampaio, F., Shafique, M., Zatt, B., Bampi, S., & Henkel, J. (2014). Energy-efficient architecture for advanced video memory. In *International Conference on Computer-Aided Design*.
10. Purnachand, N., Alves, L. N., & Navarro, A. (2012). Improvements to TZ search motion estimation algorithm for multiview video cod-ing. In *IEEE International Confernce on Systems, Signals and Image Processing (IWSSIP)*, pp. 388–391.
11. Shafique, M., Bauer, L., & Henkel, J. (2010). 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 and Test in Europe*.
12. Gurhanli, A., Chen, C.-P., & Hung, S.-H. (2010). GOP-level parallelization of the H.264 decoder without a start-code scanner. In *International Conference on Signal Processing Systems (ICSPS)*.
13. VideoLAN - x264. [Online]. Available: http://www.videolan.org/developers/x264.html. Accessed 5 Oct 2015.
14. Zhao, L., Xu, J., Zhou, Y., & Ai, M. (2012). A dynamic slice control scheme for slice-parallel video encoding. In *International Conference on Image Processing*.
15. Ba, K., Jin, X., & Goto, S. (2010). A dynamic slice-resize algorithm for fast H.264/AVC parallel encoder. In *International Symposium on Intelligent Signal Processing and Communication Systems*.
16. Khan, M. U. K., Shafique, M., & Henkel, J. (2014). Software architecture of high efficiency video coding for many-core systems with power-efficient workload balancing. In *Design, Automation and Test in Europe*.
17. Ahmad, I., & Ghafoor, A. (1991). Semi-distributed load balancing for massively parallel multicomputer systems. *IEEE Transactions on Software Engineering, 17*(10), 987–1004.
18. Williams, R. (1991). Performance of dynamic load balancing algorithms for unstructured mesh calculations. *Concurrency: Practice and experience, 3*(5), 457–481.
19. Juice Encoder– 4 in 1 MPEG-4 AVC/H.264 HD encoder. Antik Technology, [Online]. Available: http://www.antiktech.com/iptv-products/juice-encoder-EN-5004-5008/
20. Marvell 88DE3100 High-Definition Secure Media Processor System-on-Chip (SoC). [Online]. Available: http://www.marvell.com/digital-entertainment/armada-1500/assets/Marvell-ARMADA-1500-Product-Brief.pdf/
21. Distributed Coding for Video Services (DISCOVER). Application scenarios and functionalities for DVC.
22. Wyner, A., & Ziv, J. (1976). The rate-distortion function for source coding with side information at the decoder. *IEEE Transaction on Information Theory, 22*, 1–10.
23. Girod, B., Aaron, A. M., Rane, S., & Rebollo-Monedero, D. (2005). Distributed Video Coding. *Proceedings of the IEEE, 93*(1), 71–83.
24. Puri, R., & Ramchandran, K. (2003). PRISM: An uplink-friendly multimedia coding paradigm. In *International Conference on Acoustics, Speech, and Signal Processing*.
25. Chen, J., Khisti, A., Malioutov, D., & Yedidia, J. (2004). Distributed source coding using serially-concatenated-accumulate codes. In *Information Theory Workshop*.
26. Tseng, H.-Y., Shen, Y.-C., & Wu, J.-L. (2011). Distributed video coding with compressive measurements. In *International conference on Multimedia*.
27. Sejdinovic, D., Piechocki, R. J., & Doufexi, A. (2009). Rateless distributed source code design. In *Mobile Multimedia Communica-tions Conference*.

28. Chien, S.-Y., Cheng, T.-Y., Chiu, C.-C., Tsung, P.-K., Lee, C.-H., Somayazulu, V., & Chen, Y.-K. (2012). Power optimization of wireless video sensor nodes in M2M networks. In *Asia and South Pacific Design Automation Conference*.
29. Huang, Y.-W., Chen, T.-C., Tsai, C.-H., Chen, C.-Y., Chen, T.-W., Chen, C.-S., Shen, C.-F., Ma, S.-Y., Wang, T.-C., Hsieh, B.-Y., Fang, H.-C., & Chen, L.-G. (2005). A 1.3TOPS H.264/AVC single-chip en-coder for HDTV applications. In *International Solid-State Circuits Conference*.
30. Chiu, C.-C., Chien, S.-Y., Lee, C.-H., Somayazulu, V., & Chen, Y.-K.. (2011). Distributed video coding: A promising solution for distributed wireless video sensors or not?. In *Visual Communications and Image Processing*.
31. Shafique, M., Khan, M. U. K., & Henkel, J. (2013). Content-driven adaptive computation offloading for energy-aware hybrid distributed video coding. In *International Symposium on Low Power Electronics and Design (ISLPED)*.
32. Kumar, K., Liu, J., Lu, Y.-H., & Bhargava, B. (2012). A survey of computation offloading for mobile systems. *Mobile Networks and Applications, 18*(1), 129–140.
33. Colin, A., Kandhalu, A., & Rajkumar, R. (2015). Energy-efficient allocation of real-time applications onto single-ISA heterogeneous multi-core processors. *Journal of Signal Processing Systems,* pp. 1–20.
34. Schroder, D. K., & Babcock, J. A. (2003). Negative bias temperature instability: Road to cross in deep submicron silicon semiconductor manufacturing. *Journal of Applied Physics, 94*(1), 1–18.
35. Shin, J., Zyuban, V., Bose, P., & Pinkston, T. (2008). A proactive wearout recovery approach for exploiting microarchi-tectural redundancy to extend cache SRAM lifetime. In *International Symposium on Computer Architecture (ISCA)*.
36. Tiwari, A., & Torrellas, J. (2008). Facelift: Hiding and slowing down aging in multicores. In *International Symposium on Microarchitecture (MICRO)*.
37. Vattikonda, R., Wang, W., & Cao, Y. (2006). Modeling and minimization of PMOS NBTI effect for robust nanometer design. In *Design Automation Conference (DAC)*.
38. Velamala, J. B., Sutaria, K., Sato, T., & Cao, Y. (2012). Physics matters: Statistical aging prediction under trapping/detrapping. In *Design Automation Conference (DAC)*.
39. Singh, A., Shafique, M., Kumar, A., & Henkel, J. (2013). Mapping on multi/many-core systems: Survey of current and emerging trends. In *Design Automation Conference (DAC)*.
40. Chi, C. C., Alvarez-Mesa, M., Juurlink, B., Clare, G., Henry, F., Pateux, S., & Schierl, T. (2012). Parallel scalability and efficiency of HEVC parallelization approaches. *IEEE Transactions on Circuits and Systems on Video Technology, 22*(12), 1827–1838.
41. Alvanos, M., Tzenakis, G., Nikolopoulos, D. S., & Bilas, A. (2011). Task-based parallel H. 264 video encoding for explicit communication architectures. In *International Conference on Embedded Computer Systems*.
42. Brun, O., Teuliere, V., & Garcia, J. M. (2002). Parallel particle filtering. *Journal of Parallel and Distributed Computing, 62*(7), 1186–1202.
43. Rujirakul, K., So-In, C., & Arnonkijpanich, B. (2014). PEM-PCA: A parallel expectation-maximization PCA face recognition architecture. *The Scientific World Journal*.
44. Jing, X.-Y., Li, S., Zhang, D., Yang, J., & Yang, J.-Y. (2012). Supervised and unsupervised parallel subspace learning for large-scale image recognition. *IEEE Transactions on Circuits and Systems for Video Technology, 22*(10), 1497–1511.
45. Dong, C., Zhao, H., & Wang, W. (2010). Parallel nonnegative matrix factorization algorithm on the distributed memory platform. *International Journal of Parallel Programming, 38*(2), 117–137.
46. Shah, S., & Kothari, R. (2013). Convergence of the dynamic load balancing problem to Nash equilibrium using distributed local interactions. *Information Sciences, 221*, 297–305.
47. Drougas, Y., Repantis, T., & Kalogeraki, V. (2006). Load balancing techniques for distributed stream processing applications in overlay environments. In *IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing*.

48. Robertazzi, T. G. (2003). Ten reasons to use divisible load theory. *Computer, 36*(5), 63–68.
49. Wang, K., Zhou, X., Li, T., Zhao, D., Lang, M., & Raicu, I. (2014). Optimizing load balancing and data-locality with data-aware scheduling. In *International Conference on Big Data*.
50. Turakhia, Y., Raghunathan, B., Garg, S., & Marculescu, D. (2013). HaDeS: Architectural synthesis for heterogeneous dark silicon chip multi-processors. In *Design Automation Conference (DAC)*.
51. Buss, M., Givargis, T., & Dutt, N. (2003). Exploring efficient operating points for voltage scaled embedded processor cores. In *Real-Time Systems Symposium (RTSS)*.
52. Rosas, C. Morajko, A. Jorba, J., & Cesar, E. (2011). Workload balancing methodology for data-intensive applications with divisible load. In *Symposium on Computer Architecture and High Performance Computing*.
53. Matthur, A., & Mundur, P. (2003). Dynamic load balancing across mirrored multimedia servers. In *International Conference on Multimedia and Expo*.
54. Bowman, K., Duvall, S., & Meindl, J. (2002). Impact of die-to-die and within-die parameter fluctuations on the maximum clock frequency distribution for gigascale integration. *IEEE Journal of Solid-State Circuits, 37*(2), 183–190.
55. Kim, J., Yoo, S., & Kyung, C.-M. (2011). Program phase-aware dynamic voltage scaling under variable computational workload and memory stall environment. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 30*(1), 110–123.
56. Devadas, V., & Aydin, H. (2010). DFR-EDF: A unified energy management framework for real-time systems. In *Real-Time and Embedded Technology and Applications Symposium (RTAS)*.
57. Ma, K., Li, X., Chen, M., & Wang, X. (2011). Scalable power control for many-core architectures running multi-threaded applications. In *Internation Symposium on Computer Architecture*.
58. Dehyadegari, M., Marongiu, A., Kakoee, M., Mohammadi, S., Yazdani, N., & Benini, L. (2015). Architecture support for tightly-coupled multi-core clusters with shared-memory HW accelerators. *IEEE Transactions on Computer, 64*(8), 2132–2144.
59. Sarma, S., Muck, T., Bathen, L., Dutt, N., & Nicolau, A. (2015). SmartBalance: A sensing-driven linux load balancer for energy efficiency of heterogeneous MPSoCs. In *Design Automation Conference (DAC)*.
60. ARM big.LITTLE Architecture. ARM, [Online]. Available: http://www.arm.com/products/processors/technologies/biglittleprocessing.php. Accessed 07 Aug 2015.
61. Momcilovic, S., Ilic, A., Roma, N., & Sousa, L. (2014). Dynamic load balancing for real-time video encoding on heterogeneous CPU+GPU systems. *IEEE Transactions on Multimedia, 16*(1), 108–121.
62. Xiao, W., Li, B., Xu, J., Shi, G., & Wu, F. (2015). HEVC encoding optimization using multi-core CPUs and GPUs. *IEEE Transactions on Circuits and Systems for Video Technology (TCSVT), 9*(99), 1–14.
63. Momcilovic, S., Roma, N., & Sousa, L. (2013). Exploiting task and data parallelism for advanced video coding on hybrid CPU + GPU platforms. *Journal of Real-Time Image Processing,* pp. 1–17.
64. Jian, C., & John, L. (2009). Efficient program scheduling for heterogeneous multi-core processors. In *Design Automation Conference (DAC)*.
65. Mühlbauer, T., Rödiger, W., Seilbeck, R., Kemper, A., & Neumann, T. (2014). Heterogeneity-conscious parallel query execution: Getting a better mileage while driving faster!. In *International Workshop on Data Management on New Hardware*.
66. Shafique, M., Molkenthin, B., & Henkel, J. (2010). An HVS-based adaptive computational complexity reduction scheme for H.264/AVC video encoder using prognostic early mode exclusion. In *Design, Automation and Test in Europe Conference (DATE)*.
67. Bariani, M., Lambruschini, P., & Raggio, M. (2012). An efficient multi-core SIMD implementation for H.264/AVC encoder. In *VLSI Design*.

68. Rodríguez, A., González, A., & Malumbres, M. P. (2006). Hierarchical parallelization of an H.264/AVC video encoder. In *International Symposium on Parallel Computing in Electrical Engineering*.
69. Gong, P., Basciftci, Y., & Ozguner, F. (2012). A parallel resampling algorithm for particle filtering on shared-memory architectures. In *Parallel and Distributed Processing Symposium Workshops*.
70. Henkel, J., & Yanbing, L. (1998). Energy-conscious HW/SW-partitioning of embedded systems: a case study on an MPEG-2 encoder. In *International Workshop on Hardware/ Software Codesign*.
71. Cuomo, S., Michele, P. D., & Piccialli, F. (2014). 3D data denoising via nonlocal means filter by using parallel GPU strategies. In *Computational and Mathematical Methods in Medicine*.
72. Moustafa, M., Ebied, H. M., Helmy, A., Nazamy, T. M., & Tolba, M. F. (2014). Satellite super resolution image reconstruction based on parallel support vector regression. In *Advanced Machine Learning Technologies and Applications*, Springer, pp. 223–235.
73. Garcia Freitas, P., Farias, M. and De Araujo, A. (2014). A parallel framework for video super-resolution. In *Graphics, Patterns and Images (SIBGRAPI)*.
74. Jung, B., & Jeon, B. (2008). Adaptive slice-level parallelism for H.264/AVC encoding using pre macroblock mode selection. *Journal of Visual Communication Image Representation, 19* (8), 558–572.
75. Jiang, W., Mal, H., & Chen, Y. (2012). Gradient based fast mode decision algorithm for intra prediction in HEVC. In *International Conference on Consumer Electronics, Communications and Networks*.
76. Cassa, M. B., Naccari, M., & Pereira, F. (2012). Fast rate distortion optimization for the emerging HEVC standard. In *Picture Coding Symposium.*
77. Zhang, H., & Ma, Z. (2012). Fast intra prediction for high efficiency video coding. In *Advances in Multimedia Information Processing*.
78. Sun, H., Zhou, D., & Goto, S. (2012). A low-complexity HEVC Intra prediction algorithm based on level and mode filtering,. In *International Conference on Multimedia and Expo (ICME)*.
79. Pan, F., Lin, X., Rahardja, S., Lim, K. P., Li, Z. G., Wu, D., & Wu, S. (2005). Fast mode decision algorithm for intraprediction in H.264/AVC video coding. *IEEE Transactions on Circuits and Systems for Video Technology, 15*(7), 813–822.
80. Tsai, A. C., Paul, A., Wang, J. C., & Wang, J. F. (2008). Intensity gradient technique for efficient intra-prediction in H.264/AVC. *IEEE Transactions on Circuits and Systems on Video Technology, 18*(5), 694–698.
81. Fonseca, T. A., Liu, Y., & Queiroz, R. L. D. (2007). Open-loop prediction in H.264 / AVC for high definition sequences. In *SBrT*.
82. Tian, G., & Goto, S. (2012). Content adaptive prediction unit size decision algorithm for HEVC intra coding. In *Picture Coding Symposium*.
83. Zhao, L., Zhang, L., Ma, S., & Zhao, D. (2011). Fast mode decision algorithm for Intra prediction in HEVC. In *Visual Communications and Image Processing (VCIP)*.
84. Silva, T. D., Agostini, L. V., & Cruz, L. A. D. S. C. (2012). Fast HEVC intra prediction mode decision based on EDGE direction information. In *European Signal Processing Conference (Eusipco)*.
85. Haan, G. D., & Biezen, P. (1998). An efficient true-motion estimator using candidate vectors from a parametric motion model. *IEEE Transactions on Circuits and Systems for Video Technology, 8*(9), 86–91.
86. Shim, H., & Kyung, C.-M. (2009). Selective search area reuse algorithm for low external memory access motion estimation. *IEEE Transactions on Circuits and Systems for Video Technology, 19*(7), 1044–1050.
87. Kun, Z., Chun, Y., Qiang, L., & Yuzhuo, Z. (2007). A fast block type decision method for H.264/AVC intra prediction. In *International Conference on Advanced Communication Technology*.

88. Lin, Y.-K., & Chang, T. (2005). Fast block type decision algorithm for intra prediction in H.264/FRex. In *Internatianal conference on Image Processing (ICIP)*.
89. Kivanc Mihcak, M., Kozintsev, I., Ramchandran, K., & Moulin, P. (1999). Low-complexity image denoising based on statistical modeling of wavelet coefficients. *IEEE Signal Processing Letters, 6*(12), 300–303.
90. Khan, M. U. K., Bais, A., Khawaja, M., Hassan, G. M., & Arshad, R. (2009). A swift and memory efficient hough transform for systems with limited fast memory. In *International Conference on Image Analysis and Recognition (ICIAR)*.
91. OpenCV. [Online]. Available: http://opencv.org/. Accessed 08 Aug 2015.
92. OpenVX. [Online]. Available: https://www.khronos.org/openvx/. Accessed 08 Aug 2015.
93. Muthukaruppan, T. S., Pricopi, M., Venkataramani, V., Mitra, T., & Vishin, S. (2013). Hierarchical power management for asymmetric multi-core in dark silicon era. In *Design Automation Conference (DAC)*.
94. Khdr, H., Ebi, T., Shafique, M., Amrouch, H., & Henkel, J. (2014). mDTM: Multi-objective dynamic thermal management for on-chip systems. In *Design, Automation and Test in Europe Conference and Exhibition (DATE)*.
95. Devadas, V., & Aydin, H. (2012). On the interplay of voltage/frequency scaling and device power management for frame-based real-time embedded applications. *IEEE Transactions on Computers, 61*(1), 31–44.
96. Isci, C., Buyuktosunoglu, A., Cher, C., Bose, P., & Martonosi, M. (2006). An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget. In *Microarchitecture*.
97. Pagani, S., Khdr, H., Munawar, W., Chen, J.-J., Shafique, M., Li, M., & Henkel, J. (2014). TSP: Thermal safe power: Efficient power budgeting for many-core systems in dark silicon. In *International Conference on Hardware/Software Codesign and System Synthesis*.
98. Mishra, A., Srikantaiah, S., Kandemir, M., & Das, C. R. (2010). CPM in CMPs: Coordinated power management in chip-multiprocessors. In *International Conference on High Performance Computing, Networking, Storage and Analysis*.
99. Winter, J. A., Albonesi, D. H., & Shoemaker, C. A. (2010). Scalable thread scheduling and global power management for heterogeneous many-core architectures. In *Parallel Architectures and Compilation*.
100. Sharifi, A., Mishra, A., Srikantaiah, S., Kandemir, M., & Das, C. R. (2012). PEPON: Performance-aware hierarchical power budgeting for NoC based multicores. In *Parallel Architectures and Compilation Techniques*.
101. Shafique, M., Garg, S., Henkel, J., & Marculescu, D. (2014). The EDA challenges in the dark silicon era. In *Design Automation Conference*.
102. Huang, Y.-W., Hsieh, B.-Y., Chen, T.-C., & Chen, L.-G. (2005). Analysis, fast algorithm, and VLSI architec-ture design for H.264/AVC intra frame coder. *IEEE Transactions on Circuits and Systems for Video Technology, 15*(3), 378–401.
103. Wang, J.-C., Wang, J.-F., Yang, J.-F., & Chen, J.-T. (2007). A fast mode decision algorithm and its VLSI design for H.264/AVC intra-prediction. *IEEE Transactions on Circuits and Systems for Video Technology, 17*(10), 1414–1422.
104. Roszkowski, M., & Pastuszak, G. (2010). Intra prediction hardware module for high-profile H.264/AVC encoder. In *Signal Processing Algorithms, Architectures, Arrangements and Applications Conference*.
105. He, G., Zhou, D., Zhou, J., & Goto, S. (2010). Intra prediction architecture for H.264/AVC QFHD encoder. In *Picture Coding Symposium*.
106. Diniz, C., Zatt, B., Thiele, C., Susin, A., Bampi, S., Sampaio, F., Palomino, D., & Agostini, L. (2011). A high throughput H.264/AVC intra-frame encod-ing loop architecture for HD1080p. In *International Symposium on Circuits and Systems*.
107. Li, F., Shi, G., & Wu, F. (2011). An efficient VLSI architecture for 4×4 intra prediction in the High Efficiency Video Coding (HEVC) standard. In *International Conference on Image Processing*.

108. Cervero, T., Otero, A., López, S., de la Torre, E., Callicó, G., Riesgo, T., & Sarmiento, R. (2013). A scalable H.264/AVC deblocking filter architecture. *Journal of Real-Time Image Processing,* pp. 1–25.
109. Mangard, S., Aigner, M., & Dominikus, S. (2003). A highly regular and scalable AES hardware architecture. *IEEE Transactions on Computers, 52*(4), 483–491.
110. Varatkar, G. V., & Shanbhag, N. R. (2008). Error-resilient motion estimation architecture. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 16*(10), 1399–1412.
111. Saponara, S., & Fanucci, L. (2004). Data-adaptive motion estimation algorithm and VLSI architecture design for low-power video systems. *IEE Proceedings-Computers and Digital Techniques, 151*(1), 51–59.
112. Tsai, C.-Y., Chung, C., Chen, Y.-H., Chen, T.-C., & Chen, L.-G. (2007). Low power cache algorithm and architecture design for fast motion estimation in H. 264/AVC encoder system. In *In-ternational Conference on Acoustics, Speech and Signal Processing*.
113. Kim, N. S., Austin, T., Blaauw, D., Mudge, T., Flautner, K., Hu, J. S., Irwin, M. J., Kandemir, M., & Narayanan, V. (2003). Leakage current: Moore's law meets static pow-e. *Computers, 36*(12), 68–75.
114. Ma, Z., & Segall, A. (2011). Frame buffer compression for low-power video coding. In *International Conference on Image Processing*.
115. Hsu, M.-Y. (2000). Scalable module-based architecture for MPEG-4 BMA motion estimation.
116. Tuan, J.-C., Chang, T.-S., & Jen, C.-W. (2002). On the data reuse and memory bandwidth analysis for full-search block-matching VLSI architecture. *IEEE Transactions on Circuits and Systems for Video Technology, 12*(1), 61–72.
117. Wu, X., Li, J., Zhang, L., Speight, E., Rajamony, R., & Xie, Y. (2009). Hybrid cache architecture with disparate memory technologies. In *International Symposium on Computer Architecture (ISCA)*.
118. Diao, Z., Li, Z., Wang, S., Ding, Y., Panchula, A., Chen, E., Wang, L.-C., & Huai, Y. (2007). Spin-transfer torque switching in magnetic tunnel junctions and spin-transfer torque random access memory. *Journal of Physics: Condensed Matter, 19*(16), 1–13.
119. Dong, X., Wu, X., Sun, G., Xie, Y., Li, H., & Chen, Y. (2008). Circuit and microarchitecture evaluation of 3D stacking magnetic RAM (MRAM) as a universal memory replacement. In *Design Automation Conference (DAC)*.
120. Qureshi, M. K., Srinivasan, V., & Rivers, J. A. (2009). Scalable high perfor-mance main memory system using phase-change memory technology. In *International Symposium on Computer Architec-ture (ISCA)*.
121. Hanzawa, S., Kitai, N., Osada, K., Kotabe, A., Matsui, Y., Matsuzaki, N., Takaura, N., Moniwa, M., & Kawahara, T. (2007). A 512KB Embed-ded phase change memory with 416kB/s write throughput at 100uA cell write current. In *International Solid-State Circuits Conference (ISSCC)*.
122. Yang, S., & Ryu, Y. (2012). A memory management scheme for hybrid memory architecture in mission critical computers. In *International Conference on Software Technology*.
123. Dhiman, G., Ayoub, R., & Rosing, T. (2009). PDRAM: A hybrid PRAM and DRAM main memory system. In *Design Automation Conference*.
124. Bathen, L., & Dutt, N. (2012). HaVOC: A hybrid memory-aware virtualization layer for on-chip distributed scratchpad and non-volatile memories. In *Design Automation Conference*.
125. Stancu, L. C., Bathen, L. A. D., Dutt, N., & Nicolau, A. (2012). AVid : Annotation driven video decoding for hybrid memories. In *Embedded Systems for Real-Time Multimedia*.
126. Desikan, R., Lefurgy, C., Keckler, S., & Burger, D. (2002). *On-chip MRAM as a high-bandwidth, low-latency replacement for DRAM physical memories*. University of Texas at Austin.

127. Nomura, K., Abe, K., Yoda, H., & Fujita, S. (2012). Ultra low power processor using perpendicular-STT-MRAM/SRAM based hy-brid cache toward next generation normally-off computers. *Journal of Applied Physics, 111*(7).
128. Seiler, L., Carmean, D., Sprangle, E., Forsyth, T., Abrash, M., Dubey, P., Junkins, S., Lake, A., Sugerman, J., Cavin, R., Espasa, R., Grochowski, E., Juan, T., & Hanrahan, P. (2008). Larrabee: A Many-Core x86 Architecture for Visual Computing. *ACM Transactions on Graphics, 27*(3).
129. Mattina, M. (2014). *Architecture and Performance of the Tile-GX Processor Family,* White Paper.
130. Shafique, M., Bauer, L., & Henkel, J. (2010). Optimizing the H.264/AVC video encoder application structure for reconfigurable and application-specific platforms. *Journal of Signal Processing Systems (JSPS), 60*(2), 183–210.
131. Liu, C., Granados, O., Duarte, R., & Andrian, J. (2012). Energy efficient architecture using hardware acceleration for software defined radio components. *Journal of Information Processing Systems, 8*(1), 133–144.
132. Nios II Custom Instruction User Guide. Altera, (2011).
133. Khan, M. U. K., Shafique, M., & Henkel, J. (2013). Hardware-software collaborative complexity reduction scheme for the emerging HEVC intra encoder. In *Design, Automation and Test in Europe (DATE)*.
134. Shojania, H., & Baochun, L. (2007). Parallelized progressive network coding with hardware acceleration. In *International Workshop on Quality of Service*.
135. Doan, H. C., Javaid, H., & Parameswaran, S. (2014). Flexible and scalable implementation of H.264/AVC encoder for multiple resolutions using ASIPs. In *Design, Automation and Test in Europe Conference and Exhibition (DATE)*.
136. Kim, S. D., Lee, J. H., Hyun, C. J., & Sunwoo, M. H. (2006). ASIP approach for implementation of H.264/AVC. In *Asia and South Pacific Conference on Design Automation (ASP-DAC)*.
137. Swanson, S., & Taylor, M. B. (2011). GreenDroid: Exploring the next evolution in smartphone application processors. *IEEE Communications Magazine, 49*(*4*), 112–119.
138. Teich, J., Henkel, J., Herkersdorf, A., Schmitt-Landsiedel, D., Schröder-Preikschat, W., & Snelting, G. (2010). Invasive computing: An overview. In *Multiprocessor System-on-Chip*, Springer, pp. 241–268.
139. Sheldon, D., & Forin, A. (2010). An online scheduler for hardware accelerators on general purpose operating systems. Microsoft Research.
140. Huang, C., Sheldon, D., & Vahid, F. (2008). Dynamic tuning of configurable architectures: The AWW online algorithm. In *International Conference on Hardware/Software Codesign and System Synthesis*.
141. Majumder, T., Pande, P. P., & Kalyanaraman, A. (2013). High-throughput, energy-efficient network-on-chip-based hardware accelerators. *Journal of Sustainable Computing: Informatics and Systems, 3*(1), 36–46.
142. Cong, J., Ghodrat, M. A., Gill, M., Grigorian, B., & Reinman, G. (2012). Architecture support for accelerator-rich CMPs. In *Design Automation Conference*.
143. Cota, E., Mantovani, P., Petracca, M., Casu, M., & Carloni, L. (2012). Accelerator memory reuse in the dark silicon era. *Computer Architecture Letters,* pp. 1–4.
144. Clemente, J. A., Beretta, I. V., Rana, V., Atienza, D., & Sciuto, D. (2014). A mapping-scheduling algorithm for hardware acceleration on reconfigurable platform. *Transactions on Reconfigurable Technology and Systems, 7*(2).
145. Paul, S., Karam, R., Bhunia, S., & Puri, R. (2014). Energy-efficient hardware acceleration through computing in the memory. In *Design, Automation and Test in Europe (DATE)*.
146. Kothawade, S., Chakraborty, K., & Roy, S. (2011). Analysis and mitigation of NBTI aging in register file: An end-to-end approach. In *International Symposium on Quality Electronic Design (ISQED)*.

147. Amrouch, H., Ebi, T., & Henkel, J. (2013). Stress balancing to mitigate NBTI Effects in register files. In *Dependable Systems and Networks (DSN)*.
148. Siddiqua, T., & Gurumurthi, S. (2010). Recovery boosting: A technique to enhance NBTI recovery in SRAM arrays. In *Annual Symposium on VLSI*.
149. Sil, A., Ghosh, S., Gogineni, N., & Bayoumi, M. (2008). A novel high write speed, low power, read-SNM-Free 6T SRAM cell. In *Midwest Symposium on Circuits and Systems*.
150. Abella, J., Vera, X., Unsal, O., & Gonzalez, A. (2008). NBTI-resilient memory cells with NAND gates. US Patent US20080084732 A1.
151. Wang, S., Jin, T., Zheng, C., & Duan, G. (2012). Low power aging-aware register file design by duty cycle balancing. In *Design, Automation and Test in Europe (DATE)*.
152. Wang, S., Duan, G., Zheng, C., & Jin, T. (2013). Combating NBTI-induced aging in data caches. In *Great lakes symposium on VLSI*.
153. Gunadi, E., Sinkar, A. A., Kim, N. S., & Lipasti, M. H. (2010). Combating aging with the colt duty cycle equalizer. In *International Symposium on Microarchitecture*.
154. Calimera, A., Loghi, M., Macii, E., & Poncino, M. (2011). Partitioned cache architectures for reduced NBTI-induced aging. In *Design, Automation and Test in Europe (DATE)*.
155. Henkel, J., Bukhari, H., Garg, S., Khan, M. U. K., Khdr, H., Kriebel, F., Ogras, U., Parameswaran, S., & Shafique, M. (2015). Dark silicon – From computation to communication. In *International Symposium on Networks-on-Chip (NOCs)*.
156. Esmaeilzadeh, H., Sampson, A., Ceze, L., & Burger, D. (2012). Neural acceleration for general-purpose approximate programs. In *International Symposium on Microarchitecture*.
157. Mahajan, D., Yazdanbakhsh, A., Park, J., Thwaites, B., & Esmaeilzadeh, H. (2015). Prediction-based quality control for approximate accelerators. In *Workshop on Approximate Computing Across the System Stack*.
158. Allred, J., Roy, S., & Chakraborty, K. (2012). Designing for dark silicon: A methodological perspective on energy efficient systems. In *International Symposium on Low Power Electronics and Design (ISLPED)*.
159. Venkatesh, G., Sampson, J., Goulding, N., Garcia, S., Bryksin, V., Lugo-Martinez, J., Swanson, S., & Taylor, M. B. (2010). Conservation cores: Reducing the energy of mature computations. In *Architectural Support for Programming Languages and Operating Systems*.
160. Swaminathan, K., Kultursay, E., Saripalli, V., Narayanan, V., Kandemir, M., & Datta, S. (2013). Steep-slope devices: From dark to dim silicon. *IEEE Micro, 33*(5), 50–59.
161. Bokhari, H., Javaid, H., Shafique, M., Henkel, J., & Parameswaran, S. (2014). darkNoC: Designing energy-efficient network-on-chip with multi-Vt cells for dark silicon. In *Design Automation Conference (DAC)*.
162. Raghunathan, B., Turakhia, Y., Garg, S., & Marculescu, D. (2013). Cherry-picking: Exploiting process variations in dark-silicon homogeneous chip multi-processors. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*.
163. Shafique, M., Gnad, D., Garg, S., & Henkel, J. (2015). Variability-aware dark silicon management in on-chip many-core systems. In *Design, Automation and Test in Europe Conference and Exhibition*.
164. Huang, W., Rajamani, K., Stan, M., & Skadron, K. (2011). Scaling with design constraints: Predicting the future of big chips. *IEEE Micro, 31*(4), 16–29.

Chapter 3
Power-Efficient Video System Design

This chapter provides an overview of designing a video system to meet the challenges outlined in Chap. 2. Details are given about the architectural aspects and the complexity and power control knobs of the system. By examining these knobs, motivational analysis is carried out which forms the foundation of the algorithmic- and architectural-design decisions presented in this book.

In this book, we would refer to a hardware compute entity performing some calculations as a compute node. Thus, compute nodes present programmable cores, hardware accelerators, GPUs, etc., whereas a compute resource refers to any hardware entity, like memory, interconnect, resource, etc.

3.1 System Overview

The diagram of the video processing system that will be discussed in this book is shown in Fig. 3.1. This diagram categorizes the key components of the system into software and hardware layers. The software layer executes the algorithms and the hardware layer provides support to run these algorithms. Here, we consider that different multithreaded video applications are concurrently executed on the system, and the system generates the output and runtime statistics.

The hardware layer contains a multi-/many-core system, with in-core accelerators and coupled shared hardware accelerators. A gated clock feeds the shared accelerators. The hardware layer provides the video I/O and communication infrastructure among the cores and the accelerators. The hardware layer also contains custom-designed NVM (here MRAM is used) and SRAM. NVM's power is

The authors would like to point out that this work was carried out when all the authors were in Department of Computer Science, Karlsruhe Institute of Technology, Karlsruhe, Germany.

M.U.K. Khan et al., *Energy Efficient Embedded Video Processing Systems*,
DOI 10.1007/978-3-319-61455-7_3

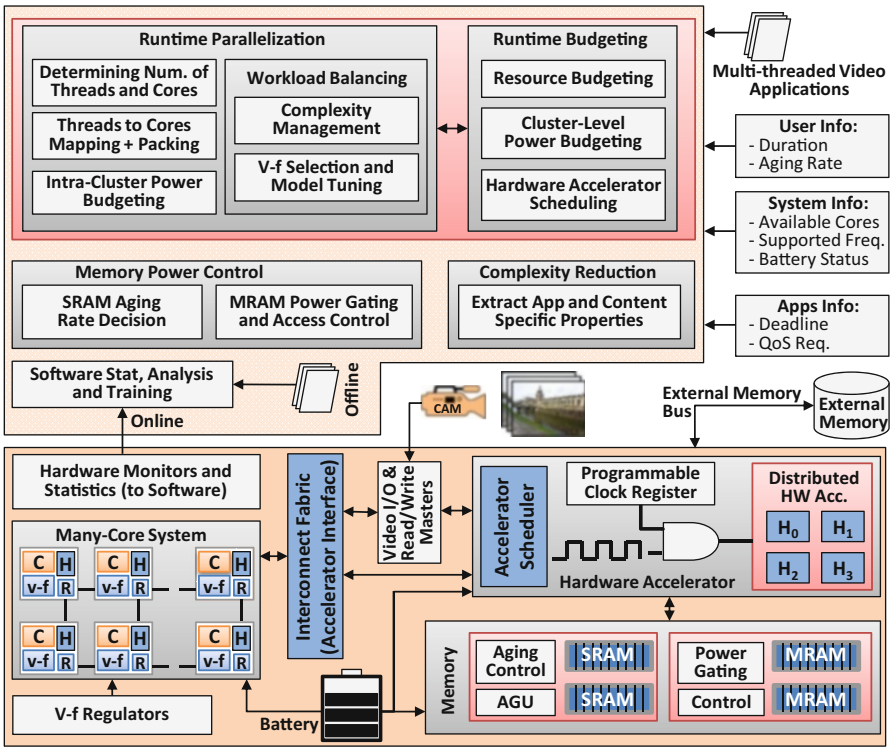


Fig. 3.1 Software and hardware layers of the video processing system

controlled by power gating the memory, and SRAM aging rate is controlled by adjusting the duty cycle of each SRAM cell.

The software layer is responsible for algorithms pertaining to runtime resource and power budgeting, parallelization, and workload balancing. It also controls the power of the system by:

- Determining the most suitable voltage frequency of the underlying hardware
- Offloading subtasks from the cores to the loosely coupled hardware accelerators and to other devices
- Gating the clock of loosely coupled accelerators
- Selectively turning *on* sectors of the NVM for allowing access to the application
- Reducing the complexity (and hence the power consumption) of the application

The software layer also intakes the user requirements to run the system. For example, this information can contain the total duration for which the system must run and the number of applications (and the number of threads of each application) that needs to be processed in parallel. The software layer also considers the underlying hardware properties for workload balancing and power management.

In addition, it exploits the applications' and contents' properties to maximize the throughput-per-watt metric.

Details about design and runtime features provided by this system will now be provided.

3.1.1 Design Time Feature Support

As discussed above, the architectural design features of the video system include a many-core system, associated hardware accelerators, and a hybrid memory subsystem.

The many-core system provides parallelization support and allows multiple, multithreaded applications to execute on the cores. In this book, a core refers to an entity that can run a program written in a high- or low-level language. These cores can be homogeneous or heterogeneous. A per-core DVFS to appropriately scale the voltage-frequency levels is available. Also considered in [1, 2, 3, 4, 5], the per-core DVFS is now commercially available (e.g., 12-core Opteron), and it is indispensable for fine-grain power management. Further, Linux software libraries like "libdvfs" [6] expose the control of cores' frequencies (or "governors" of Linux) to the programmer, such that the application-level commands can directly communicate with the hardware.

Moreover, each core in the video system has an in-core, application-specific, hardware accelerator (see Fig. 2.16a). These in-core hardware accelerators are now available in commercial devices (e.g., custom instruction in Nios II processor [7], vector-processing units in Intel CPUs like SSE-SSE4 and AVX, AMD's 3DNow!, IBM's CPACF for cryptography functions). A programmer can write "special" or "custom" instructions (also sometimes called SI or CI for short) within their program to use the in-core hardware accelerator. Moreover, each core is connected to a global communication network using an interconnect fabric.

We abstract the workload of an application as a stream of tasks and a task runs on a core. A task can be further subdivided into subtasks. The subtasks from the programmable cores can be offloaded to the loosely coupled hardware accelerators, in order to increase the throughput-per-watt metric. It is assumed that the software version of the hardware accelerator is also available as a program, and the core decides whether to offload its tasks to the hardware accelerator or do the same task via software. The loosely coupled hardware accelerators communicate with the many-core system using an interconnect fabric.

In this system, the hardware accelerator acts as a slave to many masters, i.e., it cannot deny a compute command from the cores. Therefore, a scheduler is required to assist the accelerator. The accelerator scheduler receives offloading request from the programmable cores and assigns the accelerator to process subtasks of a core, one at a time, in a round-robin fashion. Further, the accelerators have distributed hardware components, and these components are fed with gated clocks, i.e., when a particular accelerator (or part of the accelerator) is not used, its clock can be gated

to save dynamic power consumption. Clock gating is available in industrial products like Open Multimedia Applications Platform (OMAP3) processor from Texas Instruments [8], Altera FPGAs, etc. The principle of power gating can also be employed to save leakage power in addition.

The hybrid memory unit includes a high-density NVM in conjunction with a SRAM. A programmer can use both NVM and SRAM as a scratchpad memory. Here, MRAM is chosen as a reference NVM; however, other NVM technologies can be also employed. The MRAM is sectored memory and the sectors are normally *off*, i.e., under normal operation, the MRAM sectors do not consume leakage or dynamic power. A particular sector of the MRAM is powered *on* only when it is required to read/write the data to that address. Hence, only at that instant, a little leakage power and some dynamic power are consumed by the particular MRAM sector tuned *on* by demand. The SRAM aging rate is controlled using aging reduction circuits and by intelligent address generation units (AGUs). The power consumed by these circuits is content dependent and can also be controlled by the programmer.

Furthermore, the software/application layer must be able to exploit the different aspects of hardware layer (e.g., the maximum frequency of the cores). The software layer (application or OS or any other middleware) should be able to adjust the voltage-frequency levels of the cores on demand of the application (see above the discussion about "libdvfs"). Furthermore, the workload of the applications should be structured in a manner to provide an ample number of independent subtasks such that the proposed parallelization techniques can pack these subtasks for a core with enough freedom and thereby determine the number of cores to utilize at runtime. These applications can also be malleable applications [9, 10]. In addition, note that the APIs for implementing parallelism, like OpenMP [11], support this behavior. However, the condition of being malleable is not necessary for these techniques to work, because an application can have a significantly large number of threads, which can be packed and dispatched to individual cores. Examples of such applications are mobile games, which use tens to hundreds of threads. Ideally, the applications also need to have a set of operating points, whereby the complexity of the application can be traded for a drop in the output quality (see discussion about reducing number of predictions in Sect. 2.2.1). Moreover, the applications need to know about the characteristics of the shared hardware accelerator (e.g., the number of cycles the accelerator will consume for every subtask).

The above discussion demonstrates that to increase power efficiency, different techniques can be utilized. By setting some configuration knobs, a tolerable output quality degradation can be allowed for gains in power savings. In summary, the following power configuration knobs are available for the video system:

- Per-core voltage-frequency knob (or per-core DVFS), to adjust the voltage-frequency and thus the power consumption and the time spent in processing a particular subtask
- Individual clock gating of accelerator modules, to cut off the clock (and hence eliminate the dynamic power consumption) to the modules which are not utilized

- Shared hardware accelerator scheduling, by which cores can offload their assigned subtasks to the accelerator and can be turned *off* (or cores can run at a lower frequency to process their subtasks in parallel) and, hence, save power
- Tunable application parameters or selection of application operating points, which can result in a lower complexity and contribute to power savings

3.1.2 Runtime Features and System Dynamics

During runtime, the video processing system needs to do different kinds of application- and content-dependent resource budgeting, which includes distributing cores and TDP among multiple, multithreaded tasks/applications. The loosely coupled accelerators are allocated to the processing cores and the clocks of the cores are appropriately gated. Moreover, the voltage-frequency levels of the cores should be correctly tuned, depending upon the throughput constraints and the features of the shared accelerators. Furthermore, the hybrid memory subsystem needs to be properly accessed and the aging rate of SRAM is controlled. We will now discuss these aspects in a little more detail.

The resource budgeting technique determines the number of compute cores that are allocated to a particular application. This resource budgeting depends upon the workload and the throughput requirements of an application. However, the resource budgeting must be fair to all applications and should not assign too much resources to a certain application, which will starve the other parallel running applications and reduce the overall performance. Moreover, resource budgeting needs to be adaptive and should increase/reduce the number of cores allocated to an application (called a cluster) at runtime, taking into account all the parallel running applications and the workload variations of all other parallel running applications. Ideally, the number of threads per application should equal the number of cores an application is allocated, in order to avoid overheads like resource locking and context switches associated with increasing number of threads. If certain conditions do not fulfill (e.g., no DVFS is available), then multiple threads of a single application can be mapped to a single processing core, which can process the workloads of these threads in a time-multiplexed manner, thereby reducing the number of cores required to meet the throughput requirements. Further, the resource budgeting technique should also allot the loosely coupled hardware accelerator to process a part of the workload of these applications.

The power budgeting technique distributes TDP at runtime among the applications (inter-cluster power distribution). The TDP distribution considers multiple factors like the size of the cluster, the power consumption, and the throughput generation history of the current application and other applications. Specifically, an application which used more power in the past will require more power with high probability. Further, once the TDP budget is distributed among the applications, the applications also need to distribute this budget among their parallel computing cores (intra-cluster power distribution). To balance the workload of each thread

(and hence maximally utilize the hardware for maximizing the throughput), application's power is transferred among the threads of the same application as well. This power actually translates to a certain voltage-frequency level, and a higher power allocated to a particular core means that the frequency of that core can be set to a higher value (i.e., the core will take less amount of time to process the workload).

The power distribution (or frequency allocation) is workload dependent, which might vary at runtime and may be different even for threads of the same application. Therefore, it is a prerequisite to develop a relationship between power and frequency required to process the workload corresponding to the application parameters (which control the complexity and output quality). Specifically, this relationship can be derived offline using system identification techniques like regression analysis. However, such a relationship might not be precise and cannot handle the workload variations. Further, this relationship will only be applicable to a particular core on a particular system, i.e., the portability of the application will be an issue. Parallel system workloads (like OS, parallel running applications on the same core) might also render this relationship inaccurate. Therefore, the video system tunes this relationship at runtime by getting software and hardware statistics. This also assists in balancing the workload, as a more accurate relationship can result in higher accuracy of power and resource distribution among the competing applications.

Runtime power control is also triggered by using the hardware accelerator's power gating signals, which are controlled by a register written by the program. These signals depend upon the architectural as well as the code currently executed on the architecture, thus requiring application and architectural awareness. Further, the sectors of MRAM are turned *on* that are used for read or write access, while the others are kept *off*. This *on*/*off* decision is also program flow dependent, and the control reduces the memory wake-up latency, by tuning its predictions to determine which memory sector will be turned *on* next. Moreover, the SRAM aging controller also considers the content properties that are written to the SRAM and adjusts the aging rate of the SRAM cells.

Regarding the discussion above, the following information passes between the layers. From software to hardware, the number of parallel threads, hardware accelerator assignment, voltage-frequency levels of the cores, and clock enable/disable signals are passed. On the other hand, hardware passes the time/cycles consumed in processing a subtask, hardware accelerator information, and many-core system's attribute (e.g., number of cores, minimum/maximum voltage-frequency settings) information to the software layer.

3.2 Application and Motivational Analysis

In this section, we analyze the video applications and impact of software and hardware layers on the execution of these applications is carried out. This analysis will be leveraged by the techniques discussed in this book.

3.2.1 Video Application Parallelization

Video processing applications usually consume many system resources and power, in order to meet their throughput constraints. An example is shown in Fig. 3.2, which shows the time and energy consumption per frame (with different resolutions) for encoding a video with HEVC. These experiments are performed for an x86 core running at 2.16 GHz. For example, encoding one HD frame (1920 × 1080) consumes ~8.2 s and 119.02 J. Note that increasing the core frequency to support high frame rates under real-time constraints is not scalable and viable due to power density issues (power-wall or dark silicon).

The total time consumed at various clock frequencies (f) and the size of the frame in blocks (n_{frm}) are shown in the color-coded plot of Fig. 3.3a. As expected, increasing f reduces the time consumption and increasing n_{frm} increases time

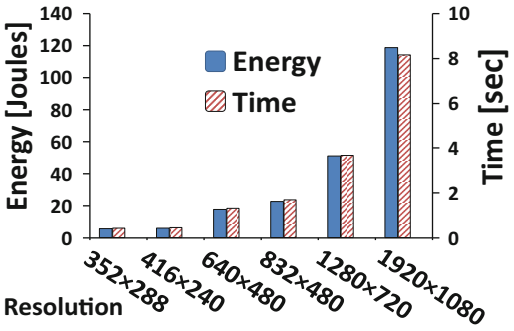


Fig. 3.2 Energy and time consumption for HEVC Intra-encoding on a single x86 core using different resolutions

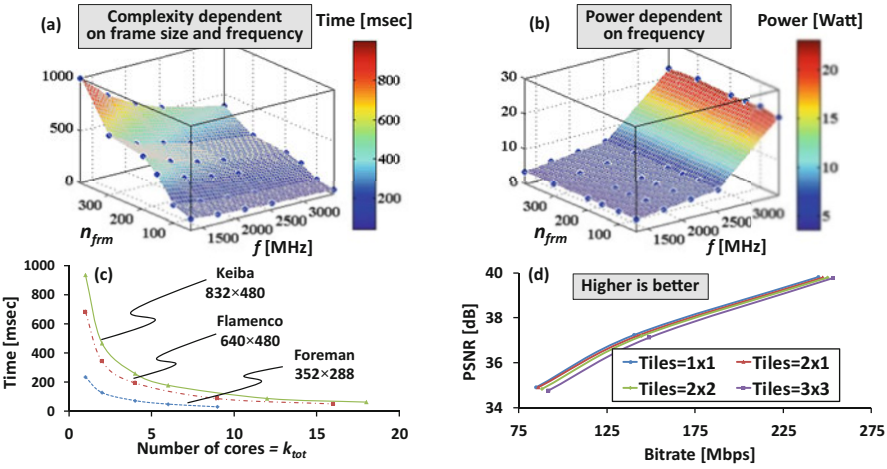


Fig. 3.3 (**a**) Average time at different clock frequencies; (**b**) power consumption for different frequencies and frame sizes; (**c**) at $f = 2.16$ GHz, average time per frame with varying number of cores; and (**d**) PSNR vs. bitrate plot by using "Foreman" video sequence (352 × 288) for HEVC encoding

consumption of the core. Hence, for processing n_{frm}, a particular frequency of the core can be determined which will not consume more than a predefined (constant) amount of processing cycles. That is, given n_{frm}, the timing constraints can be satisfied by appropriately selecting the frequency. The color-coded plot in Fig. 3.3b shows the dynamic power consumption of a single core in terms of n_{frm} and f for HEVC encoding. Increasing f results in high dynamic power consumption and vice versa. Power is independent of n_{frm} as the power formula only considers the voltage and working frequency of the core. Therefore, it is possible to select an appropriate clock frequency of the core that will limit the total power consumption below a certain limit. In other words, the power constraint of the core requires an appropriate selection of the core's frequency. Therefore, using the allocated power to a core, its frequency could be determined.

To enable real-time video processing, video applications provide inherent support for parallelization. For example, in HEVC standard, parallelization support occurs in the form of slices, tiles, and Wavefront Parallel Processing (WPP) [12]. To efficiently utilize the hardware resources and to reduce power consumption in a many-core processor, there is a need to (1) ascertain the parallel processing workload and (2) reduce the number of active cores while fulfilling the throughput constraints. Figure 3.3c denotes the total time consumed for encoding a single frame using HEVC for changing the number of processing cores (k_{tot}) at $f = 2.16$ GHz. We notice that increasing k_{tot} reduces the time consumption by an approximately second-degree relation. Thus, appropriate k_{tot} can be selected that can encode the video frame within the given amount of time, i.e., for satisfying the time/fps constraint. Using tile-based parallelism in HEVC, the output video quality for different configurations of tiles (and also different parallelism) is shown in Fig. 3.3d. Here, a tile is processed on a single core. However, to enable parallelism by dividing the video frame into parts such that each part can be processed independently of other parts by breaking the sequential dependencies across the boundaries, a potential video quality loss may occur [12]. Since tiles break coding dependencies, therefore, increasing number of tiles also results in video quality loss, and a single tile per frame results in the best video quality. Since parallelization is indispensable, therefore, with u^2 tiles per frame, a $u \times u$ tile structure (u columns and u rows) should be used for the best video quality. Therefore, the total number of tiles per frame must be minimized such that it just fulfills the workload of HEVC encoding. This will not only increase the video quality but also reduce the power consumption of the system. If a perfect $u \times u$ tile structure is not possible, then effort must be made to make the total tile columns and row as similar as possible.

Additionally, in video applications, the collocated tiles (same tiles of consecutive video frames, see Fig. 2.10b) exhibit high correlation in computational and output characteristics. For HEVC, correlation of complexity and the output compressed bytes between collocated tiles is given in Fig. 3.4. This correlation is plotted as a histogram of percentage difference in time (Fig. 3.4a) and total bytes per tile (Fig. 3.4b). Larger crowding around zero shows a high correlation. Therefore, time

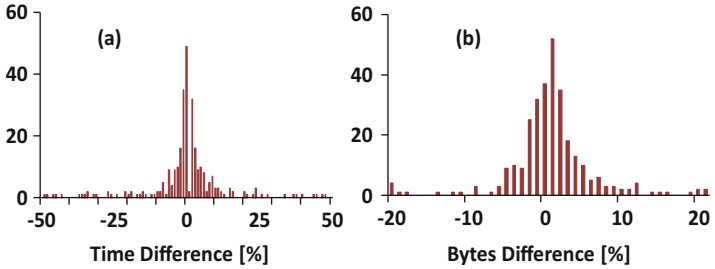


Fig. 3.4 Percentage difference histogram of (**a**) time and (**b**) bytes for collocated tile 0 of "Foreman" sequence (352 × 288) for 300 frames

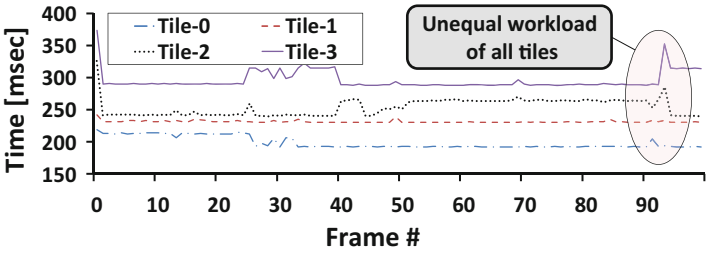


Fig. 3.5 Time consumption of tiles (2 × 2 = 4 tiles per frame) of nonlocal means filter for "Foreman" sequence (352 × 288)

complexity and output quality (bit-rate) of the current tile can be estimated from previous collocated tile(s), and this can be used to estimate the workload of the current tile.

3.2.2 Workload Variations

The workload of a thread varies at runtime, whether it is the only thread of the application or one of the many threads of a multithreaded application. Figure 3.5 plots the time consumed by each tile for NLMF application of the sequence "Foreman" (352 × 288). A tile structure of $u_w \times u_h = 2 \times 2$ (two tile columns and two tile rows per frame) is specified for this experiment. As seen, in addition to the complexity of a tile different from the rest, the complexity of the tile also varies at runtime. This complexity is a direct measure of the workload. Additionally, if timing constraints are imposed, the power consumption of the cores will also vary. Therefore, it becomes imperative to regulate the workload of each compute core individually. This adaptation should depend upon the throughput requirement, as well as the hardware resources of the underlying many-core system.

This workload variation can be due to multiple factors. Some of these factors are described below:

- Some threads are assigned more data to process. There are multiple reasons for the non-uniform data allocation. Like, it may not be possible to get equal-sized tiles by having odd number of blocks within a row (or column) of a video frame, e.g., 11 blocks per row of the video frame that must be divided into four tiles.
- In many video applications (particularly video coding), the number of processor cycles required for processing a block is also content dependent. For example, ME will spend more time for a tile with high motion blocks compared to a tile containing low motion blocks.
- Some video tiles may require a higher quality than other tiles. This is possible because region of interest (ROI)-based processing requires a better video quality at the expense of more calculations while processing the ROI within a video frame. Therefore, a thread that processes the ROI may have more complexity than the rest. An example could be detection of eyes within a video, whereby the face can be the ROI. Here, face is detected first within the video frame, and then eye localization algorithm is performed on that region only. Similar examples exist for video coding, where the ROI includes faces, moving objects like cars, etc. In such cases, compression is kept comparatively lower for a better visual effect for ROI regions, whereas the background regions (non-ROI) are aggressively compressed.
- Parallel system workloads (due to OS, kernel, or parallel running applications) may render a thread of an application to run slower than the others.
- Heterogeneity among the compute cores can exist which can in turn run the core slower/faster. For example, some cores might have bigger caches than the rest, and some cores are designed to run at a higher frequency (or some cores can run at higher frequencies due to process variations [13]).

The fundamental takeaway from the discussion above is that workload variations cause unequal complexity of threads of a video system, which reduces the throughput of the application due to load imbalance. This is intuitive because minimum time for processing a video frame will occur if all the threads finish processing their tasks/tiles at the same time. Hence, to achieve maximum benefit from the underlying hardware, it is necessary to balance the workload of parallel computing jobs at runtime.

Note that there can be other parallel running threads of the application, used for management jobs unlike the number-crunching threads (e.g., which process a video tile). In this book, we will ignore these management threads and assume their impact is negligible.

3.2.3 HEVC Complexity Analysis

As previously discussed, HEVC is an important next-generation application pertaining to video coding. By dividing the CTU iteratively into respective CUs, PUs, and TUs, HEVC tries to maximally exploit the redundancies within an image.

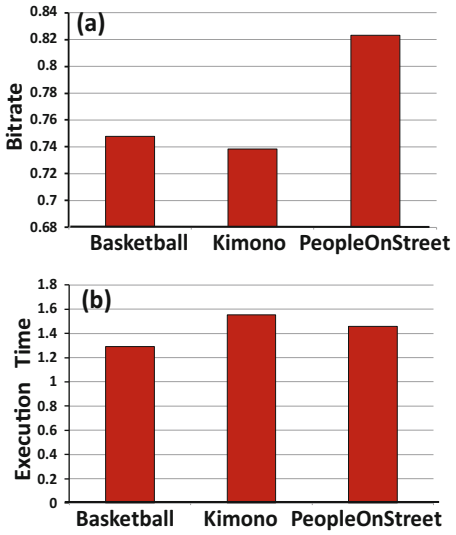


Fig. 3.6 Normalized (**a**) bit rate and (**b**) execution time of HEVC with reference to H.264 for different video sequences

This iterative and recursive behavior incurs substantial complexity overhead, even for intra-only encoders, because the RDO decision has to recursively check each possible PU and intra-mode combination (see Fig. 2.6). This enormous decision space significantly enlarges the computational complexity compared to H.264. Our experiments in Fig. 3.6 show that compared to intra-only H.264/AVC encoder, the computational complexity of intra-only HEVC has increased by a factor of ~1.4× for a compression efficiency increase of around 35%. A similar analysis is reported in [14]. This demonstrates a significant challenge towards designing fast HEVC encoders. Therefore, it is essential to develop fast algorithms to decrease the computational complexity of HEVC encoders and to realize real-world applications.

3.2.3.1 Texture and PU Size Interdependence

In Fig. 3.7, the borders of PUs (generated via RDO decision) are plotted on top of the frame of the "Sheilds" video sequence. The PU sizes and their corresponding locations illustrate that PU size decision is based upon the texture (or variance) of the video frame content. Note that the RDO iterative decision process of selecting the best PU sizes selects smaller blocks for image regions with high variance and texture details. For example, in region A of Fig. 3.7, a video frame region with low texture is encoded using larger PU sizes, whereas in region B, a smaller PU size and dense PU partitioning are usually selected for highly detailed texture. An interesting case is presented in region C where it is noticed that some areas are encoded using small-sized PUs and the uniform areas of the block are encoded using larger PUs.

This analysis illustrates that a complexity reduction technique can estimate the sizes of PU using video content, instead of performing high-complexity RDO

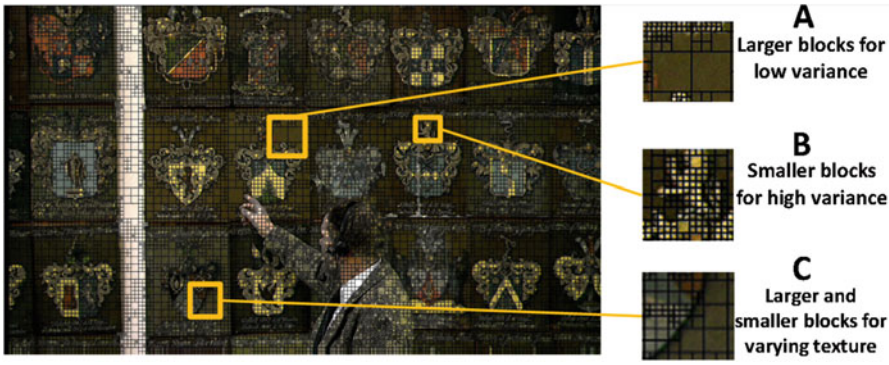


Fig. 3.7 PU borders on the third frame of "Sheilds" sequence

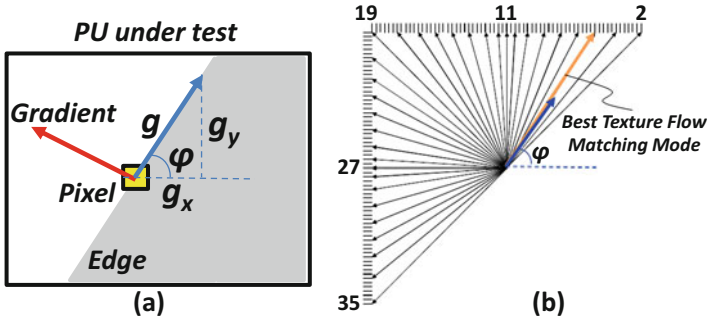


Fig. 3.8 Estimating intra angular mode selection

decisions employing a full search to determine the right PU sizes. The PU size generation (involving early PU size estimation, i.e., before the RDO) needs to account for the video texture (i.e., variance) of frame regions in order to curtail the RDO search space for fast mode evaluations. If such a "map" of the PU sizes and locations is available, users can bypass the RDO process for improbable PU sizes and, hence, reduce the complexity.

3.2.3.2 Edge Gradients and Intra Angular Modes

The final intra angular mode selected by the RDO process by testing all the possible angular modes can be a time-consuming process. However, one can infer the coarse intra angular mode by determining the direction of the gradient as shown in Fig. 3.8. Usually the texture flow angle (perpendicular to the gradient) is selected as the intra-prediction [15, 16] of the PU. With high probability, the intra-prediction mode lies in that vicinity of the angular direction perpendicular to the direction of the gradient. We can exploit this knowledge to reduce the number of intra angular

modes tested for HEVC and achieving complexity savings. Depending upon the gradient of the video frame block, only a reduced set of highly probable intra-prediction modes can be tested, excluding the unlikely modes. A similar case can be employed for inter-encoding, whereby the complexity of ME can be reduced by observing the video properties.

3.3 Hardware Platform Analysis

This section provides analysis about the architectural aspects of a video processing system.

3.3.1 Heterogeneity Among Computing Nodes

For presenting the impact of heterogeneity on different characteristics of the system, we provide different compute nodes (here, programmable cores) and benchmarks, as given in Table 3.1. These numbers are obtained via Sniper simulator [17] and McPAT [18] for 45 nm x86 cores, with 32 KB L1 data and instruction caches. The average cycles per operation are obtained by running the benchmarks for the set of frequencies given in Fig. 3.9. The variance of the cycles per

Table 3.1 Characteristics of cores and benchmarks

Core	Attributes	Area [mm^2]	Average cycles per operation		
			DCT	Quant	HEVC
Tiny	L2 = 64	86.76	4594	3416	100×10^6
	Dispatch = 1				
Medium	L2 = 256	89.99	2378	1658	59.0×10^6
	Dispatch = 2				
Large	L2 = 512	95.26	1687	971	45.3×10^6
	Dispatch = 4				

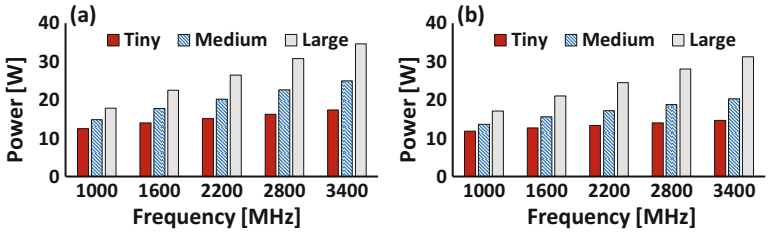


Fig. 3.9 Power and frequency profiles for (**a**) DCT and (**b**) quant

benchmark is negligible for varying frequency, and, therefore, the average numbers are reported.

Figure 3.9 also provides the power profiles of the cores. For the "quant" benchmark (H.264/AVC quantization), although the number of cycles consumed by the "large" core is ~3.5× lesser than the "tiny" core, the power consumed by the "large" core at the lowest frequency (i.e., 1000 MHz) is greater than the power consumed by the "tiny" core at maximum frequency (3400 MHz). This discussion conveys that heterogeneous nodes result in varying complexity (in number of cycles consumed) and power consumption for the same workload. Therefore, the workload balancing techniques must consider the complexity and power characteristics of the underlying compute nodes. Further, notice that the power consumption is approximately independent of the type of application used in this benchmark.

3.3.2 Memory Subsystem

Recent studies have shown that memory is one of the main energy-consuming system modules [19], especially in video processing/compression systems [20, 21]. Characteristically, in video applications, large frame dimensions and high processing rates put huge pressure on the off-chip and on-chip memories [22]. The primary reason is that the storage and repetitive accesses to large-sized video frame buffers [23, 24] result in high energy consumption. For instance, video coding of 4K UHD at 30 fps results in a memory access rate of >356 megapixels per second, i.e., 2.78 Gbps. In addition, each frame will require ~12 MBytes of on-chip memory.

Video frames are usually stored in the high-capacity off-chip memory. To avoid frequent accesses to the off-chip video memory or to lighten the external memory transfers, state-of-the-art video applications deploy on-chip video memories to store parts or full video frames [25, 26]. Therefore, data must be brought from the off-chip memory to the on-chip memory (typically SRAM) in order to lessen the access latency and external memory bus contention [27]. Moreover, several repetitive accesses are made to the same pixel locations in advanced video coding standards due to multilevel filtering and excessive ME operations. Therefore, larger on-chip memories are indispensable to improve the performance and energy efficiency of video coding applications. On the other hand, large on-chip memories also contribute in high power/energy consumption due to:

- High leakage power, as a result of larger on-chip memories to store video frames (e.g., reference and current video frames in video coding application). Traditional SRAM-based on-chip memories have high silicon footprint and leakage power that may even surpass their dynamic power. A reduction in the leakage power may be obtained by exploiting emerging on-chip memory types like NVMs.

- High dynamic power, as a result of bigger resolutions, high frame rates, and complex processing flow of advanced video applications (like HEVC, which employs multiple filters and a complex multimode ME process).

Therefore, to significantly cut the overall energy consumption of on-chip video memory subsystem, application-specific memory subsystem architectural optimizations and energy management are required.

3.3.2.1 Analysis of Motion Estimation

Figure 3.10a shows the memory access requirements for ME algorithm using a single reference frame. Three different search window sizes ($s_w \times s_h$) are selected for ME, for both HEVC and H.264/AVC. HEVC memory access requirements are ~3.86× more than H.264/AVC. This shows that HEVC puts high pressure on the memory system by generating a large amount of memory read accesses. H.264/AVC-based external memory access reduction techniques thus may not scale properly if applied to HEVC. Furthermore, the search window method requires a

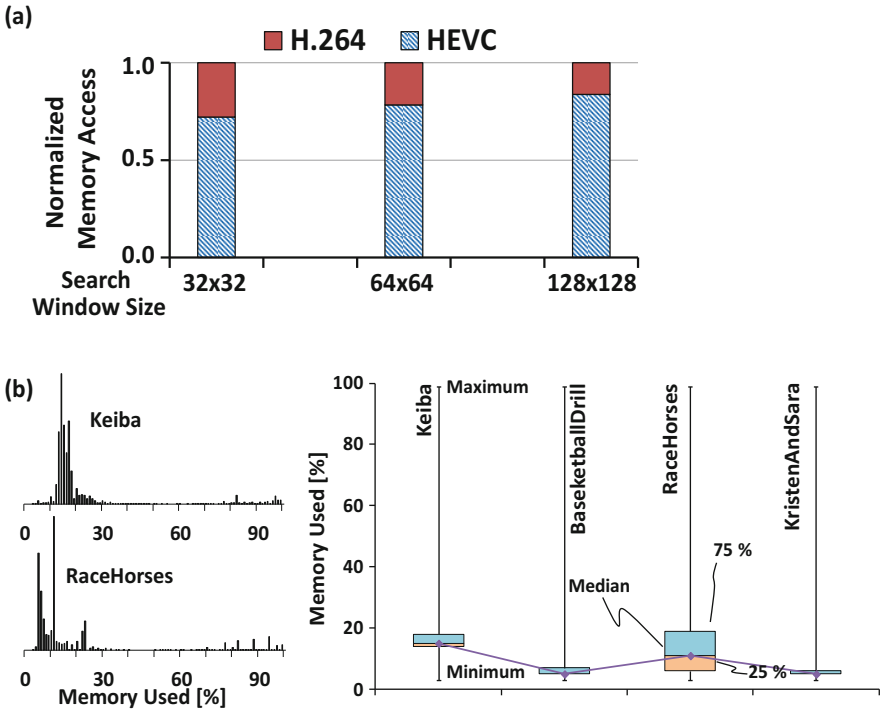


Fig. 3.10 (**a**) Using different search window sizes, ME memory access comparison between H.264/AVC and HEVC, (**b**) ME memory access statistics when using TZ search technique for ME in HEVC

reference frame to be read at least $r_f = s_h/b_h \geq 3$ times from the external memory into the search window (see discussion in Sect. 2.2.1.2). Using a single search window of size 256 × 256, a 4K UHD 30 fps video with $b_h = 64$ will therefore require 11.12 Gbps fetched from the external memory only for the reference frame. With n_r reference frames and additional read and write to the external memory for the current frame, a total of $(11.12n_r + 2.78)$ Gbps are read, and around 2.78 Gbps are written as reconstructed frame. Note that the external bus power dissipation is directly proportional to the total number of bit toggles per transition [28]. This massive amount of data results in high latency and energy consumption (around 40% of the total system energy is consumed by the external I/O [29, 30]). Therefore, saving external memory accesses becomes vital for reducing the energy consumption of a video encoding system. Thus, using large on-chip frame buffers will reduce the dynamic energy consumption and the external memory bus contention. However, their contribution to the leakage energy becomes a design challenge.

Using the state-of-the-art ME algorithm, TZ search [31], Fig. 3.10b shows the histogram of percentage memory accessed within the search window in HEVC for different video sequences. A search window of size 64 × 64 (or 36 KB) is taken. These graphs are averaged per frame. The box plots for these statistics are also given, and we notice that less than 20% of the search window is utilized (see discussion in Sect. 2.2.1.2), i.e., most of the times, only a part of the search window is utilized. The unused search window consumes leakage power. Therefore, adapting the search window according to the needs of the block-matching algorithm can result in high energy gains.

3.3.2.2 Hybrid Memories

For general memory subsystems, in order to reduce the leakage energy consumption and to reduce the latencies and dynamic energy associated with the write operation, research has focused on combining the best of NVMs and VMs and coined the term "hybrid memory" [32, 33]. An equivalent principle can be applied for on-chip video memory required by video applications like ME. However, from Table 2.2, it is evident that straightforward replacement of the SRAM- or DRAM-based memories with NVMs is impractical. Special thought to the memory characteristics must be involved in the system design. We can reduce the leakage energy of a system by introducing NVMs in place of traditional VMs (like SRAMs or DRAMs), but at the same time, NVMs can play a major role in worsening system response time and dynamic energy consumption. Therefore, the advantages and disadvantages of NVMs must be carefully weighed.

The on-chip video frame buffers can be either SRAM- or MRAM-based memories. A system designer needs to consider the application behavior, usage constraints, and system dynamics before employing the memory subsystem, which would result in the best benefit-to-cost ratio. For example, using Table 2.2, we can generate a benefit-to-cost technology precedence by adding the blues for each technology and subtracting the reds from it. For example, if the application requires

Table 3.2 Memory attributes for 65 nm technology [34]

	Latency (*nsec*)		Energy and power			
			Dynamic (*nJ*)			
Memory type	Read	Write	Read	Write	Leak. (*W*)	Area (mm^2)
SRAM – 4 MB	4.659	4.659	0.103	0.103	5.20	44
DRAM – 16 MB	5.845	5.845	0.381	0.381	0.52	49
MRAM – 16 MB	4.693	12.272	0.102	2.126	0.97	38

processing large data sets with high data reuse (high number of reads and low writes in a memory), MRAM is a better option than SRAM, DRAM etc.

Attributes of different memory subsystems are tabulated in Table 3.2. From this table and Table 2.2, notice that the SRAM and MRAM read latencies and energies are similar. On the contrary, the leakage energy of SRAM is ~21× of MRAM of the same capacity. In addition, MRAM capacity is about four times that of SRAM for the same area. Therefore, MRAM is a superior candidate for on-chip video frame buffer. However, the write energy and latency of MRAM are ~20× and 2.6× of SRAM, respectively. Thus, feeding the MRAM from external memory is not ideal, as the external memory read latency added with the write latency of MRAM could severely degrade system power efficiency.

3.3.3 *Analysis of Different Aging Balancing Circuits*

In this section, we provide a detailed aging analysis, in terms of duty cycle imbalance, for different test video sequences [35, 36] and highlight issues related to image regions with features that result in different SNM degradation of the memory. This analysis is leveraged for developing the aging-resilient video memory discussed in this book.

As discussed in the previous chapter, the aging rate is minimized if both the PMOS transistors in the 6T SRAM cell are stressed equally (i.e., duty cycle, $\Delta = 0.5$). One way of equally stressing the transistors is to overwrite the SRAM cell with complementary bit in every cycle. Figure 3.11 shows the total percentage of bits (in form of a histogram) for a frame memory that is overwritten by a complementary bit of the new frame at the same position. Usually, for video sequences with low activity (low motion, no camera panning, etc.), this histogram is crowded towards smaller percentages, which tells us the writing new frame into SRAM will only release stress on some 6T cells (as the duty cycle will be highly biased towards 0.0 or 1.0). Additionally, the histogram for these sequences is not dispersed, demonstrating that there is significant correlation in terms of texture and motion between temporally neighboring frames. Moreover, the properties of the subsequent frames can be estimated from the history, such that it can be leveraged for efficient aging balancing. In a nutshell, we can predict the aging impact of the

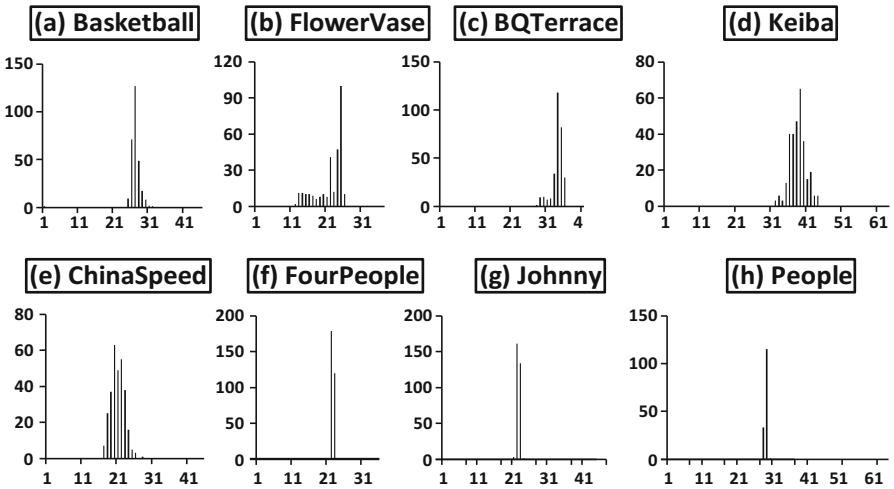


Fig. 3.11 Histogram of percentage SRAM memory overwritten with complementary bits by video sequences. X-axis presents the percentage of bits changed in the subsequent frames, and y-axis presents the number of times a certain percentage occurs for 300 continuous frames

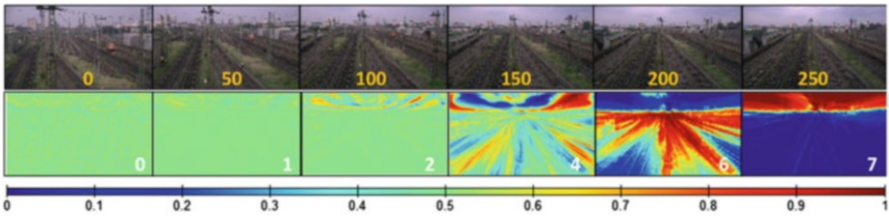


Fig. 3.12 (*Top*) Frames of the video sequence "Station" (1920 × 1080) with frame numbers written and (*bottom*) Δ stressmaps for specific bits of the video sequence, by plotting duty cycle (Δ) of specific bit on spatial scale

current and future video frame on the SRAM, by analyzing the aging effects of the previous video frames.

In Fig. 3.12, duty cycle of some selected bits of the luminance component of "Station" video sequence is plotted on the spatial scale, in the form of "stressmaps." A stressmap represents the value of duty cycle per pixel, for a given bit position. A balanced duty cycle ($\Delta = 50\%$ or 0.5) is represented with a light greenish color (see the color-coded scale below the pictures). For duty cycles heavily biased towards "0" and "1," we obtain a blue- and a red-colored distribution in the stressmaps, respectively. Notice that lower-order bits (LSB bits) have a balanced duty cycle; thus, the 6T SRAM cells storing these bits have consistent relaxations and, therefore, an extended lifetime. However, the higher-order bits (MSB bits) have a highly biased duty cycle, which causes an imbalance in the 6T SRAM cells storing MSBs, i.e., one out of the two PMOS transistors of the SRAM cell is under increased stress.

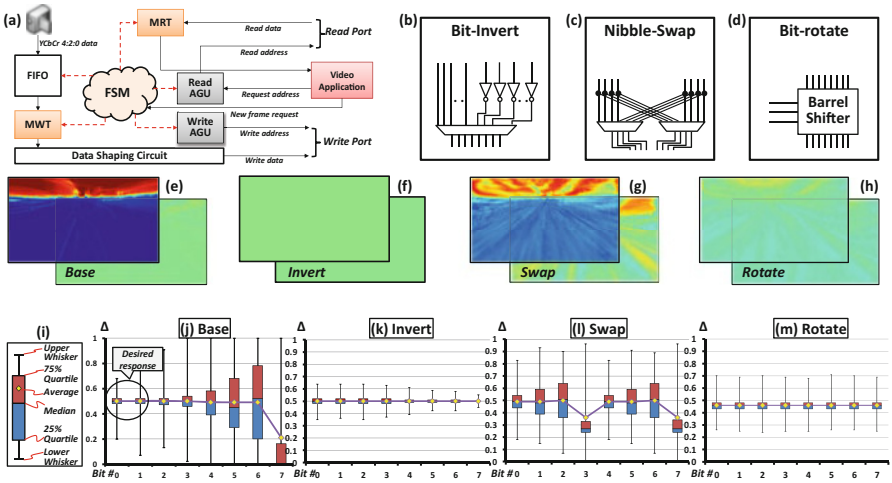


Fig. 3.13 (**a**) Inserting aging resiliency components (MWT and MRT) in video memory management of Fig. 2.2. (**b**) Bit-inversion MWT, (**c**) nibble-swapping MWT, and (**d**) bit-rotation MWT. (**e**–**h**) Stressmaps for bit-7 (*foreground*) and bit-0 (*background*) for the above MWT. (**i**) *Box plot* legend; (**j**–**m**) *Box plots* for the above MWTs

In summary, duty cycles of SRAM cells are not balanced and some cells age faster than others. In such cases, it becomes crucial to balance duty cycle of each bit individually and to leverage the knowledge of bit location before applying an aging resiliency technique.

The less frequently varying video samples will introduce the most amount of stress on the 6T SRAM cells, for instance, low complexity texture and large, static backgrounds. Different mission critical video applications like security surveillance and space exploration experience long duration static scenes, which will adversely influence the SNM. Therefore, the knowledge of less and more frequently changing data locations can be exploited to distribute video samples in the on-chip memory, such that each transistor of the SRAM memory experiences some relaxation.

In order to balance the duty cycle, we extend the memory architecture of Fig. 2.2 with additional aging resiliency tools in form of Memory Read Transducer (MRT) and Memory Write Transducer (MWT). These transducers are connected to the memory read/write ports (see Fig. 3.13a). The transducers can be implemented using many architectural flavors. Three commonly employed designs for the transducer circuits are shown in Fig. 3.13b–d. We investigate inversion (similar to [37]), nibble swapping (similar to [38]), and bit rotation (similar to [39]), which correspond to the state-of-the-art techniques to tackle SRAM aging. For inversion and nibble swapping, the bits of every second frame are inverted and swapped, respectively. For the bit-rotation transducer, the video sample bits are incrementally rotated by one bit position with every frame, before writing to the frame memory location.

In Fig. 3.13e–h, the stressmaps of bits 0 and 7 with no balancing (the base case) and the balancing circuits presented in Fig. 3.13b–d are given. As noticed, the duty cycle for the inverter case is greatly balanced compared to the other circuits. For convenience, the duty cycles for all bits are presented in the form of box plots, as shown in Fig. 3.13i–m. In the box plot, the distribution of duty cycle of each bit is mapped to quartiles. In an ideal case, in the box plot, the spread of whiskers should be minimal and the median of the box plot should be at "0.5." The spread biased towards "0" indicates a larger number of "zeros" at the bit location and vice versa. For the base case (i.e., without using MWT/MRT) and lower-order bits (bits "0" and "1"), the spread of duty cycle is limited and the median is closer to "0.5." However, for higher-order bits (bits 3–7), the spread of duty cycle is large, and the median is not strictly "0.5." This suggests that the higher-order bits in the video sequence need more consideration, and resiliency features embedded into the system must account for these bits. Furthermore, the lower-order bits experience auto-balancing and the aging resiliency features for these bits are not essential.

By using balancing circuits of Fig. 3.13b–d, the box plots change significantly. For inverters, we notice that the duty cycle is nicely balanced for each bit and the spread of the duty cycle is limited. The nibble-swapping technique introduces some improvement in balancing the duty cycle, but it is not comparable to that of the inverter. Moreover, the nibble swapping also harshly impacts the duty cycle of bits "0" and "1." Bit rotation fits in the middle of the inversion and nibble-swapping cases. Moreover, by keeping the inverter *on* for all bits, at all the time, is not energy efficient. Therefore, the challenge is to design an adaptive, configurable controller to select the frame inversion rate and bits to invert at runtime.

3.4 Summary

In this chapter, we performed a detailed motivational analysis of video processing systems, regarding their software and hardware layers. At the software layer, our focus is to develop efficient parallelization techniques, which can be used to increase the throughput-per-watt ratio of the system. Further, workload distribution and balancing are discussed. The main focus at the hardware layer is to tackle heterogeneity and efficiently utilize the memory subsystem to increase the performance of the complete video processing application. In the coming chapters, we will leverage the knowledge gained by the analysis performed in this chapter and design techniques to exploit this information.

References

1. Ma, K., Li, X., Chen, M., & Wang, X. (2011). Scalable power control for many-core architectures running multi-threaded applications. In *Internation Symposium on Computer Architecture*.
2. Sharifi, A., Mishra, A., Srikantaiah, S., Kandemir, M., & Das, C. R. (2012). PEPON: Performance-aware hierarchical power budgeting for NoC based multicores. In *Parallel Architectures and Compilation Techniques*.
3. Jevtic, R., Le, H.-P., Blagojevic, M., Bailey, S., Asanovic, K., Alon, E., & Nikolic, B. (2015). Per-core DVFS with switched-capacitor converters for energy efficiency in manycore processors. *IEEE Transactions on Very Large Sclae Integration Systems (TVLSI), 23*(4), 723–730.
4. Kim, W., Gupta, M., Wei, G.-Y., & Brooks, D. (2008). System level analysis of fast, per-core DVFS using on-chip switching regulators. In *International Symposium on High Performance Computer Architecture (HPCA)*.
5. Lee, W., Wang, Y., & Pedram, M. (2014). VRCon: Dynamic reconfiguration of voltage regulators in a multicore platform. In *Design, Automation & Test in Europe and Exhibition (DATE)*.
6. libdvfs. [Online]. Available: https://github.com/LittleWhite-tb/libdvfs. Accessed 12 Aug 2015.
7. Nios II Custom Instruction User Guide. Altera, (2011).
8. OMAP 3 Processor. Texas instruments, [Online]. Available: http://www.ti.com/product/omap3530. Accessed 13 Aug 2015.
9. Feitelson, D. G., & Rudolph, L. (1996). Towards convergence in job schedulers for parallel supercomputers. In *Workshop on Job Scheduling Strategies for Parallel Processing*.
10. Desell, T., Maghraoui, K. E., & Varela, C. A. (2007). Malleable applications for scalable high performance computing. *Cluster Computing, 10*(3), 323–337.
11. OpenMP. [Online]. Available: http://openmp.org/wp/. Accessed 12 Aug 2015.
12. Chi, C. C., Alvarez-Mesa, M., Juurlink, B., Clare, G., Henry, F., Pateux, S., & Schierl, T. (2012). Parallel scalability and efficiency of HEVC parallelization approaches. *IEEE Transactions on Circuits and Systems on Video Technology, 22*(12), 1827–1838.
13. Kuhn, K., Kenyon, C., Kornfeld, A., Liu, M., Maheshwari, A., Shih, W.-k., Sivakumar, S., Taylor, G., VanDerVoorn, P., & Zawadzki, K. (2008). Managing process variation in intel's 45nm CMOS technology. *Intel Technology Journal, 12*(2), 93–109.
14. Nguyen, T., & Marpe, D. (2012). Performance analysis of HEVC-based intra coding for still image compression. In *Picture Coding Symposium (PCS)*.
15. Shafique, M., Molkenthin, B., & Henkel, J. (2010). An HVS-based adaptive computational complexity reduction scheme for H.264/AVC video encoder using prognostic early mode exclusion. In *Design, Automation and Test in Europe Conference (DATE)*.
16. Jiang, W., Mal, H., & Chen, Y. (2012). Gradient based fast mode decision algorithm for intra prediction in HEVC. In *International Conference on Consumer Electronics, Communications and Networks*.
17. Carlson, T., Heirman, W., & Eeckhout, L. (2011). Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation. In *High Performance Computing, Networking, Storage and Analysis*.
18. Li, S., Ahn, J. H., Strong, R., Brockman, J., Tullsen, D., & Jouppi, N. (2009). McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Microarchitecture*.
19. Barroso, L., & Holzle, U. (2007). The case for energy-proportional computing. *Computer, 40*(12), 33–37.
20. Ostermann, J., Bormans, J., List, P., Marpe, D., Narroschke, M., Pereira, F., Stockhammer, T., & Wedi, T. (2004). Video coding with H.264/AVC: Tools, performance, and complexity. *IEEE Circuits and Systems Magazine, 4*(1), 7–28.

21. Sullivan, G. J., Ohm, J., Han, W., & Wiegand, T. (2012). Overview of the high efficiency video coding. *IEEE Transactions on Circuits and Systems for Video Technology, 22*(12), 1649–1668.
22. Shafique, M., Zatt, B., Walter, F. L., Bampi, S., & Henkel, J. (2012). Adaptive power management of on-chip video memory for mul-tiview video coding. In *Design Automation Conference*.
23. Stancu, L. C., Bathen, L. A. D., Dutt, N., & Nicolau, A. (2012). AVid : Annotation driven video decoding for hybrid memories. In *Embedded Systems for Real-Time Multimedia*.
24. Bathen, L., & Dutt, N. (2012). HaVOC: A hybrid memory-aware virtualization layer for on-chip distributed scratchpad and non-volatile memories. In *Design Automation Conference*.
25. Chen, C., Huang, C., Chen, Y., & Chen, L. (2006). Level C+ data reuse scheme for motion estimation with corresponding coding orders. *IEEE Transactions on Circuits and Systems for Video Technology, 16*(4), 553–558.
26. Gangadharan, D., Phan, L., Chakraborty, S., Zimmer-mann, R., & Insup, L. (2011). Video quality driven buffer sizing via frame drops. In *Embedded and Real-Time Computing Systems and Applications (RTCSA)*.
27. Tsung, P.-K., Chen, W.-Y., Ding, L.-F., Chien, S.-Y., & Chen, L.-G. (2009). Cache-based integer motion / disparity estimation for Quad-HD H.264/AVC and HD multiview video coding. In *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*.
28. Ning, K., & Kaeli, D. (2005). Power aware external bus arbitration for system-on-a-chip embedded systems. *High Performance Embedded Architectures and Compilers, 3793*, 87–101.
29. Sze, V., Finchelstein, D. F., Sinangil, M. E., & Chandraksan, A. P. (2009). A 0.7-V 1.8-mW H.264/AVC 720p video decoder. *IEEE Journal of Solid-Sate Circuits, 44*(11), 2943–2956.
30. Ma, Z., & Segall, A. (2011). Frame buffer compression for low-power video coding. In *International Conference on Image Processing*.
31. Purnachand, N., Alves, L. N., & Navarro, A. (2012). Improvements to TZ search motion estimation algorithm for multiview video cod-ing. In *IEEE International Confernce on Systems, Signals and Image Processing (IWSSIP)*, pp. 388–391.
32. Wu, X., Li, J., Zhang, L., Speight, E., Rajamony, R., & Xie, Y. (2009). Hybrid cache architecture with disparate memory technologies. In *International Symposium on Computer Architecture (ISCA)*.
33. Loh, G. H. (2009). Extending the effectiveness of 3D-stacked DRAM caches with an adaptive multi-queue policy. In *International Symposium on Microarchitecture (MICRO)*.
34. Dong, X., Wu, X., Sun, G., Xie, Y., Li, H., & Chen, Y. (2008). Circuit and microarchitecture evaluation of 3D stacking magnetic RAM (MRAM) as a universal memory replacement. In *Design Automation Conference (DAC)*.
35. Bossen, F. (2012). Common test conditions. Joint Collaborative Team on Video Coding (JCT-VC) Doc. I1100.
36. Common YUV 4:2:0 test sequences. [Online]. Available: https://media.xiph.org/video/derf. Accessed 16 Aug 2015.
37. Siddiqua, T., & Gurumurthi, S. (2010). Recovery boosting: A technique to enhance NBTI recovery in SRAM arrays. In *Annual Symposium on VLSI*.
38. Amrouch, H., Ebi, T., & Henkel, J. (2013). Stress balancing to mitigate NBTI Effects in register files. In *Dependable Systems and Networks (DSN)*.
39. Kothawade, S., Chakraborty, K., & Roy, S. (2011). Analysis and mitigation of NBTI aging in register file: An end-to-end approach. In *International Symposium on Quality Electronic Design (ISQED)*.

Chapter 4
Energy-Efficient Software Design for Video Systems

This chapter provides details about the runtime supervision of the video processing system at the software layer. The main responsibilities addressed in this layer are to allocate compute nodes, realize power efficiency and budget power to the video system. In order to parallelize the execution of a video application, resources are allocated to the application at runtime, by considering the user demands, and application and hardware attributes of the system. Further, the workload is distributed among the compute nodes in a way that throughput-per-watt is increased. Video application properties are also exploited at runtime, and these properties are used to adjust the configuration knobs, which leverage power/complexity with the output video quality. Moreover, resource and power allocation to multiple applications running concurrently on multi/many-core homogeneous and heterogeneous systems are also discussed.

4.1 Power-Efficient Application Parallelization

As discussed in previous chapters, a video application usually processes a block of pixels at one time. Consider that a block of pixels processed by the application is treated as a job (e.g., a MB in H.264/AVC and CTU in HEVC). A set of jobs is given by a subtask, η. And a set of subtasks collectively constitute a task $\boldsymbol{\eta}$. To imagine the hierarchy, one can consider a task $\boldsymbol{\eta}$ being analogous to processing the complete video frame, a subtask η as processing a video slice/tile, and job as processing the video frame block within the slice/tile. Mathematically, a task i can be composed of n_i total subtask and given by:

The authors would like to point out that this work was carried out when all the authors were in Department of Computer Science, Karlsruhe Institute of Technology, Karlsruhe, Germany.

M.U.K. Khan et al., *Energy Efficient Embedded Video Processing Systems*,
DOI 10.1007/978-3-319-61455-7_4

$$\boldsymbol{\eta}_i = \left\{\eta_{i,0}, \eta_{i,1}, \cdots, \eta_{i,n_i-1}\right\} \tag{4.1}$$

An application can be designed with independent threads for each job, subtask, or a task. In case each job has its associated thread, there can be a large number of threads (a lot more than the number of compute nodes) in the system which will result in more context switches and communication/synchronization among the threads, leading to a large overhead. For a thread per subtask or task, the number of threads is reasonable and so is their associated overhead. In case of video applications, per slice/tile (i.e., subtask) thread results in higher video quality compared to a per block thread, because the dependencies among the video blocks are exploited which can result in higher compression. A per block thread may not let a block to maximally exploit dependencies among neighboring blocks.

Let us assume an application a is required to process all its subtasks (a complete task) within a deadline $t_{a,\max}$, given r_{tot} number of compute nodes. A compute node can process one or more subtasks. Suppose a core j, associated with a task i, takes $t_{i,j}$ amount of time to process its allocated subtask(s). Mathematically, the workload of a task (defined in time units) is given by:

$$t_i = \max_{\forall j \in \left\{0,\cdots,k_{a,\text{tot}}\right\}} \left\{t_{i,j}\right\} \tag{4.2}$$

Here, $k_{a,\text{tot}}$ is the number of nodes (e.g., cores) used for processing the subtasks. In order to maximally utilize the hardware resources and to increase the throughput-per-watt ratio, the time taken by each node to process its allocated subtask(s) should ideally be equal, i.e., all nodes start and end their allocated subtask(s) at the same time. This means that the workload distribution among the cores processing these threads must be balanced. Generally, reducing the number of subtasks (for an equivalent amount of jobs) worsens the output of workload balancing strategies due to lesser degree of freedom, but it decreases the management overhead. On the other hand, a large number of subtasks will result in better workload balancing [1] at the expense of (a) increased management, (b) communication overhead, and (c) reduced output video quality. Moreover, voltage-frequency levels of the associated compute nodes can also be scaled (DVFS) to achieve workload balancing among the cores.

An additional challenge to address is that the time taken by a subtask (and thus the time taken by a task) can vary at runtime, according to the scenarios discussed in the previous chapter. Moreover, the system must determine the correct number of resources allocated to an application a ($k_{a,\text{tot}} \leq r_{\text{tot}}$) which will result in high power efficiency while still meeting the application's deadline. That is, the following optimization problem needs to be solved:

$$\underset{k_{a,\text{tot}} \leq r_{\text{tot}}}{\text{argmin}} \left\{ \max_{\forall j \in \left\{0,\cdots,k_{a,tot}\right\}} \left\{t_{i,j}\right\} \leq t_{a,\max} \right\} \tag{4.3}$$

In this optimization goal, the number of nodes utilized for processing a task is reduced as much as possible. A large number of nodes will unnecessarily reduce the

Table 4.1 HEVC encoding characteristics for "Exit" (640 × 480) video sequence using [3, 4]

Encoder	*Enc-1*	*Enc-2*	*Enc-3*	*Enc-4*
Cores/threads	1	4	9	4
Power [W]	4.865	18.03	40.54	18.02
Time [msec]	659	178	78	125
Bytes	6806	7247	7262	7328
ΔPSNR [dB]	0	0.058	0.061	0.015

All cores have a constant frequency of 2 GHz during the execution. PSNR, peak signal to noise ratio. "Time" and "Bytes" denote time and bytes to encode one frame

time consumed (considerably below $t_{a,\max}$) in processing a task. However, it will also increase the resource and power consumption. This optimization problem suggests the number of nodes should be reduced such that they just meet the throughput requirement.

Video applications need to process data under tight constraints. For example, HEVC necessitates high compression with a throughput constraint [2], which generates numerous design and runtime challenges. As an example, Table 4.1 shows the result of utilizing different number of x86 cores in order to encode one frame of the "Exit" sequence under a 125 m sec. Ideally, all the values in Table 4.1 should be as low as possible. Encoder-1 with the least power consumption does not meet the throughput, whereas Encoder-3 needlessly increases the number of cores and hence the power. Encoder-2 might be able to sustain the workload if its frequency (and thus its power) is increased. Encoder-4 is a special case, where the application configuration parameters are intelligently tuned (i.e., selection of an operating point) to reduce the complexity. This results in power reduction, at the cost of reduced compression and quality (compare with Encoder-2). Thus, it is possible to determine a *compute configuration* (the number of nodes and their frequencies) and *application configuration* (operating point which allows tolerable quality degradation) that fulfills the throughput requirement while lowering the power consumption.

For ease of reading, we would mention a compute node as a programmable core in the following text, unless explicitly stated.

4.1.1 Power-Efficient Workload Balancing

For achieving balanced application execution on a multi-/many-core system, a technique to adaptively determine the system configuration is discussed, such that throughput constraints are met at low video quality degradation, while minimizing the system power consumption. By accounting for the resources of the system, the number of cores and their frequencies that must be used to manage the application's workload is determined. To estimate the frequency, the frequency estimation model is derived and adjusted at runtime, in order to (a) encounter the load variations of

the core and (b) make the technique portable. Afterwards, the application knobs can be optionally applied which results in further power savings. In summary, the discussed technique provides:

- *Selecting an appropriate compute configuration* that selects (a) the number of cores and (b) their frequencies (power), and (c) the number of subtasks (d) along with their maximum workload (output quality), depending upon the required throughput and the hardware characteristics.
- *Subtask-to-core mapping techniques* to fulfill the throughput requirement, pertaining to the structure in which task is divided in subtasks. This technique determines the optimal number of cores for supporting the application's workload. A bin-packing heuristic to assign subtasks to the available cores is employed, and hence, it is made sure that the utilization of a core is maximized, while minimizing the number of active cores.
- *Adaptive frequency estimation model*, which self-regulates by adjusting model coefficients or constants using runtime statistics, to determine the frequency of the cores and make the technique portable to other multi-/many-core systems.
- *Optional application configuration*, which tunes (curtails or enlarges) the workload of each subtask by tuning their configurations at runtime. It utilizes a feedback mechanism to maintain the output quality within tolerable limits. The frequencies of the cores are also adapted with reference to these subtasks. This results in reduced power consumption.

The outline of this technique is shown in Fig. 4.1. The compute configuration is an independent module, which gathers the throughput requirement and status signals from the application and directly controls the frequency of the cores. This module can be implemented as a separate library, interfaced with the kernel via system calls. This requires little effort from the application designer. Application configuration must tune application parameters, requiring the designer to modify the source. However, since the application designer best knows the application, therefore, application configuration can be implemented via moderate effort.

The pseudo-code of selecting the proposed compute and application configuration techniques is given in Algorithm 1, and the sketch of the video system

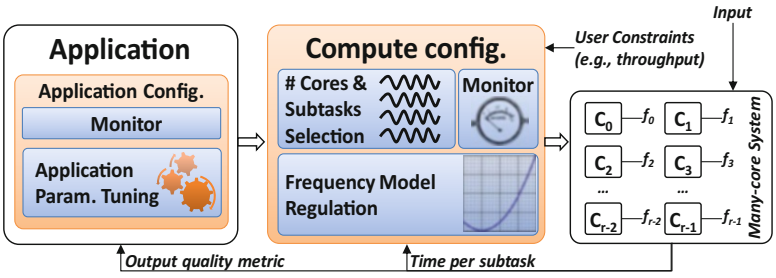


Fig. 4.1 Overview of the power-efficient workload balancing technique presented in this book on a multi- or many-core system

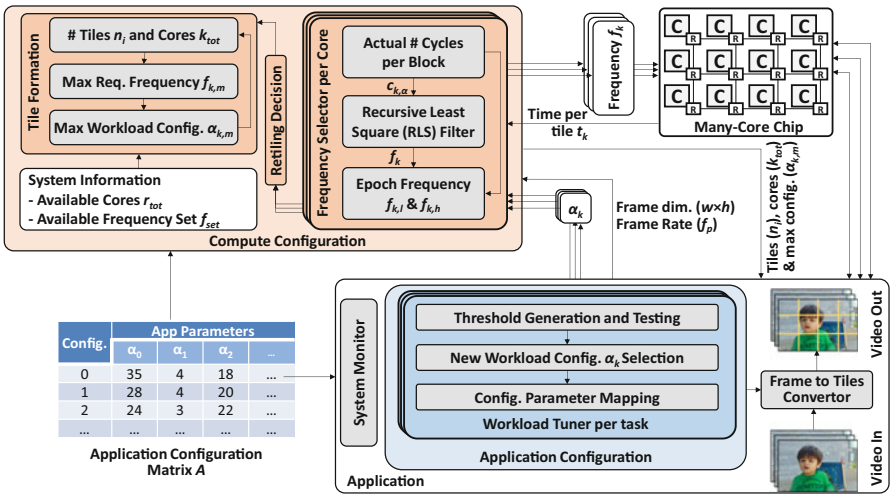


Fig. 4.2 Power-efficient workload balancing technique on a many-core system

employing these techniques is presented in Fig. 4.2. In the current text, the parallelization for a single application is under consideration; therefore, the subscript "a" for application is omitted for readability. Moreover, the above discussion is generally valid for all frame-based processing applications. However, for better understanding, the task is replaced by a video frame and the subtasks by tiles. The index k is used for representing a specific core.

In summary:

- At start, appropriate compute configuration is determined (line 6 of Algorithm 1) whereby the number of tiles/cores/threads is calculated ($k_{a,\text{tot}}$) to meet the deadline $t_{a,\max}$. The maximum required frequency to support the workload of each tile is also determined ($\boldsymbol{f_m}$, a vector having maximum frequencies $f_{k,m}$ of all cores). Further, if the number of cores is insufficient, the workload of application is self-curtailed (by determining application configuration $\boldsymbol{\alpha_m}$). Throughout the execution of the program on core k, the frequency of (f_k) and workload ($\boldsymbol{\alpha_k}$) cannot exceed $f_{k,m}$ and $\boldsymbol{\alpha_{k,m}}$, respectively.
- The optional local workload tuner per tile determines the tile's application configuration $\boldsymbol{\alpha_k}$ for the GOP or epoch (a set of video frames, line 9). If reducing workload has a tolerable impact on the output, the workload is reduced further and vice versa.
- Based upon $\boldsymbol{\alpha_k}$, the frequency of a core k is predetermined for a complete epoch (lines 10–11). Afterwards, the tile processing starts (line 12).
- Statistics are fed back to the frequency estimation model for adjusting the frequency of the next epoch (line 14). A Recursive Least Square (RLS) filter is used to adapt (or derive) the frequency estimation model constants.

The basic steps of the technique described above have the purpose of: (a) meeting the throughput demands by choosing the degree of parallelism (number

of tiles), (b) properly balancing the workload, and (c) reducing the power consumption by selecting suitable frequencies of the cores through exploiting tolerance to the output quality. In the coming text, the important modules given in Fig. 4.2 are discussed in more detail.

4.2 Compute Configuration

For parallel video processing, a multi-/many-core system with per-core DVFS is considered here. A video frame needs to be converted into tiles for parallel processing. The tile formation block is responsible for appropriate division of the video frame into tiles (with n_i total tiles) and also selecting the number of cores (k_{tot}) to process the tiles. Selecting number of tiles and cores depends upon the allowable frequency of a core (available in a discrete set of frequencies $\boldsymbol{f}_{\text{set}}$, from f_{min} to f_{max}), the total (or allowable) number of available cores (r_{tot}), and the required frame rate (f_p, i.e., user constraint). The tile formation technique determines three attributes of the system. First, the number of cores (k_{tot}) is calculated which will sustain application's workload according to Eq. (4.3). Second, the maximum frequencies of these cores ($f_{k,m}$) are determined for the allocated workload (discussed later). And third, the maximum allowable workload configuration ($\boldsymbol{\alpha}_{k,m}$) is determined (for best output quality) which can be sustained by the hardware platform to satisfy f_p. These three objectives can be quantified as a goal programming problem mathematically presented as:

$$\begin{aligned} &\{\min\{k_{\text{tot}}\},\ \min\{f_{k,m}\},\ \max\{\boldsymbol{\alpha}_{k,m}\}\} \\ &\text{s.t.}\ \ k_{\text{tot}} \leq r_{\text{tot}},\ \ t_k \leq 1/f_p \\ &\quad f_{\text{min}} \leq f_{k,m} \leq f_{\text{max}},\ f_{k,m} \in \boldsymbol{f}_{\text{set}} \\ &\quad \forall k = 0, \cdots, k_{\text{tot}} - 1 \end{aligned} \tag{4.4}$$

Here, the highest priority goal is to reduce the total number of computing cores. However, the priority can be reformed to minimize the frequency or maximize output quality.

4.2.1 Uniform Tiling

Here, each tile (subtask η) is associated with a single thread, and an individual core processes a single thread, i.e., number of tiles, threads, and cores are all equal. This load distribution and balancing technique is termed as uniform tiling because it tries to keep the number of tiles equal to the cores ($n_i = k_{\text{tot}}$) and tries to equally distribute n_{frm} jobs to every core. That is, the tile-based workload balancing attempts to equalize the number of blocks within every tile.

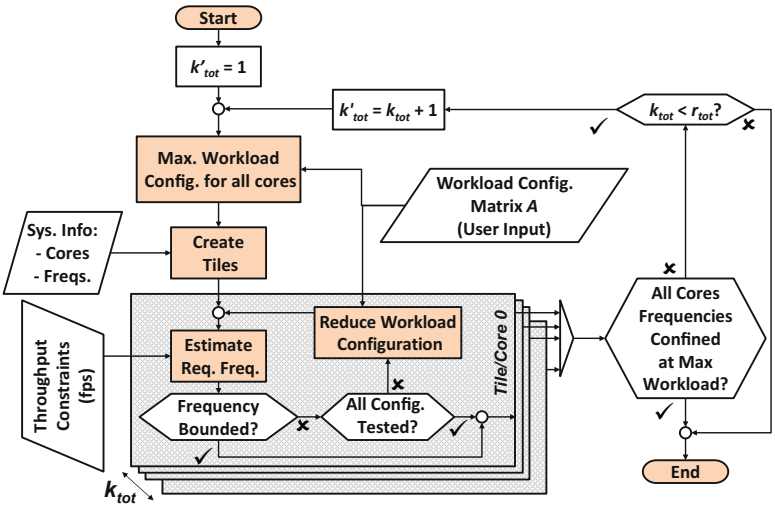


Fig. 4.3 Number of cores, frequency, and workload selection for uniform tiling

The program given in Eq. (4.4) is solved by the algorithm given in Algorithm 2. This algorithm is diagrammatically shown in Fig. 4.3. Primarily, this algorithm determines if a single tile/core/thread would potentially support the video processing workload (line 1 in Algorithm 2). This test is carried out using the maximum output quality (or maximum possible workload, denoted by $\boldsymbol{\alpha}_{\max} = \max(A)$, line 6) at a minimum possible core frequency, $f_{k,m}$ (lines 7–8) while processing all blocks in a frame, n_{frm}. Such an arrangement will result in using least amount of compute resources, best video quality, and minimal power consumption. If this configuration is not conceivable due to $f_{k,m}$ greater than the maximum supportable frequency ($f_{\max}$), the workload (and, hence, the output quality) is reduced (line 14) and the frequency is brought within $f_{\max}$ (lines 12–13). If there are other cores available, they are adaptively introduced (line 21). The algorithm repeats selection of the minimal frequency and maximum supportable workload for each individual core. The algorithm continues until either all the utilized cores can sustain the maximum workload (line 19) or there are no more cores available (line 15). Once the tile structure, frequencies, and maximum workloads of each core are determined, application execution begins.

Tile Structure Selection ($u_w \times u_h$) For enhanced video quality, the number of tiles (and cores) and their structure in terms of tiles in frame width × tiles in frame height ($u_w \times u_h$) are adjusted by reading a lookup table (line 3 in Algorithm 2), as shown in Table 4.2. The input number of tiles/cores (k'_{tot}) is used to give the actual number of tiles/cores (k_{tot}) by reading this lookup table. For example, if $k'_{\text{tot}} = 13$, a higher video quality is achieved by having $5 \times 3 = 12$ tiles instead of $1 \times 13 = 13$ tiles. Thus, in this case, $n_i = k_{\text{tot}} = 15$.

Table 4.2 Tile structure generation table, which can be changed for other applications

k'_{tot}	k_{tot}	$u_w \times u_h$	k'_{tot}	k_{tot}	$u_w \times u_h$	k'_{tot}	k_{tot}	$u_w \times u_h$
0	1	1×1	*6*	6	3×2	*12*	12	4×3
1	1	1×1	*7*	8	4×2	*13*	15	5×3
2	2	2×1	*8*	8	4×2	*14*	15	5×3
3	3	3×1	*9*	9	3×3	*15*	15	5×3
4	4	2×2	*10*	12	4×3	*16*	16	4×4
5	6	3×2	*11*	12	4×3	*17*	18	6×3

Here, "k'_{tot}" is the input; k_{tot} and $u_w \times u_h$ are the outputs

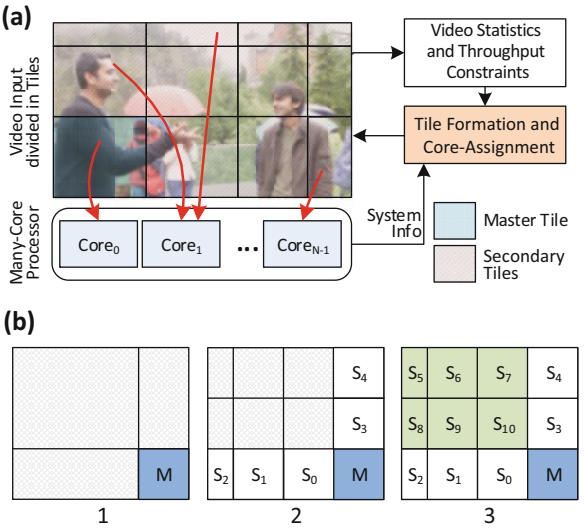


Fig. 4.4 (**a**) Non-uniform tiling and assignment to the cores, (**b**) master and secondary tile formation method

However, this kind of table is used as an example, and different video applications might benefit from a drastically different tile structure table. An application designer with the best knowledge about the application should be able to create such a table with ease.

4.2.2 Non-uniform Tiling

In addition to the uniform tiling, a non-uniform tiling technique can be used to reduce the number of cores to process the video application's workload. This situation is advantageous when systems with no DVFS capabilities are used to balance the workload of the cores. As the name shows, the video tiles in this technique are not uniformly structured. Basically, the number of tiles/subtasks $n_i \geq k_{\text{tot}}$ and the n_{frm} jobs are not equally distributed among subtasks (i.e., it might occur that size(η_i) $\neq$ size(η_j) with $i \neq j$). Figure 4.4a shows the conceptual

overview of the technique. Generating the tile structure for non-uniform tiling is a two-step process. In the first step the master tile is identified and the second step is determining the secondary tiles. Further, subject to the sustainable workload of a core, each tile might have different dimensions than the rest. An added benefit is that a table like Table 4.2 is not required.

For generating the video tile structure, following steps are taken (also outlined in Fig. 4.4b):

- *Identifying of the master tile* (size and location) depending upon the video-content properties. The selection of the master tile location and dimensions requires computing the variance of video blocks.
- *Forming the secondary tiles* (size and location), depending upon the size and location of the master tile. The secondary tile structure is generated by extrapolating the master tile throughout the video frame.
- *Determining number of cores* required for video processing and assigning the master and secondary tiles to the cores. A bin-packing heuristic is used to minimize the number of cores.

To identify the master tile, a system model that equates the sustainable throughput of a core in terms of the number of blocks and frame rate requirement is derived. Specifically, an equation can be derived via offline regression (or using online RLS filtering given in Sect. 4.2.5) that relates the time consumed in processing a certain number of blocks (n), frequency of the core (f), frame rate requirement (f_p), and other application parameters (some of which are discussed in Sect. 4.3):

$$t = g_1\left(n, f, f_p, \boldsymbol{\alpha}\right) \tag{4.5}$$

Here, $\boldsymbol{\alpha}$ represents application configurations, discussed in more detail in Sect. 4.3. However, if a timing constraint ($t_{\max}$) is given, then one can determine n at $f = f_{\max}$ using the relationship:

$$n = g_2\left(t_{\max}, f, f_p, \boldsymbol{\alpha}\right) \tag{4.6}$$

Hence, the number of blocks in the master tile (n) is determined using this equation. To find the dimensions of the master tile ($w_m \times h_m$), following formulas are used:

$$\begin{aligned} w_m(< w) &= b_w \times \left\lfloor \sqrt{n/a_r} \right\rfloor \\ h_m(< h) &= b_w \times \lfloor n \times b_w / w_m \rfloor \end{aligned} \tag{4.7}$$

In these equations, $w \times h$ is the resolution of the video frame, and the aspect ratio of the video frame is given by $a_r = w/h$. This construction ensures that the master tile dimensions cannot exceed the dimensions of the frame. Notice that h_m is generated from w_m and thus, $h_m \leq w_m$ if $a_r < 1$. However, it is possible to generate

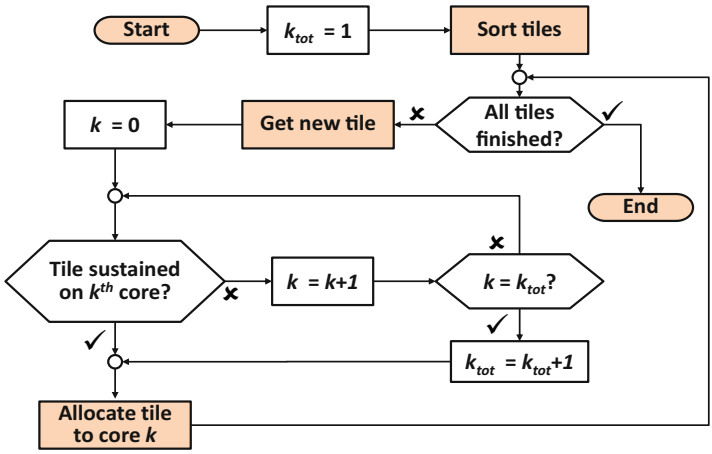


Fig. 4.5 Heuristic for tile-to-core assignment for non-uniform tiling

w_m from h_m. In this technique, the order depends upon the resolution of the frame. That is, if $w \geq h$, h_m is generated from w_m and vice versa.

The size and dimensions of the master tile are now available using the above formulation. The subsequent step is to find the location of the master tile. For this purpose, the four corners of the video frame are searched for blocks with highest variance. The corner with the highest accumulative variance is selected as the anchor point for the master tile (as given in the lower right corner of the frame in step 1 of Fig. 4.4b). Afterwards, the determination of secondary tiles is straightforward as seen by steps 2 and 3 of Fig. 4.4b.

Now, these tiles must be packed and dispatched to the individual processing cores. For this purpose, we can use a bin-packing heuristic given in Fig. 4.5. At first, the tiles are sorted in descending order according to the number of blocks n in the tiles. A tile is taken from the sorted list of the tiles, and iteratively, every core is tested whether the core can sustain the additional workload of this tile, along with the other tiles the core is already assigned. If yes, then the core will process the current tile in a time-multiplexed manner. Otherwise, a new core is introduced in to the computations. Once all the tiles are allotted to their respective cores, the frequencies of the cores are adjusted ($f \leq f_{\max}$) to (a) reduce the power consumption and (b) just fulfil the workload of the tiles. The frequency adjustment formulas will be discussed in the coming text.

4.2.2.1 Evaluation of Non-uniform Tiling

A demonstration of tile formation for a video frame of size 832 × 480 pixels using the uniform and non-uniform tiling, using different FPS, is shown in Fig. 4.6. For the non-uniform tiling technique, a master tile is randomly selected and marked for easy reference. Using the same techniques, Table 4.3 tabulates the number of cores and tiles used for HEVC processing, BD-Rate, and BD-PSNR [5] at different frame

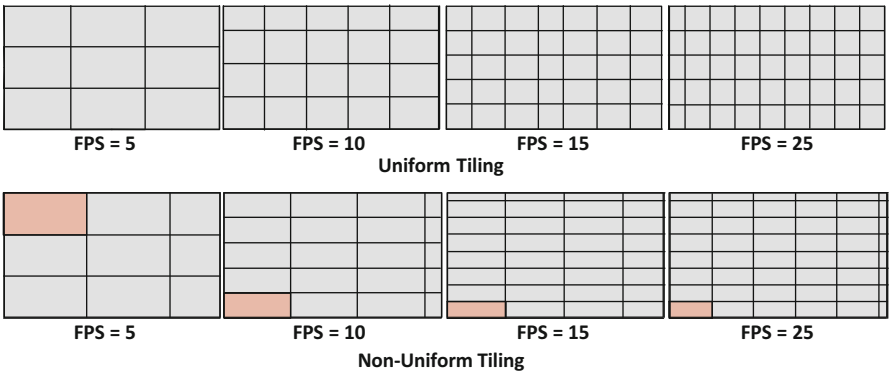


Fig. 4.6 Demonstration of video tile formation in HEVC, using uniform and non-uniform tiling techniques presented in this book

Table 4.3 Resource and quality analysis of uniform and non-uniform tiling for HEVC

	FPS	Cores	Tiles	BD-Rate	BD-PSNR
Uniform	*5*	9	9	1.4126	−0.0897
	10	20	20	−0.1197	0.0089
	15	35	35	1.1671	−0.0667
	25	45	45	1.8266	−0.1048
Non-uniform	*5*	9	9	1.5246	−0.0963
	10	17	20	0.0102	0.0014
	15	30	32	2.0505	−0.1182
	25	39	48	2.8470	−0.1645

rate requirements of the sequence "RaceHorses." Note that uniform tiling achieves better video quality. However, the cores required to sustain the throughput constraint (frame rate) are lesser for the non-uniform tiling technique.

4.2.3 *Frequency Estimation ($\mathbf{f_{k,m}}$)*

To estimate the frequency of a core ($f_{k,m}$ line 7 in Algorithm 2), the number of cycles consumed in processing a block ($\hat{c}_{k,\boldsymbol{\alpha}}$) is required (lines 4, 11). Determining $\hat{c}_{k,\boldsymbol{\alpha}}$ can be achieved via offline or online exploration. Mathematically, the number of cycles required per second on the core k, to encode tiles/subtasks having a total of n_k blocks, at a frame rate demand of f_p frames per second is given by:

$$f_{k,m} = n_k \times \widehat{c}_{k_{\text{tot}},\boldsymbol{\alpha}_m} \times f_p \tag{4.8}$$

This equation precisely represents the required number of cycles per second (Hertz) of the core to encode the assigned tiles. Hence, one can estimate the

frequency of a core (i.e., $f_{k,m}$) if the size of the tiles in blocks and the required frame rate are provided.

4.2.4 Maximum Workload Estimation ($\alpha_{k,m}$)

As seen in line 13 of Algorithm 2, the maximum workload a core k can sustain is denoted by configuration tuple ($\boldsymbol{\alpha}_{k,m}$). This denotes that once the tiling structure is defined and the processing starts, the application's workload must never exceed the workload defined by $\boldsymbol{\alpha}_{k,m}$. The application configuration is selected from a matrix A where:

$$A = [\cdots, \boldsymbol{\alpha}_{\min}, \cdots, \boldsymbol{\alpha}_{\max}, \cdots]^T \tag{4.9}$$

The tuple $\boldsymbol{\alpha}_{\max}$ has configurations for best output quality and hence maximum workload. The tuple $\boldsymbol{\alpha}_{\min}$ is for lowest quality and workload. Both $\boldsymbol{\alpha}_{\min}$ and $\boldsymbol{\alpha}_{\max}$ are selected by the user. For the matrix A, the tuple $\boldsymbol{\alpha} = (\alpha_0, \alpha_1, \ldots, \alpha_\psi)^T$ denotes the application parameters that can be configured and adapted at runtime. Figure 4.7 shows an example configuration matrix A.

4.2.5 Self-Regulated Frequency Model

For the video system presented here, the video processing can start after determining compute and application configurations (k_{tot}, $\boldsymbol{f_m}$ and $\boldsymbol{\alpha_m}$). We will refer to Fig. 4.8 in this section to explain the process in detail. The video system uses tile-based parallelism and each tile is processed in epochs. For each epoch of size z, an appropriate frequency ($f_k \leq f_{k,m}$) is selected and adapted at runtime. The details of selecting these attributes are given below.

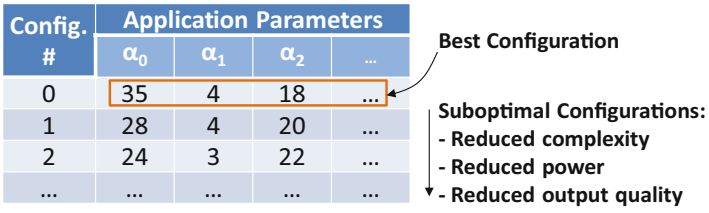


Fig. 4.7 Example workload configuration matrix A showing three workload configuration tuples

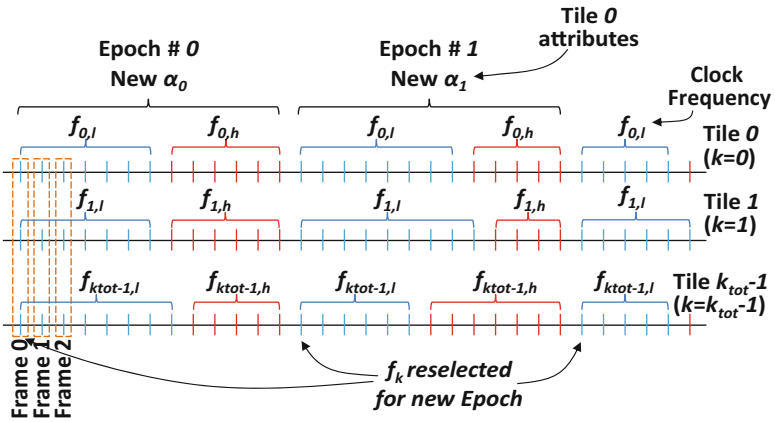


Fig. 4.8 Example frequency and workload allocation to collocated tiles using the presented technique

4.2.5.1 Frequency Estimation

The goal of the per-core frequency estimator is to adjust the clock frequency of the core, f_k, according to the workload assigned to the core, $\boldsymbol{\alpha_k}$, and the number of video frame blocks assigned to the core, n_k.

The frequency of a core is determined using Eq. (4.8) given above. However, one needs to estimate the total number of cycles for a given workload configuration, $\hat{c}_{k,\boldsymbol{\alpha}}$. The total number of cycles consumed for processing a video block or for a job can be estimated using system identification techniques. For example, one can use a model:

$$\widehat{c}_{k,\boldsymbol{\alpha}} = g(\boldsymbol{x}, \boldsymbol{\omega}) = \boldsymbol{x}^T \boldsymbol{\omega}_k \tag{4.10}$$

Here, $\boldsymbol{\omega}_k$ is the vector which holds the model constants and the vector $\boldsymbol{x}$ holds the configuration parameters. As an example, one can use the following configuration parameters:

$$\boldsymbol{x} = \left[\alpha_0, \cdots, \alpha_\psi, 1\right]^T \tag{4.11}$$

Note that $\boldsymbol{x}$ uses the configuration parameters from the tuple $\boldsymbol{\alpha}$ in matrix A. The value 1 at the end is used to encounter the error in the estimation model. The constant, which multiplies with 1, is also sometimes referred to as bias of the model. Equation (4.10) is both application and platform dependent. In derivation of this model, note that:

- High-complexity applications with timing constraints (like HEVC) are customized for specific platforms (e.g., tablets, smart phones). Since the application designer has the best knowledge of both application and platform, it is

straightforward for the designer to derive this model via regression analysis. However, this technique is least flexible compared to the next two techniques presented below.
- Dummy runs of the application in the setup phase can be used to derive this model.
- For maximum flexibility, the model can also be derived online, at runtime. Basically, runtime adaptation of this model is achieved by using the technique presented below.

4.2.5.2 Runtime Frequency Estimation Model Adjustment

The purpose of this technique is to either (a) accurately determine the model constant $\boldsymbol{\omega}_k$ or (b) fine-tune the model constants at runtime. The reason being $\boldsymbol{\omega}_k$ derived for one platform might not accurately estimate the frequency for other many-core systems if the system parameters (like processor type, cache size, external memory and output bandwidth, etc.) are different. Further, scheduling other applications on the same cores used for the video application will also fluctuate the total number of cycles consumed for processing. Moreover, some cores are physically located near the external memory controller and consume lesser cycles. It is also possible that no such model exists for estimating the number of consumed cycles, or it is too much of effort. In all such cases mentioned above, it becomes advantageous to adjust (or derive) the model constants at runtime for an accurate estimation of the frequency of the core used to process a tile. Therefore, scenarios where the system parameters are unknown or when the system load changes, adaptive model derivation or adjustment are of prime importance.

In this book, a technique based upon recursive least square (RLS) filter is used to adjust/derive $\boldsymbol{\omega}_k$ for each core at runtime, as shown in Fig. 4.9. After the end of an epoch, the RLS filter is used to regulate $\boldsymbol{\omega}_k$. Every core has an associated RLS filter. With each frame, in a feedback loop, the frequency adaptation technique receives

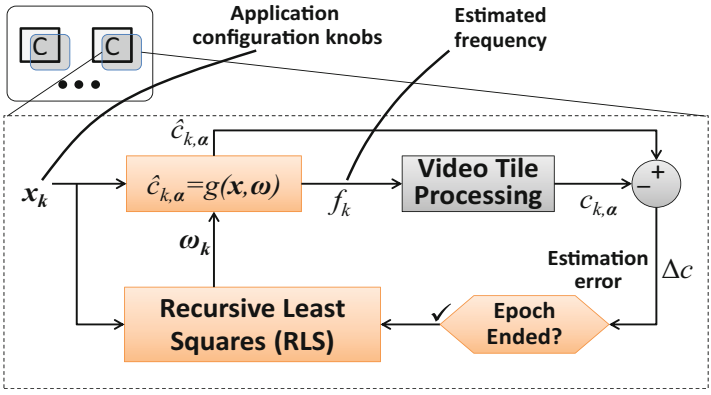


Fig. 4.9 Frequency estimation model for a core employing a RLS filter

the time consumed for processing the allocated tile(s). Let t_k present the average time for processing on the core k. Since the frequency of the core (f_k) used for processing the tiles is known, actual number of cycles consumed per block can be calculated ($c_{k,\boldsymbol{\alpha}} = f_k \times t_k/n_k$). RLS filter adjusts the model constants $\boldsymbol{\omega}_k$ at iteration m by using the formulas:

$$\begin{aligned}
&\widehat{c}_{k,\boldsymbol{\alpha}} = \boldsymbol{x}_m^T \boldsymbol{\omega}_{k,m-1} \\
&H = E_{m-1}\boldsymbol{x}_m \left(1 + \boldsymbol{x}_m^T E_{m-1}\boldsymbol{x}_m\right)^{-1} \\
&E_m = E_{m-1} - E_{m-1}\boldsymbol{x}_m \boldsymbol{x}_m^T E_{m-1}\left(1 + \boldsymbol{x}_m^T E_{m-1}\boldsymbol{x}_m\right)^{-1} \\
&\boldsymbol{\omega}_{k,m} = \boldsymbol{\omega}_{k,m-1} + H(c_{k,\boldsymbol{\alpha}} - \widehat{c}_{k,\boldsymbol{\alpha}})
\end{aligned} \tag{4.12}$$

In the above equations, H and E are matrices, and c is a scalar. With every iteration of RLS, the model constants $\boldsymbol{\omega}_k$ and the estimation-error covariance matrix E are updated. Note that in Eq. (4.12), the expression $(1 + \boldsymbol{x}_i^T E_{i-1}\boldsymbol{x}_i)$ is a scalar; hence, no matrix inversion is involved while computing $(1 + \boldsymbol{x}_i^T E_{i-1}\boldsymbol{x}_i)^{-1}$. The takeaway from this discussion is that RLS algorithm can determine and regulate the frequency model constants $\boldsymbol{\omega}$ and reduce the error in frequency estimation at runtime.

4.2.5.3 Core Frequency Allocation per Epoch

Note that the estimated f_k might not be supported by the hardware platform due to quantized frequency levels. Usually, f_k lies in an interval bounded by $f_{k,l}$ and $f_{k,h}$ (with $f_{k,l} \leq f_k \leq f_{k,h}$), where $f_{k,l}$ and $f_{k,h}$ are supported by the hardware platform. Therefore, we implement a frequency allocation heuristic (see Algorithm 3, diagrammatically shown in Fig. 4.8). For every core k, we solve the following integer linear program:

$$\begin{aligned}
&\max\{\psi\} \\
&\text{s.t. } f_{k,m} \geq \sum \left(\frac{\psi f_{k,l} + (z-\psi) f_{k,h}}{z}\right) \geq f_k \\
&\quad f_{k,l} \leq f_{k,h}, \ \ \psi \in N^+
\end{aligned} \tag{4.13}$$

For consecutive z frames to be processed, the collocated tile k of every frame is encoded either at $f_{k,l}$ or $f_{k,h}$. In each epoch, we try to increase the number of collocated tiles associated with a lower frequency ($f_{k,l}$ in this case, by maximizing ψ). The high frequency collocated tiles (associated with $f_{k,h}$) are reduced as much as possible, while keeping the average processing time under the timing constraint (determined by FPS).

4.2.6 Retiling

If the frequency of all the cores is stuck to $f_{\min}$ or $f_{\max}$ after a specific number of video frames (λ) are processed, then retiling is performed (see Fig. 4.2). Mathematically:

$$\mathrm{NT}(\,) = \begin{cases} \text{true} & \text{if } f_k \in \{f_{\min}, f_{\max}\}, \forall k = \{0, \cdots, k_{\text{tot}} - 1\} \\ \text{false} & \text{otherwise} \end{cases} \tag{4.14}$$

This is given in line 3 of Algorithm 1. It suggests that the estimation model of $\hat{c}_{k,\boldsymbol{\alpha}}$ (see Eq. (4.10)) used to initially govern the compute configuration ($k_{\text{tot}}, \boldsymbol{f_m}$ and $\boldsymbol{\alpha_m}$) no longer applies. This can occur because the estimation model for $\hat{c}_{k,\boldsymbol{\alpha}}$ is no longer applicable due to the above mentioned workload fluctuations. Hence, the tile structure is regenerated with the latest model of $\hat{c}_{k,\boldsymbol{\alpha}}$ adjusted by the runtime statistics of the video application (see Sect. 4.2.5.2). One can take $\lambda = 5z$, where z is the number of frames in an epoch. Basically, $5z$ number of frames insure that a considerable number of times the estimation model of $\hat{c}_{k,\boldsymbol{\alpha}}$ is adjusted (according to Eq. (4.12)).

4.3 Application Configuration

If required, the designer can exploit the user's tolerance to output video quality for gaining further power efficiency. In such situations, any workload reduction technique can be added, and the frequency of the cores can be reduced because, now, the throughput requirements can be met even at a lower frequency under lower complexity. In short, the workload fluctuation and user's tolerance collectively translate to a workload-driven frequency/power adaptation while satisfying the throughput requirement. The per subtask/tile workload tuner shown in Fig. 4.2 is accountable for adjusting the workload. Workload tuning is achieved by picking a configuration tuple from A (see Fig. 4.7). The application configuration is updated after every epoch. In addition, since the collocated tiles are highly correlated, the adjusted configuration is applied to the tiles in the next epoch. Additionally, since application configuration is application specific, we will use HEVC as a case study, and the concept can be generalized to other applications.

4.3.1 HEVC Application Configurations

A tuple $\boldsymbol{\alpha}_k$ contains HEVC encoder settings for the tile k. As an example, the following parameters can be used for adapting the HEVC workload at runtime:

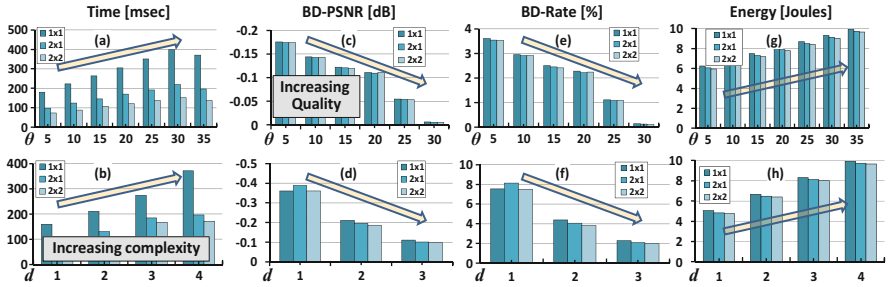


Fig. 4.10 For "Keiba" sequence (832 × 480), impact of θ and d along with trend lines with different tile settings on (**a**, **b**) time per frame, (**c**, **d**) BD-PSNR, (**e**, **f**) BD-Rate [5], and (**g**, **h**) energy. The anchor/reference encoding is done with one tile, maximum θ and d

$$\boldsymbol{\alpha} = [\alpha_0, \alpha_1, \alpha_2, \alpha_3]^T = [\theta, d, QP, n_{\text{frm}}/k_{\text{tot}}]^T \tag{4.15}$$

Using these settings, we can rewrite $\boldsymbol{x}$ in Eq. (4.11) as:

$$\boldsymbol{x} = [\theta, d, QP, n_{\text{frm}}/k_{\text{tot}}, 1]^T \tag{4.16}$$

Here, $\theta \in \{1,\ldots,35\}$ presents the total number of HEVC intra-predictions that are performed per PU (see [6, 7] for details). However, the important aspect is that increasing θ will increase the probability of generating a prediction that results in the best compression efficiency. Obviously, this comes at the cost of additional workload and energy consumption. This situation is depicted in Fig. 4.10a, c, e, g.

The HEVC parameter $d \in \{1,2,3,4\}$ presents the allowable depth of PU subdivision (see details in Chap. 2). Fig. 4.10b, d, f, h plots the impact of d on the timesavings and bitrate. Notice that by reducing d, the bitrate increases, and the energy and workload decreases, resulting in reduced compression efficiency. Compared to reduction in θ, although the reduction of d causes a higher energy saving, it also incurs a higher quality loss (compare Fig. 4.10c–f). Therefore, it is rational to have θ as the first choice of parameter tuning. Using the above mentioned definitions of θ and d (and keeping the visual quality constant, by keeping quantization parameter QP constant), one way of writing the configuration matrix A is:

$$A = \begin{bmatrix} 1 & 2 & \cdots & 35 & 1 & \cdots & 35 \\ 1 & 1 & \cdots & 1 & 2 & \cdots & 4 \end{bmatrix}^T \tag{4.17}$$

Further, simulations point that the model to estimate the frequency is highly dependent upon the parameter n_{frm}/k_{tot} and this term is also included in computation of the RLS update Eq. (4.12). The reason to include k_{tot} as the denominator is that the time required to process a video frame reduces in an inverse relation with the number of tiles (see Chap. 2). Hence, $\boldsymbol{\omega}_k$ and $\boldsymbol{x}$ are 5 × 1 vectors, and E is a 5 × 5 matrix due to the five workload tuning parameters (including θ and d) chosen here. A similar technique can be used for other video applications.

4.3.2 HEVC Configuration Tuning

As discussed earlier, the application configuration parameters (θ, d) are calculated at the start of an epoch, and they remain fixed for all the collocated tiles in the epoch as shown in Fig. 4.8. If these parameters decrease the workload, the frequency f_k can be reduced, which will result in lower power consumption of the system.

However, there must be a feedback mechanism to determine the configuration parameters for the next epoch. The feedback control monitors the output of the system and then tunes the knobs of the system to reach the desired output. Inspired by this, one way of workload tuning at runtime is to monitor the output of the encoder and then take informed decisions. One variable to monitor can be the number of compressed bytes (b_k) generated after encoding video tiles associated with the core k. For example, if b_k increases, this means that the output quality degrades and vice versa. Increasing b_k points to a worsening encoding efficiency. If b_k increases beyond a certain threshold ($\tau_{b,k}$), the workload $\boldsymbol{\alpha}_k$ should increase to balance the degrading encoding quality (see Fig. 4.10). One way to define this threshold adaptively is:

$$\tau_{b,k} = \mu_k(b_k) + \sqrt{\sigma_k^2(b_k)} \tag{4.18}$$

Here, $\mu_k(b_k)$ is the average and ${\sigma_k}^2(b_k)$ is the variance of b_k, for all collocated tiles in epoch. Note that using Knuth's formula [8], one can update the mean and variance iteratively with every frame.

For each block within a tile, if threshold in Eq. (4.18) is satisfied, θ is adjusted as:

$$\begin{aligned} \theta_k &= \lfloor \mu_k(\theta) + \gamma_k(b_k - \tau_{b,k}) \rfloor_{\theta_{k,m}} \\ \gamma(h) &= \begin{cases} +\psi/2 & \text{if } h > 0 \\ -\psi & \text{if } h \leq 0 \end{cases} \end{aligned} \tag{4.19}$$

In the equation above, ψ is a user-defined parameter. The concept behind this technique is that the workload is reduced continuously (i.e., we reduce the number of predictions tested θ) until there is a surge in the output bitrate owing to a bad prediction. A larger ψ means that θ is more sensitive to variation in b_k and vice versa. Further, the reduction in number of intra angular predictions tested (i.e., θ) is stricter compared to increasing θ to sustain a higher video quality. Note that while employing Eq. (4.19), $\theta_{k,m}$ (derived in Algorithm 2, $\boldsymbol{\alpha}_{k,m}$) also saturates θ_k as it is the maximum number of predictions tested for the tiles associated with core k. Further, it is possible that impact of reducing θ_k to increase the output bytes is not high and Eq. (4.18) is always satisfied. If θ_k reaches a minimum possible value ($\theta_{\min} = 5$, see Fig. 4.10), the workload is reduced further by shifting d to the next lower level. If the output bytes increase and $\theta_k = \theta_{k,m}$, the next higher d_k ($d_k \leq d_{k,m}$) is selected. Note that this technique is equivalent to picking the next workload configuration tuple from the matrix A.

If a certain number of frames have been processed or b_k exceeds another threshold (denoted by $\tau_{b,k,m}$), we may reset $(\theta_k,d_k) = (\theta_{k,m},d_{k,m})$ for the current epoch. One way of mathematically describing this condition is:

$$b_k > (1+\varepsilon)\tau_{b,k,m} \tag{4.20}$$

In this equation, $\tau_{b,k,m}$ equals b_k of the most recent tile with $(\theta_k,d_k) = (\theta_{k,m},d_{k,m})$. The reason for resetting the configuration parameters to their maximum quality configuration is that scene changes or sudden appearance of high texture scenes might worsen the prediction quality. For example, a sudden scene change will cause the ME quality to drop. Similarly, highly irregular texture scenes might make fast intra predictions ineffective in capturing the texture flow. In such cases, there will be sudden increase in b_k, and the above condition will be met. Hence, the best possible compression configuration should be applied to achieve highest video quality for the given throughput constraints.

Referring again to the above equation, ε is a user-defined tolerance metric for b_k increase. Higher tolerance will result in overall reduced workload and vice versa. For example, if b_k of current tile is greater than $\tau_{k,m}$ by 5% ($\varepsilon = 0.05$), the respective tile in the new epoch is encoded with maximum workload configuration and frequency (and maximum power).

4.3.3 HEVC Parameter Mapping

In the previous section, it is shown that the decrease in workload is achieved by curtailing $\boldsymbol{\alpha}$ (i.e., reducing θ_k and d_k). However, one must sensibly include the best possible selection candidate for both θ_k and d_k while curtailing the workload by exploiting HEVC-specific properties. In this way, the degradation of output quality will be smaller (or none), and the workload of the application can be further curtailed to increase throughput-per-watt.

4.3.3.1 Intra Mode Estimation

The individual pixels in a PU can provide a hint to the possible texture flow direction if their gradients (generated via Sobel edge filtering kernel) are accumulated. This texture flow direction can be used to estimate the vicinity of the best intra angular prediction. It is noteworthy that computing gradients is a time-consuming operation, and not all pixels correctly suggest the actual direction of intra prediction mode that is finally selected after a brute-force search. Only the pixels that lie on the boundary of an edge may rightly hint the final prediction selected by the intra encoder (see discussion in Chap. 3). Thus, to increase the speed of computations, it is advisable to avoid performing gradient extractions on pixels

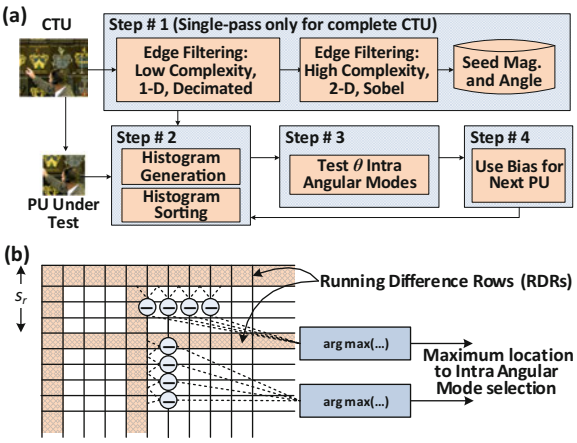


Fig. 4.11 (**a**) Fast intra angular mode estimation, (**b**) seed pixel extraction process for $s_r = 4$ using [7]

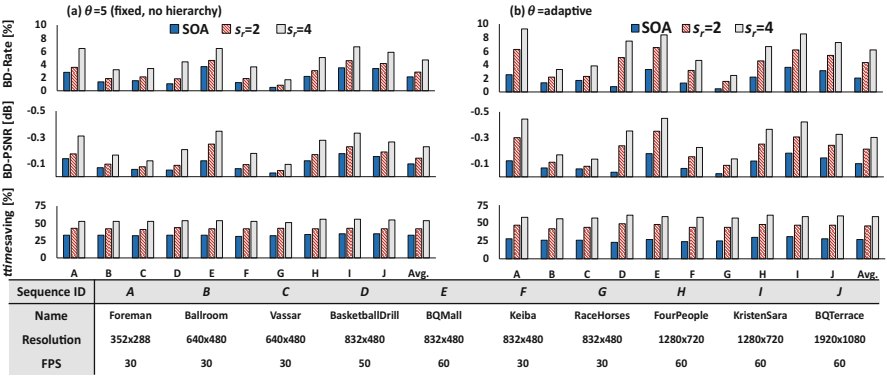
that will not contribute positively towards estimating the final intra angular prediction with high precision.

To estimate the final intra angular prediction with high accuracy and to reduce the time complexity, a technique is shown in Fig. 4.11a. As seen, a complete CTU is preprocessed before being provided to the HEVC intra encoder. Firstly, seed pixels and seed intra angular modes are extracted from the CTU (shown in Fig. 4.11b) and stored in a memory. Subsequently, only the contents of this memory are used to estimate the most probable intra angular prediction that will be selected by the brute-force search process. Moreover, two complexity knobs (s_r and θ) are also available for the user using which a user can leverage the computational complexity with the quality of the estimate.

To avoid complexity overhead due to the Sobel operation, only two pixels with largest running difference on the boundary of the block $s_r \times s_r$ are used for generating gradient magnitudes and direction. Afterwards, a histogram of gradient angles is created using gradient magnitudes as weights. This histogram is sorted and the first θ angles within the sorted list are used for intra angular estimation. The value θ can be selected as given in Eq. (4.19).

Evaluation of Fast Prediction Estimation Technique The evaluation of the early prediction mode estimation techniques for HEVC intra encoding is carried out using our in-house ces265 video encoder [10] (for more information, see Appendix B). These evaluations are conducted on a computer, with 2.7 GHz Dual-Core Intel CPU and 8 GB RAM. To avoid any outliers and other influencers, only a single thread of ces265 is used for evaluations.

The resulting video quality comparison in terms of BD-Rate and BD-PSNR and timesaving are given in Fig. 4.12. In Fig. 4.12a, θ is fixed for both the presented and state-of-the-art [11] techniques, whereas in Fig. 4.12b, adaptivity of both techniques is enabled. Note that the technique presented in this book with $s_r = 4$ provides the best timesaving but also results in the largest video quality loss. Using $s_r = 2$ delivers a good balance between the timesaving of the presented



Sequence ID	A	B	C	D	E	F	G	H	I	J
Name	Foreman	Ballroom	Vassar	BasketballDrill	BQMall	Keiba	RaceHorses	FourPeople	KristenSara	BQTerrace
Resolution	352x288	640x480	640x480	832x480	832x480	832x480	832x480	1280x720	1280x720	1920x1080
FPS	30	30	30	50	60	30	30	60	60	60

Fig. 4.12 Comparison of video quality (BD-Rate and BD-PSNR [5]) and timesaving for the presented mode estimation technique with state-of-the-art (SOA) [11] technique, using (**a**) fixed and (**b**) adaptive number of angular modes

technique compared to [11]. With a fixed $\theta = 5$, the additional timesaving of this technique compared to [11] including the overhead of the technique is 18% and 32% for $s_r = 2$ and $s_r = 4$, respectively. For adaptive θ, the additional timesaving compared to [11] is 27% and 44%, respectively. The formula to calculate the timesaving is:

$$\text{timesaving } [\%] = (t_{\text{baseline}} - t)/t_{\text{baseline}} \times 100 \tag{4.21}$$

In this equation, t_{baseline} is the time spent by the encoder without any complexity reduction technique. Similarly, to compute BD-Rate and BD-PSNR, the PSNR of the video sequence is required. In this case, PSNR for a particular QP is computed via:

$$\text{PSNR}_{QP} = \frac{4 \times \text{PSNR}_Y + \text{PSNR}_{Cb} + \text{PSNR}_{Cr}}{6} \tag{4.22}$$

4.3.3.2 PU Depth and Size Selection

From the discussions in the previous chapters, we notice that HEVC tests every PU size in order to determine the PU structure, which results in the best compression. The application configuration discussed above, however, limits that a PU should only be tested for a selected number of depths (denoted by d). Unfortunately, using PU sizes, which are far off from the actual PU size selected by brute-force processing, dents the compression performance of the encoder. For example, if allowable $d = 1$, then the encoder has only a single pass to determine the PU structure of the complete CTU. For example, in this case, the encoder may only use a single PU of the same size as the CTU. For CTUs encompassing high variance

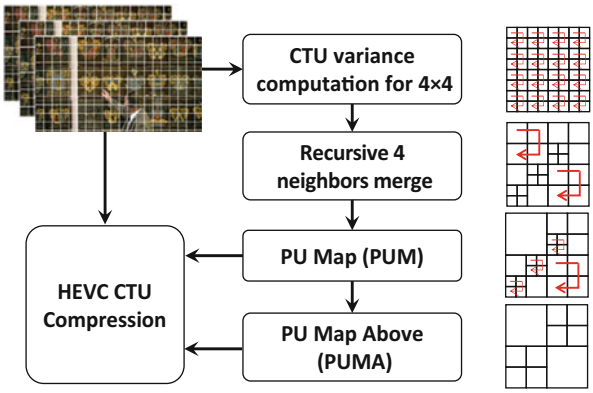


Fig. 4.13 PU structure determination technique by using the variance of the subblocks

regions, this can result in compression loss. Another example could be to select regular PU sizes (say 8×8) for the complete CTU. Once again, overhead due to PU headers and other auxiliary information will reduce the compression efficiency. In short, none of the simpler techniques will result in high compression efficiency. Therefore, an adaptive technique is required to predetermine the PU structure of the CTU while accounting for the video content (texture).

The PU structure selection technique presented here works according the flowchart given in Fig. 4.13 and Algorithm 4. In the first stage, the complete CTU is divided into subblocks of size 4×4, and variance of each subblock is calculated, which is a scalar. Afterwards, the PU structure is generated iteratively, by jointly considering the variance of the neighboring blocks. That is, the neighboring four subblocks are combined/joined into a larger block if they fulfill all of the following conditions:

- All four subblocks have identical size.
- Their corners join at a single point.
- All four subblocks have a variance less than a threshold (discussed below).
- The variance of the combined subblock is also less than the threshold.

Once the above technique is used for subblocks of size 4×4 for the complete CTU, the algorithm repeats with subblock sizes of 8×8, until finally it cannot combine any four neighboring blocks. This PU structure is called as PU map (PUM).

If the required depth (d) for processing is set to 1, then PUM is used for encoding, and all other possible PU structures are discarded. If the depth is set to 2, then the adjacent/neighboring four subblocks of equal sizes are directly combined without checking the variance thresholds to form a PUM Above (PUMA), and only PUM and PUMA are used for picking the best PU size while encoding the CTU. If it is not possible to combine any subblock from PUM to form PUMA, then the PUMA consists of all 4×4 PUs.

An important feature of the above algorithm is to find the variance (v) of the combined subblock. This will incur additional overhead if we revisit the value of every sample in the combine block. However, this overhead can be reduced by using Chan's formula [12]:

$$(\mu, v) = \left(\frac{\mu_1 + \mu_2}{2}, \frac{2(n-2)(v_1 + v_2) + (\mu_1 - \mu_2)^2 n}{4(n-1)} \right) \tag{4.23}$$

This equation relates the resultant mean and variance (μ, v) of a combined block of n entries, from two blocks, with mean and variances as (μ_1, v_1) and (μ_1, v_1), respectively, having $n/2$ entries. The computational burden can be further reduced by simplifying Chan's formula:

$$(\mu, v) \approx \left(\frac{\mu_1 + \mu_2}{2}, \frac{2(v_1 + v_2) + (\mu_1 - \mu_2)^2}{4} \right) \tag{4.24}$$

This formula is derived by approximating $(n-1)$ and $(n-2)$ as n (for $n \in \{32, 64, 128\ldots\}$).

Evaluation The results reported in Fig. 4.14 show the impact of the complexity reduction technique presented in this section for HEVC intra encoder. Figure 4.14a, b provides the BD-Rate and BD-PSNR comparison of this technique against the

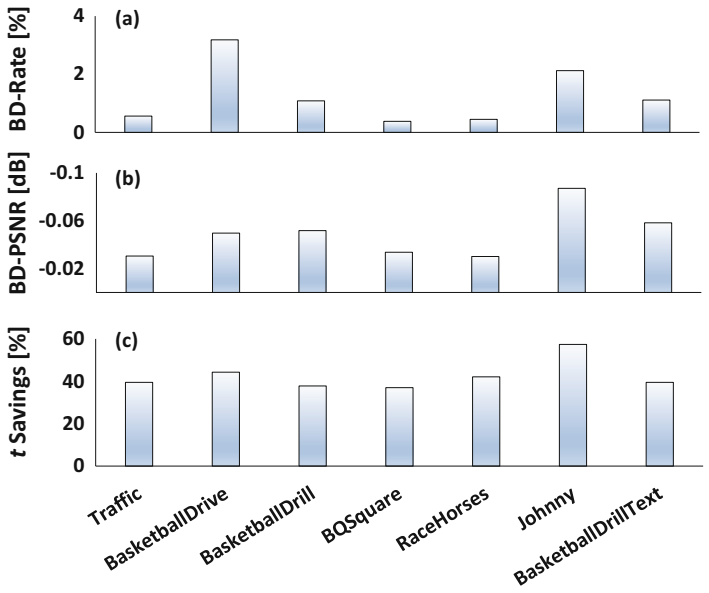


Fig. 4.14 (**a**) BD-Rate, (**b**) BD-PSNR, and (**c**) timesaving for the presented PU structure estimation technique

RDO (or full search) for different sequences. On average, the technique presented here incurs a BD-PSNR loss of −0.048 dB, which is insignificant compared the state-of-the-art technique (−0.1184 dB) [13], which also uses two levels of PU size selection.

The percentage timesaving of this technique for different video sequences is presented in Fig. 4.14c. The time complexity is compared with the reference implementation. Note that up to 57% timesaving is achieved, for sequences with diverse texture and motion properties.

4.4 Workload Balancing on Heterogeneous Systems

In previous sections, the focus was mainly on distributing the processing to the compute nodes. However, it is not stated how the nodes can be prioritized for distribution of subtasks. Logically, a compute node with high power efficiency must be allocated a larger quota of subtasks compared to a node with lower efficiency. For example, a hardware accelerator might be much more efficient than a soft-core; therefore, the hardware accelerator should be utilized maximally to increase the throughput-per-watt.

Continuing with our previous discussion, this section will focus on workload distribution and balancing on heterogeneous nodes. These nodes will be working together in synergy to handle the application's throughput demands. Here, we will revert to use the term compute node instead of the core. Once again, the reader can draw an analogy among tasks (video frames), subtasks (tiles), and jobs (image blocks). The reason to use these terms is that the technique presented here is not limited to video applications.

4.4.1 System Model

Consider an application, which should compute its multiple allocated tasks (also termed as the application's load), that is scheduled to run on a heterogeneous system with multiple nodes. The application consists of multiple independent tasks. A task i can be subdivided into n_i subtasks, and the subtasks can be executed in parallel. A node must process its assigned subtasks within a deadline ($t_{i,\max}$).

The heterogeneous system consists of r_{tot} total nodes. The technique which we will present here selects only k_{tot} nodes ($k_{\text{tot}} \leq r_{\text{tot}}$) for sustaining the throughput of the application. It will also determine their voltage and frequencies (f_k for node k), which determines the power consumed by these nodes (p_k). Note that the frequency of the node is a function of the number of subtasks allocated to the node ($n_{i,k}$), the throughput requirement, and the cycles consumed to process a job on node k ($c_{i,k}$), as will be discussed later.

The compute nodes can either be soft-cores, accelerators, GPUs, etc. No assumption is made about the sustainable throughput and power consumed by these nodes. That is, each of these nodes can sustain a different throughput compared to others and can consume different power for equivalent throughput. Furthermore, a node has either a program written in software layer to process the assigned jobs or has a custom hardware unit implemented in the hardware layer to process the given job.

The target of the technique presented here is to minimize the total power consumed by the heterogeneous system (p_{tot}) while meeting a certain throughput constraint. Mathematically, the optimization goal is:

$$\begin{aligned} &\min\left(p_{tot} = \sum_{k=0}^{r_{\text{tot}-1}} p_k(f_k)\right) \\ &\text{s.t.} \\ &f_k \in \{(0) \cup \psi | (f_{\min} \leq \psi \leq f_{\max})\} \quad \forall k \in \{0, \cdots, r_{\text{tot}}\} \\ &\qquad t_{i,k} \leq t_{i,\max} \qquad \forall k \in \{0, \cdots, k_{\text{tot}}\} \\ &\sum_{k=0}^{k_{\text{tot}-1}} n_{i,k} = n_i \end{aligned} \tag{4.25}$$

This optimization problem advises that a node can either be power gated (with $f_k = 0$) or its frequency range can be between permissible limits. However, each parallel computing node must finish the tasks within the deadline $t_{i,\max}$, while the cumulative number of subtasks processed by individual nodes should equal the total number of subtasks.

To solve the load distribution and balancing problem given above, it is essential to address some issues. First, determining k_{tot} (the actual number of nodes used for processing) is unknown beforehand, and it is used as a constant in the optimization problem. Secondly, the frequency of a node is permissible to have values within two distinct ranges (i.e., either it can be 0 or between $f_{\min}$ and $f_{\max}$), which will create problems for the optimization algorithm. Moreover, the variables used in Eq. (4.25) must be derived in terms of tunable system parameters for optimization, which might be cumbersome. Owing to these issues, a heuristic can be efficiently utilized, as explained below.

4.4.2 *Load Balancing Algorithm*

The resource allocation and load balancing technique is shown in Fig. 4.15 and Algorithm 5. In summary, the load balancing technique on heterogeneous systems is a two-step process:

- Collecting the compute and power profiles of all nodes and computing an "efficiency index" of each node.

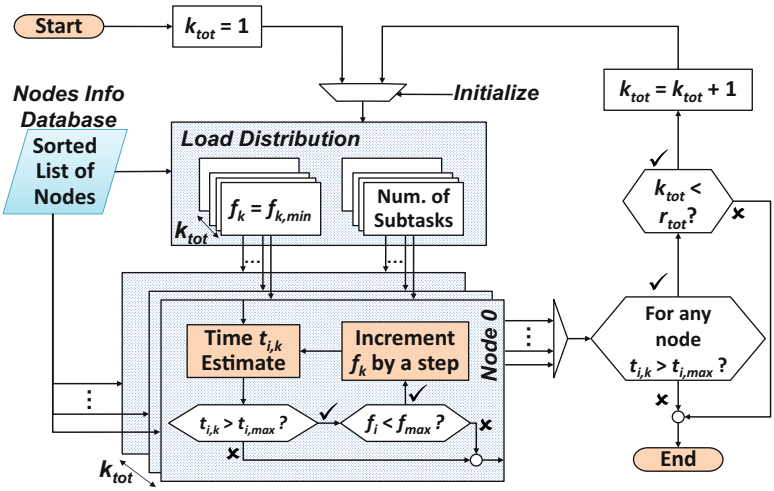


Fig. 4.15 Workload distribution and balancing technique on heterogeneous nodes using efficiency index of the nodes

- Actual load distribution technique, which allocates the subtasks among the nodes, depending upon the metrics computed in the first step. Further, the compute and power profiles can be additionally used for selecting the appropriate clock frequency of the nodes.

For the first step, the power profiles (a function to map frequency of the node to the power consumed by the ndoe, $\boldsymbol{p}(\boldsymbol{f})$) of all the nodes are collected. This can be done using online power measurement (e.g., using Intel Power Gadget and reading MSR registers) or by exploring the datasheets of the nodes. Afterwards, the average power over the available frequency range or operation modes ($\boldsymbol{p}_{\boldsymbol{k},\mu}$) is computed. Moreover, the maximum average power for all nodes ($p_{\mu,\max}$) is determined as:

$$p_{\mu,\max} = \max_{\forall r_{\text{tot}}\ \text{nodes}} \left\{ \boldsymbol{p}_{\boldsymbol{k},\boldsymbol{\mu}} \right\} \tag{4.26}$$

In this equation, for simplicity, it is presumed that the power profile of the nodes is independent of the type of the task. To encounter this, task-based average power calculation can be employed. Further, the time taken by a task is defined as the maximum time taken by any node to process its assigned subtasks. In addition, this algorithm collects the cycles consumed to process a subtask associated with task i on a node k (given by $c_{i,k}$) for all the nodes. For this, an expression can be derived using either offline regression analysis or online dummy runs of the application during setup as discussed in the previous section. Similarly, the maximum cycles consumed by a node to process a subtask of task i ($c_{i,\max}$) are also determined. Using

these metrics, the *efficiency index* for a node k (ϕ_k) can be computed using various system metrics. In the following, some example formulations of ϕ_k are given:

Formulation 1 $\phi_{k,1}$ The efficiency index of a node k is the ratio of the maximum number of cycles and power consumed to process a task i, for all the nodes, to the cycles and power consumed by the node k.

$$\phi_{k,1} = \frac{c_{i,\max} p_{\mu,\max}}{c_{i,k} p_{k,\mu}} \tag{4.27}$$

Formulation 2 $\phi_{k,2}$ The efficiency index of a node k is the ratio of the maximum cycles spent in processing a task i for all the nodes to the cycles consumed by the node k.

$$\phi_{k,2} = \frac{c_{i,\max}}{c_{i,k}} \tag{4.28}$$

Formulation 3 $\phi_{k,3}$ The efficiency index of a node k is the ratio of the maximum average power for all the nodes to the average power of the node k.

$$\phi_{k,3} = \frac{p_{\mu,\max}}{p_{k,\mu}} \tag{4.29}$$

Formulation 1 indicates that the node, which consumes the least cycles and power to process a job, has higher computation efficiency (throughput-per-watt). Therefore, it must be preferred for use if a higher throughput-per-watt metric is essential. Similar argumentation can be made for formulation 2 and 3. Other formulations of the efficiency index can account for other metrics, e.g., network hops among the nodes, external memory access latency of the node, the amount of on-chip memory that a node possesses etc.

After compute statistics for all the nodes are gathered, the actual load distribution algorithm commences as shown in Fig. 4.15 and Algorithm 5. Nodes are adaptively introduced to share the load of the application, and their frequencies are tuned such that the system consumes minimum amount of power.

At start of the algorithm, the nodes are ordered in a list (g_ϕ) with descending efficiency indices (Algorithm 5, line 1). Every time a new node needs to be introduced for distribution of the subtasks, the next node from this list is considered (line 19). The processing starts with only one node (having the highest efficiency index $\max(\phi)\ \forall\ k_{\text{tot}}$) with its minimum frequency ($f_k = f_{k,\min}$, line 5). That is, the complete load is allocated to this node ($n_{i,k} = n_{\text{tot}}$). Afterwards, the time spent by a node to process the workload is given by the formula:

$$t_{i,k} = \frac{c_{i,k} n_{i,k}}{f_{i,k}} \tag{4.30}$$

In case the timing constraints are not satisfied ($t_{i,k} > t_{i,\max}$), the frequency of this node is increased by a single frequency step until $f_{i,k} = f_{k,\max}$. If $f_{i,k} = f_{k,\max}$ and the timing constraints are still not met, the frequency cannot be increased further, and another node is introduced ($k_{\text{tot}} = k_{\text{tot}} + 1$). The next node is fetched from the efficiency index array $\boldsymbol{g}_{\boldsymbol{\phi}}$. The algorithm starts again with minimum power configuration ($f_{i,k} = f_{k,\min}$) for all k_{tot} nodes. The load is distributed to a node k using the following relation:

$$n_{i,k} = \frac{n_i \phi_k}{\sum_{j=0}^{k_{tot}-1} \phi_j} \tag{4.31}$$

The basic idea of load distribution is that the node with the highest efficiency index gets the most load. Once again, the time that will be consumed by a node while processing $n_{i,k}$ subtasks can be estimated using Eq. (4.30). In case a node is incapable to process the allocated number of subtasks within $t_{i,max}$, its frequency is increased. If any node is at its maximum power capacity and the system still cannot meet the throughput requirement, another computation node is introduced and the process is repeated. If all the nodes meet their deadlines, the algorithm stops and the compute configuration, which can sustain the application's load, is determined.

4.5 Resource Budgeting for Mixed Multithreaded Workloads

One can exchange the goals and the constraints in Eq. (4.4), and the goal of the optimization changes to maximize the throughput of the system or an application, under a given power constraint (e.g., TDP, power wall). This optimization construct is particularly valuable for batch processing, high processing computing and offline video processing. If power constraints of the system are not specified, the techniques mentioned in Sects. 4.2 and 4.4 can be extended by running each core at maximum frequency.

In such cases, we must determine the appropriate compute configuration for the given power budget (p_{tot}). We can easily extend the compute configuration selection goal programming problem to account for power consumption. However, this resource distribution becomes challenging if multiple tasks with associated subtasks and jobs are running in parallel. This is a more realistic and challenging scenario, which is considered in this section.

Every independent task can represented as a separate application, and therefore, the application index a and task index i can be used interchangeably in the coming

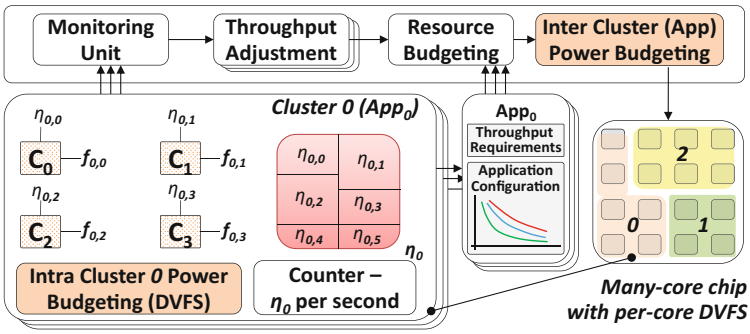


Fig. 4.16 Resource budgeting for mixed multithreaded workloads

text, i.e., $a = i$. Note that in this section, the resources of a task i are given by the number of cores k_i and power p_i associated with the task. For video applications, this resource budgeting scenario is applicable to multicasting scenarios (see Sect. 2.2.2), whereby every encoder is an isolated application trying to encode its video frames (i.e., tasks) and each video frame can be divided into tiles (i.e., subtasks). Each tile is made of multiple image blocks (jobs).

This section presents a multi-granularity resource budgeting technique for multithreaded workloads at different levels (i.e., among tasks and within tasks). The complexity of the resource distribution problem is reduced by a hierarchical budgeting method, which also provides performance isolation and fairness among applications running these tasks. The resource distribution tries to meet the throughput requirement of all tasks. Summarizing, the problem addressed by the technique presented here is: "how to budget resources and TDP among different tasks/applications, such that the throughput of concurrently executing, multithreaded applications is maximized?" Since each multithreaded application will require multiple cores for executing its workload, we denote the set of cores allocated to a multithreaded application as a "cluster." The technique discussed in this section (outlined in Figs. 4.16 and 4.17) executes the following key operations:

- *Selecting Number of Cores (i.e., Cluster Size):* For a given task, the number of required cores is estimated that fulfills its throughput constraints based upon the task's characteristics, performance constraint, and underlying resources.
- *Cluster-Level Power Budgeting:* For a given power budget of the whole chip and considering the unpredictability in power demand of different tasks, a global cluster-level power allocator distributes the power to the clusters at a coarse granularity. The budget allocation depends upon cluster's characteristics and the feedback error in power budgeting.
- *Intra-cluster Power Budgeting:* An intra-cluster, local power allocator distributes the cluster's power budget to the individual cores of the cluster at a finer granularity. This power distribution depends upon the workload of the specific core. The operating frequencies of the cores are adjusted to satisfy the upper limit on the power consumption of the cluster.

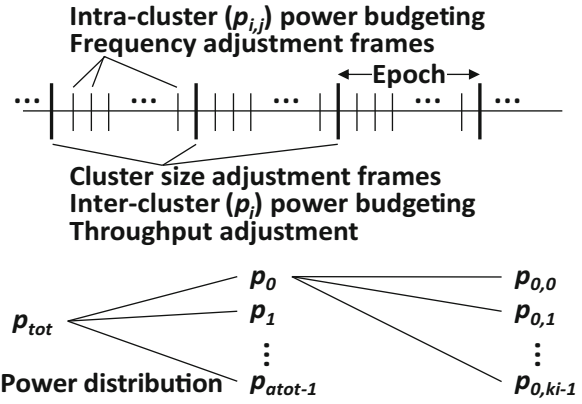


Fig. 4.17 Resource budgeting timeline

Summarizing, this technique is a two-step process. First, it determines the cluster size and power of all the applications. Afterwards, the power is divided among the constituent cores of the cluster. For illustration of this technique, a homogenous many-core chip with per-core DVFS and uniform tiling is considered. The global intercluster resource and power budgeting take into account (a) the performance history of the application, (b) the throughput requirement of the application, and (c) the operating mode of the application. Further, this distribution is tuned at runtime after every epoch (e.g., a set of tasks like GOP or after a specific time) based upon the tasks' requirements and the assigned budgets. Once the inter-cluster resources and powers have been budgeted, the intra-cluster budgeting commences and decides the best frequency ($f_{i,j}$) of each core under cluster power budget cap. Each cluster is accountable for processing a task i ($\boldsymbol{\eta}_i$) within the time $t_{i,\max}$. Each tasks consists of multiple subtasks ($\eta_{i,j}$). Each core in the cluster is responsible for processing a subtask. A task is deemed finished if all cores within the cluster finish processing their respective subtasks. The technical challenge is to define the right number of cores and their frequencies within a cluster, in order to fairly maximize the throughput requirements of all the tasks ($\boldsymbol{\eta}_i$ processing time $< t_{i,\max}$) under the power budget (p_{tot}) of the complete chip.

4.5.1 Hierarchical Resource Budgeting

The detailed operational flow of the hierarchical resource budgeting technique is illustrated in Fig. 4.18. This technique distributes power among a_{tot} concurrently executing multithreaded tasks/applications, competing for r_{tot} total cores and p_{tot} total power budget. In this discussion, multiple tasks form a single epoch. After each epoch, the inter-cluster resource and power budgeting are triggered to adjust resource distribution to every cluster. The intra-cluster power budgeting is executed to adjust the power of each core after every completed task ($\boldsymbol{\eta}_i$).

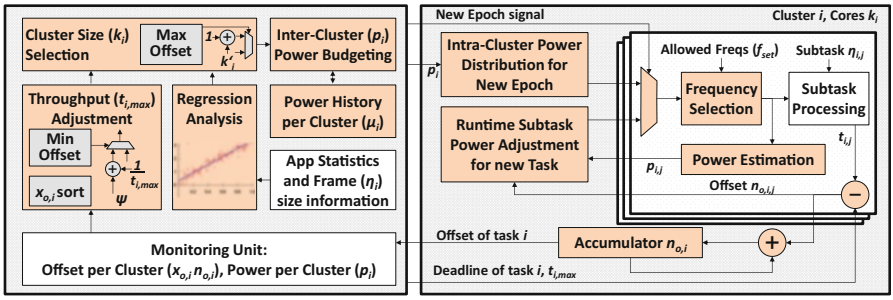


Fig. 4.18 Details of the multi-granularity power budgeting technique. For simplicity, only one application is shown

A task i is accompanied with a user- or pre-defined $t_{i,\max}$ seconds for processing $\boldsymbol{\eta}_i$ with n_i total jobs. This corresponds to each core in the cluster i trying not to take more than $t_{i,\max}$ seconds to process its allocated subtask ($\eta_{i,j}$). The number of jobs within a subtask j of task i is denoted by $n_{i,j}$. The optimization problem solved in this technique is thus:

$$\begin{aligned}
&\max\{\min_{\forall i \in \text{tasks}}\{1/(t_i)\}\} \\
&s.t. \\
&t_i = \max\{t_{i,j}\} \leq t_{i,\max} \quad \forall \eta_{i,j} \in \boldsymbol{\eta}_i, \forall \text{ tasks} \\
&\textstyle\sum_{\forall \text{ tasks}} k_i \leq r_{tot}, \quad \sum_{\forall \text{ tasks}} p_i \leq p_{tot} \\
&f_{i,j} \in \{f_{\min}, \cdots, f_{\max}\} \ \forall \text{ cores}
\end{aligned} \tag{4.32}$$

The basic processing steps for allocation of compute resources and power budget (given in Algorithm 6) with reference to the above optimization problem are as follows:

1. Determining the size of a cluster i (the number of cores, k_i) that is allocated to a particular multithreaded application, in order to satisfy the quality of service constraint (line 2). The cluster size is constrained by the available cores.
2. Performing cluster-level power budgeting such that the total power of the system does not exceed the TDP budget (line 3). Therefore, it is possible that not all enabled cores run at maximum frequency-voltage setting.
3. Fine-grained (intra-cluster) power budgeting among the cores within a cluster (lines 4–5), by only considering discrete set of frequencies of the cores.
4. Adjusting the power/frequency of individual cores within the cluster, depending upon the input data and power consumption (lines 10–12). The voltage of the cores is also scaled accordingly.

Note that OS or a specialized core within a cluster can perform the overall management (selecting cluster sizes, power distribution, etc.) after a control period. In the following, each key factor of this technique will be explained with reference

to Fig. 4.18 and Algorithm 6. For ease of explanation, discussion will start from intra-cluster power distribution and adjustment (steps 3–4) and end with the selection of appropriate cluster size (step 1).

4.5.2 *Intra-Cluster Power Budgeting* $\mathbf{p_{i,j}}$

Assume that the number of cores within the cluster i (k_i) and the power (p_i) provided to the cluster (which process the task $\boldsymbol{\eta}_i$) are known. p_i is distributed among the constituent cores of the cluster. Each core handles a single subtask j ($\eta_{i,j}$), i.e., $\boldsymbol{\eta}_i$ is divided into k_i subtasks. However, it is possible that the workload of each subtask differs from the rest. For example, in video applications, one of the video tile might have high motion content that will result in larger workload compared to the rest (see Fig. 3.5). This subtask can then become a critical path and will hurt the throughput of the application. Therefore, the intra-cluster power budgeting technique presented here is based upon workload of the subtasks, by budgeting more power to critical subtasks.

For the first iteration of intra-cluster power budget distribution (Algorithm 6, lines 4–5), the jth core of the cluster i is allocated power $p_{i,j}$ using the equations:

$$\begin{aligned} p_{i,j,\text{Alloc}} &= \left(p_i + p_{i,d}\right) \times n_{i,j}/n_i \\ p_{i,j} &= \begin{cases} p_{i,j,\text{Alloc}} \text{ if } & p_{i,j,\text{Alloc}} < p_{\max} \\ p_{\max} & \text{otherwise} \end{cases} \\ p_{i,d} &= p_{i,j,\text{Alloc}} - p_{i,j} \end{aligned} \tag{4.33}$$

These equations point out that each core is allocated power depending upon the number of jobs within its respective subtask. Given the power of a core, frequency of the core ($f_{i,j}$) can be determined using techniques similar to [14, 15]. The relationship between power and frequency of a core can be approximated with a linear equation ($\Delta p = \psi \Delta f$, where ψ is a design time parameter). However, since the frequency set ($\boldsymbol{f}_{\mathbf{set}}$) is quantized and a core cannot run at a power larger than $p_{\max}$, therefore, frequency out of only a specific frequency set can be selected for the core. At this particular frequency, the core j will consume power, $p_{i,j}$. Therefore, the difference ($p_{i,d}$) between the allocated ($p_{i,j,\text{Alloc}}$) and consumed ($p_{i,j}$) powers is added while allocating power to the next core in the cluster. Note that it is possible that $p_{i,j} < p_{i,j,\text{Alloc}}$ because the power consumed by a core j might be lesser than the actual power allocated to the core.

Once the power of each core and a particular frequency connected with it is determined, the application can start (Algorithm 6, lines 9–11). Due to varying workload of each $\eta_{i,j}$, it is conceivable that the time consumed by a subtask ($t_{i,j}$) differs from the rest and also varies at runtime. For every subtask, a variable $n_{o,i,j}$ is used to determine the offset (or misprediction) of the power/frequency of each core:

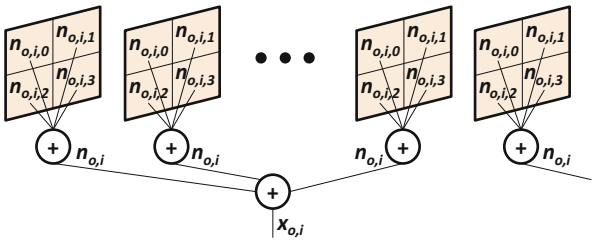


Fig. 4.19 Concept of offset per subtask, task, and epoch. An example four-core cluster is shown, where each core processes a single subtask and generates subtasks offset $n_{o,\,i,\,j}$

$$n_{o,i,j} = t_{i,j} - t_{i,\max} \tag{4.34}$$

A large positive value of $n_{o,i,j}$ denotes more power should have been allocated to the core and vice versa. This is because for the subtask j running on a core, a large amount of time $t_{i,j}$ is taken compared to the deadline $t_{i,\max}$, and therefore, the core running the subtask j should be allocated more power. Further explanation about $n_{o,i,j}$ can be found in Fig. 4.19.

Frequency Adaptation In this step, power is taken from a core with a smaller workload and given to the core with a larger workload. Power exchange among the cores of the cluster occurs, if the new allocated power of the cores does not violate the cluster power budget.

If at runtime, the workload characteristics of subtasks are changing or the system load varies due to parallel running applications, it becomes essential to exchange power among cores within a cluster. After a task is processed, $n_{o,i,j}$ for all threads of the application i is collected. The cores with larger $n_{o,i,j}$ should have their frequencies elevated. However, since a power quota is allocated to the application, a priority-based algorithm increases or reduces the power of the cores. This process is outlined in Algorithm 7. At first, the subtasks are sorted by their $n_{o,i,j}$ magnitude (Algorithm 7, line 1). The algorithm iteratively checks if there are some subtasks with $n_{o,i,j} > 0$. This subtask is termed as $\mathrm{T_{high}}$ and this is the critical subtask of the application, as it consumes the most time. In case such subtasks exist, the remaining subtasks are searched having $n_{o,i,j}$ lesser than the critical subtask and are called $\mathrm{T_{low}}$. The algorithm tries to take power from $\mathrm{T_{low}}$ and give that power to $\mathrm{T_{high}}$. This causes an increase in the frequency of $\mathrm{T_{high}}$ while reducing the frequency of $\mathrm{T_{low}}$ by a single frequency step. If after this exchange, the power consumption of the cluster is lower than the allocated power to the cluster, the change in frequencies is accepted, and the algorithm searches for next subtask pairs to adapt. Note that this algorithm will only continue if $n_{o,i,j}$ of at least one of the remaining subtasks is greater than 0 and greater than $n_{o,i,j}$ of at least one other subtask.

4.5.3 *Inter-Cluster Power Budgeting* $\mathbf{p_i}$

If the cluster size (k_i) is known, the TDP system (p_{tot}) can be distributed among the a_{tot} concurrently executing tasks with every epoch (Algorithm 6, line 3). Note that

TDP is less than the sum of maximum power consumption of all the cores, thus defining the amount of dark silicon. The intercluster power budgeting is a two-step process. In the first step, the power budgeted to a cluster i (p_i) after an epoch is given by the following relation.

$$p_i = p_{\text{tot}} \times \frac{(k_i + \mu(p_i))}{\left(\sum_{j=0}^{a_{\text{tot}}-1} k_j + \sum_{j=0}^{a_{\text{tot}}-1} \mu(p_j)\right)} \tag{4.35}$$

In the above equation, k_i-dependent terms try to budget more power to the cluster with larger number of cores. However, it is possible that the power of the cluster diverges under workload demands. Thus, another factor $\mu(p_i)$ is used to account for the task i's power consumption history in the previous epochs (e.g., previous five epochs). In this equation, $\mu(p_i)$ denotes the expected (average) power of the i^{th} task for the previous epochs and acts as a feedback variable to the inter-cluster power budgeter. Therefore, a task with high power consumption history is allotted more power. For the first epoch, average power of all tasks is zero, and hence every core of the system gets an equal amount of power. Further, note that the summation of powers of all clusters will result in power $\leq p_{\text{tot}}$ at any time instance. Note that this inter-cluster power budgeting is different than a control-based power adjustment technique discussed in [14], which requires a feedback loop and a tuning parameter.

Power Adjustment For the next epoch, the technique presented here tries to budget more power to a cluster processing a high-complexity application. This is the second step of power budgeting process, where the actual power allocated to a cluster (p_i) is based upon the offset information from all the tasks ($n_{o,i}$, see Fig. 4.19). For each task i, $n_{o,i}$ is computed by:

$$n_{o,i} = \sum_{j=0}^{k_i-1} n_{o,i,j} \tag{4.36}$$

For a complete epoch, $n_{o,i}$ of each task is accumulated in $x_{o,i}$ (see Fig. 4.19 for reference). If there is a task $a_{\max}$ with nonnegative, maximum $x_{o,i}$ among all tasks, a part of the power from the task $a_{\min}$ with minimum $x_{o,i}$, is taken and given to this task. Mathematically, this process is defined as:

$$\Delta p = p_{a\min}/\psi_1, \;\; p_{a\max} = p_{a\max} + \Delta p, \;\; p_{a\min} = p_{a\min} - \Delta p \tag{4.37}$$

In this equation, ψ_1 is a user-defined parameter or a design constant and regulates the amount of power shifted from the low-complexity task to the high-complexity task.

Throughput Adjustment If the maximum $x_{o,i}$ among all tasks is negative, it means that all tasks are meeting their processing deadlines. Since TDP budget can be fully utilized, more power can be pumped into the system to gain higher throughput. Therefore, the algorithm tries to escalate the throughput requirements

of the applications internally, such that they demand more resources. In this technique, $t_{i,\max}$ for the next epoch is decreased using:

$$t_{i,\max} = \frac{t_{i,\max}}{\psi_2 t_{i,\max} + 1} \tag{4.38}$$

The user-defined factor or a design constant ψ_2 denotes the amount of change in $t_{i,\max}$. A higher value of ψ_2 will reduce $t_{i,\max}$ quicker. If the goal is only to meet the deadline, then one can ignore this procedure.

4.5.4 Selection of Cluster Size

Note that unlike [14, 16, 15], which only target to increase the average instructions executed per second of all single-threaded applications, the technique presented here introduces a timing constraint (deadline, $t_{i,\max}$). The resource distribution technique tries to meet this deadline for every task. This deadline can be specified for the complete run of the application or for processing individual tasks of the application. The task-based deadline is a more realistic concept for streaming and image, video, and audio processing applications.

For selecting k_i, the technique presented here uses the maximum allowable time ($t_{i,\max}$) of the task i for processing a single task $\boldsymbol{\eta}_i$ (Algorithm 6, line 2). To keep the processing time below $t_{i,\max}$, the available cores (r_{tot}) that can be distributed among different tasks are considered. All cores run at some frequency f at startup. Using this frequency, it is possible to estimate the total number of cores required for processing a subtask within $t_{i,\max}$:

$$k_i^{'} \geq g(t_{i,\max}, \boldsymbol{\eta}_i, f) \tag{4.39}$$

This equation relates $t_{i,\max}$, the characteristics of $\boldsymbol{\eta}_i$, and the minimum number of expected cores (k'_i) that can sustain the workload of the application i. This equation can be generated using offline statistics or regression analysis [17, 18]. This equation can also be derived using the technique outlined in Eq. (4.10). For example, using a core i7 at 3.4 GHz frequency (fixed), the time consumed for encoding a video frame/task via HEVC, with n_{frm}CTUs/jobs and k_i tiles/subtasks, is given by:

$$t_{i,3.4G} = -1.844 + 0.01827 n_{\text{frm}} + 0.5441 n_{\text{frm}}/k_i \tag{4.40}$$

However, note that this equation can be scaled and used to approximate the time consumed for any other frequency f via:

$$t_{i,f} = \left(\frac{3.4 \times 10^9}{f}\right) t_{i,3.4G} \tag{4.41}$$

Typically, for performance-constrained applications, the number of cores required for the given deadline can be estimated at the setup stage (offline). Afterwards, this number can be updated at runtime depending upon the task properties (see Sect. 4.2.5). Since this technique adapts $t_{i,max}$ (see Eq. (4.38)), therefore, k'_i also changes at runtime.

As established earlier, the workload characteristics of the tasks might change at runtime. Therefore, the technique presented here also adapts the size of the cluster after every epoch. As a concrete example of cluster size adaptation, k'_i is incremented by one for a task i with $x_{o,i} > 0$. Hence, this technique tries to allocate more resources to the task requiring more computations.

The number of cores that is actually budgeted to the task i is derived using the following formula:

$$k_i = \lfloor k_i^{'} \times r_{\text{tot}} / \sum_{j=0}^{a_{\text{tot}-1}} k_j^{'} \rfloor \tag{4.42}$$

Afterwards:

$$k_i = \begin{cases} k'_i & , k_i > k'_i \\ k_i & , \text{otherwise} \end{cases} \tag{4.43}$$

Due to the increment operation, even if summation of k'_i is greater than r_{tot}, summation of k_i will still be in bounds ($\leq r_{\text{tot}}$) due to Eq. (4.42). This also suggests that if the joint requirements of all the concurrently running tasks exceed the global power budget and resources, the system will continue to run at a degraded quality (with time per task $> t_{i,\max}$) under TDP budget. Further, Eq. (4.43) shows that not all of the r_{tot} bright cores might be utilized at a single time.

References

1. Cesar, E., Moreno, A., Sorribes, J., & Luque, E. (2006). Modeling Master/Worker applications for automatic performance tuning. *Parallel Computing, 32*(7), 568–589.
2. Lian, C. J., Chien, S. Y., Lin, C. P., Tseng, P. C., & Chen, L. G. (2007). Power-aware multimedia: concepts and design perspectives. *IEEE Circuits and Systems Magazine*, 26–34.
3. Carlson, T., Heirman, W., & Eeckhout, L. (2011). Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation. In *High Performance Computing, Networking, Storage and Analysis*.
4. Li, S., Ahn, J. H., Strong, R., Brockman, J., Tullsen, D., & Jouppi, N. (2009). McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Microarchitecture*.
5. Bjontegaard, G. (2001). Calculation of average PSNR differences between RD-curves. VCEG Contribution VCEG-M33.

6. Sullivan, G. J., Ohm, J., Han, W., & Wiegand, T. (2012). Overview of high efficiency video coding. *IEEE Transactions on Circuits and Systems for Video Technology, 22*(12), 1649–1668.
7. Khan, M. U. K., Shafique, M., & Henkel, J. (2014). Fast hierarchical intra angular mode selection for high efficiency video coding. In *International Conference on Image Processing*.
8. Knuth, D. E. (1998). The art of computer programming, Addison-Wesley, p. 232.
9. Huck, S. (2011). Measuring processor power, TDP vs. ACP. Intel.
10. ces265: Open-source C++ based multithreaded HEVC software. Chair for Embedded Systems. [Online]. Available: http://ces.itec.kit.edu/ces265/. Accessed 06 Aug 2014.
11. Jiang, W., Mal, H., & Chen, Y. (2012). Gradient based fast mode decision algorithm for intra prediction in HEVC. In *International Conference on Consumer Electronics, Communications and Networks*.
12. Chan, T. F., Golub, G. H., & LeVequ, R. J. (1979). *Updating formulae and a pairwise algorithm for computing sample variances*. Stanford: Stanford University.
13. Sun, H., Zhou, D., & Goto, S. (2012). A low-complexity HEVC Intra prediction algorithm based on level and mode filtering,. In *International Conference on Multimedia and Expo (ICME)*.
14. Ma, K., Li, X., Chen, M., & Wang, X. (2011). Scalable power control for many-core architectures running multi-threaded applications. In *Internation Symposium on Computer Architecture*.
15. Sharifi, A., Mishra, A., Srikantaiah, S., Kandemir, M., & Das, C. R. (2012). PEPON: Performance-aware hierarchical power budgeting for NoC based multicores. In *Parallel Architectures and Compilation Techniques*.
16. Mishra, A., Srikantaiah, S., Kandemir, M., & Das, C. R. (2010). CPM in CMPs: Coordinated power management in chip-multiprocessors. In *International Conference on High Performance Computing, Networking, Storage and Analysis*.
17. Khan, M. U. K., Shafique, M., & Henkel, J. (2014). Software architecture of high efficiency video coding for many-core systems with power-efficient workload balancing. In *Design, Automation and Test in Europe*.
18. Shafique, M., Zatt, B., Walter, F. L., Bampi, S., & Henkel, J. (2012). Adaptive power management of on-chip video memory for multiview video coding. In *Design Automation Conference*.

Chapter 5
Energy-Efficient Hardware Design for Video Systems

The techniques based in the software layer for computation- and power-efficient video processing system given in Chap. 4 do not necessitate any custom hardware. However, custom hardware architectures for video processing systems are in wide use because they produce higher throughput and have higher complexity and power reduction potential compared to the software-only solutions. This chapter outlines some of the hardware architectural enhancements and custom accelerators for highly efficient video processing systems. Efficient I/O and internode communications for video processing system are discussed. Hardware architectures of the complete system and accelerators are also given, specifically pertaining to H.264/AVC and HEVC encoders. Furthermore, the hardware accelerator allocation or workload administration (whereby the accelerator provides its services to multiple nodes) is also discussed, which can be useful in shared hardware accelerator paradigms. Targeting the memory subsystem, power-efficient hybrid memory architectures and SRAM aging mitigation are also presented.

5.1 Custom Video Processing Architectures

The techniques outlined in Chap. 4 can be employed on a given multi-/many-core system with little or no alterations. However, if a custom platform is designed, with embedded "soft-cores" (cores that run software for control/orchestration), and hardware coprocessors or accelerators used for number-crunching, an architectural support is required to achieve efficient communication mechanisms among the computing nodes. A communication technique must be adequate to fulfill the

The authors would like to point out that this work was carried out when all the authors were in Department of Computer Science, Karlsruhe Institute of Technology, Karlsruhe, Germany.

M.U.K. Khan et al., *Energy Efficient Embedded Video Processing Systems*,
DOI 10.1007/978-3-319-61455-7_5

computation requirements, and at the same time it must not reduce the performance of the system.

In case of heterogeneous many-core systems, with in-core, tightly or loosely coupled hardware accelerators, soft-cores with external memory, and I/O ports, the data input and output of the system can become challenging. Specifically, for video encoders, the data from the camera can be analog, not in standard-compliant format (e.g., instead of YCbCr 4:2:0 required by the HEVC encoder, the camera provides RGB), and may require clipping, chroma resampling, reordering etc. To address these issues, this section will present a study of video memory, and an architecture to support reading/writing video samples to and from the video processing system is discussed. Afterwards, the discussion about data communication is generalized and extended to multi-/many-core heterogeneous systems.

5.1.1 Memory Analysis and Video Input

As seen from our discussion in Chap. 2, blocks of video need to be fetched from the external memory or camera and fed to the video processing system. This communication must consider the bandwidth between the video generating source and the video processing system.

For better understanding, we present an example data communication scenario and discuss the requirements of the communication subsystem. For example, a 16-bit DDR3 on Altera's DK DEV 2AGX260N FPGA development kit, running at 300 MHz, can theoretically support a maximum bandwidth of $16 \times 300\text{M} \times 2 = 8.9$ Gbps of data transmission between the DDR3 memory and the FPGA. However, this bandwidth reduces to 6.23 Gbps by taking a conservative memory efficiency of 70% for a typical DRAM [1]. From the discussion around Eq. (2.3), a system processing full-HD frames (1920×1080 pixels) would require ~0.58 Gbps to read the video frame at frame rate of $f_p = 25$. For video encoders, this frame also needs to be written back to the external memory, and therefore, the total bandwidth requirement becomes ~1.16 Gbps. However, if inter-encoding is used, additional reference frame(s) must also be read from the external memory, and from our discussion about Eq. (2.6), a reference frame must be read at least r_f times (usually, $r_f \geq 3$) for a single frame. With only one reference frame ($n_r = 1$) and $r_f = 3$, the bandwidth requirement surges to ~1.74 Gbps. To generalize, the bandwidth requirement for 8-bit per sample, YCbCr 4:2:0 video encoder, with n_r number of reference frames, can be written as:

$$w \times h \times 8 \times f_p \times (3 + r_f n_r) \tag{5.1}$$

However, it is conceivable that more than one video is processed concurrently by the video processing system. Such a paradigm is usually encountered in multicasting [2, 3] or multiview, 3D video processing [4, 5]. Consider that n_v full-HD views/videos are processed in parallel by the video application. For the system

mentioned above, the total external memory bandwidth required for intra-encoding is ~4.6 Gbps with $n_v = 4$, which can be supportable by the FPGA development kit. Additionally, note that external memory data transfer results in high-energy consumption. As given in [6, 7] , ~40% of the total system power is consumed by the external I/O, and the total bus power dissipation is directly proportional to the bus voltage toggles [8] . Therefore, increasing n_v not only increases the external memory bus contention but also results in a larger energy consumption.

5.1.2 Video Preprocessing

Video samples provided by the cameras may require preprocessing (e.g., filtering) before placing these samples in the external memory. Usually, a video input pipeline (VIP) is used to preprocess the samples, making them compliant to encoder's specification. Note that for multicasting scenario with n_v views, n_v individual VIPs and n_v frame memories in the external memory (to store the video frames) are required.

For real-time camera processing systems, the streaming video data from a video camera is fed to a VIP. An example VIP consists of a clocked video input sampler, a video frame clipper, a color plane sequencer, and an optional deinterlacer, as shown in Fig. 5.1. The video input sampler is used to synchronize the byte-serial data stream from the camera by handling clock-domain crossings. Clipper adjusts the resolution of the input video according to the specifications of the encoder. Plane sequencer is used to convert a serial video stream (with one luminance and two chrominance components) into a parallel stream for parallel processing of luminance and chrominance components. The deinterlacer is used if the input is from an analog video source. If the chrominance sampling does not match the encoder's specifications, an additional chroma resampler (e.g., 4:2:2 to 4:2:0) can be employed.

5.1.3 DDR Video Write Master

This module gathers the streaming video data from the camera and write it to the external memory (DDR in this case). An implementation of this module is shown in Fig. 5.2. Streaming YCbCr 4:2:0 pixels from VIP are routed to a 4:2:0 pixel FIFO.

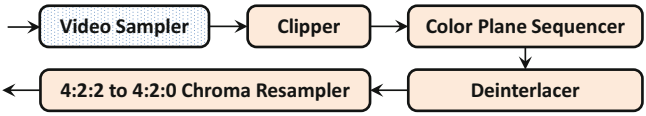


Fig. 5.1 Video Input Pipeline (VIP) module for a streamed video source

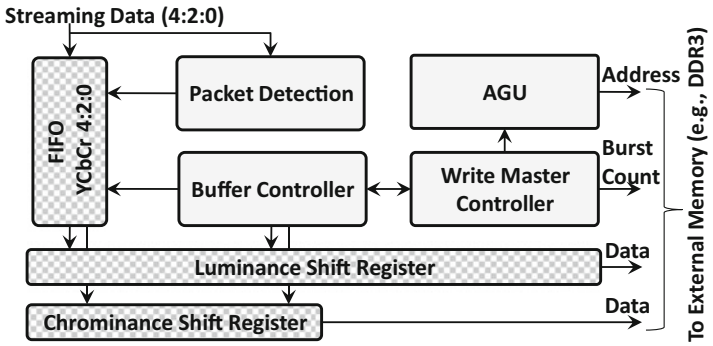


Fig. 5.2 Video writer hardware module

A packet detection circuit detects the arrival of video samples and generates write enable signals to the video sample FIFO. These pixels are forwarded to shift registers, and every luminance and chrominance component has a separate shift register. In this example implementation, the luminance shift register is a 128-bit wide, and chrominance shift register is 64-bit wide (owing to the 4:2:0 format and writing 16 8-bit luminance samples and 8 8-bit chroma samples at once to the external memory). The write master controller determines when to push the data from the shift-registers to the external memory. Depending upon the size of the video, the address generation unit (AGU) determines the write address. The write master controller also directs the AGU by selecting appropriate address in luminance or chrominance frame memories in the DDR3. The buffer controller provides the information whether a Cb or Cr address space needs to be written with the data from the chrominance shift register. It is possible that the software configures parts of this module. For example, it can configure the AGU to assign starting address of the frame.

For video applications, the frame data in the external memory is stored in a triple-buffering order (called the ring buffer). This ordering facilitates dropping the frames in case the video system gets slower or stalls (due to DDR bandwidth or Ethernet constraints). In such a scenario, only the current frame under process is retained, and the last written current frame in the ring buffer is updated, hence the name.

Moreover, to optionally write back the reconstructed block in case of video encoding application, an external memory write master is used. For H.264/AVC and HEVC intra-encoding, this write back to the external memory is not required, because the intra-encoding requires a single row of pixels above the current block (see Chap. 2 and Chap. 4). This row can be stored in an on-chip memory if large enough. However, for inter-encoding, write back of the reconstructed block is required since the on-chip memory capacity is usually not sufficient to hold a full reconstructed frame. The write master controller can also be configured via software, by writing appropriate values to the internal registers of the controller.

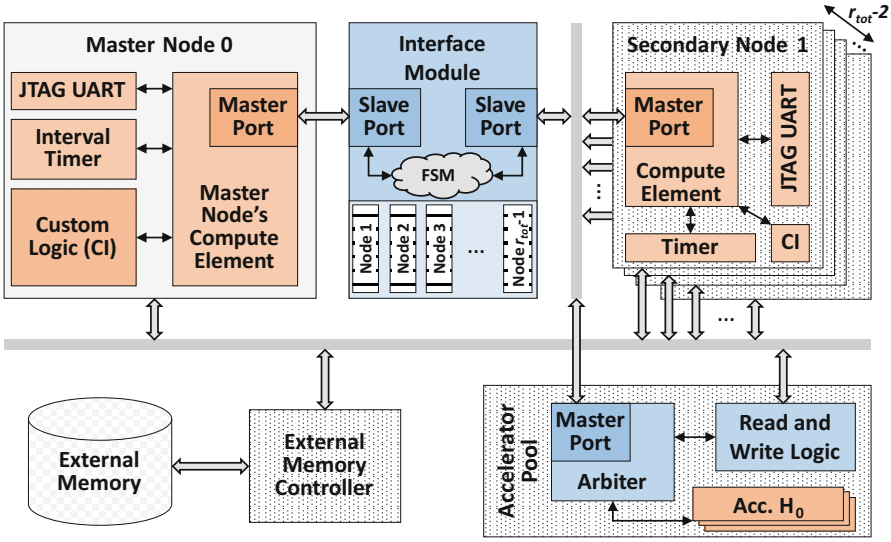


Fig. 5.3 Heterogeneous system with custom hardware modules and communication interface

5.1.4 Heterogeneous Computing Platform

As discussed earlier, a communication mechanism must be employed whereby the master node can communicate with secondary nodes in a heterogeneous system. This mechanism must be efficient and should result in minimal resource and communication overhead. For load balancing techniques employed on customized systems, one requires a synchronization and load distribution mechanism. One must consider the possibility of these mechanisms being employed on an accelerator-based multi-/many-core heterogeneous system, where multiple heterogeneous cores process the input data together with hardware accelerators. The architectural diagram of such a system is shown in Fig. 5.3. The cores can be architecturally different, and these cores can also have in-core hardware accelerators (in addition to the loosely coupled hardware accelerators). The in-core accelerators can be accessed via Special or Custom Instructions (SIs or CIs), e.g., as in Nios-II processor [9]. Thus, these cores take different amount of cycles and power to process the same job.

In addition to processing compute jobs, a master node is used to distribute the jobs among other cores and accelerators. This node is also responsible for setting the frequency of all the other nodes in the heterogeneous system (see Algorithm 5). Processing jobs can be assigned to the loosely coupled hardware accelerator modules, in which multiple accelerators pertaining to different functionalities can reside. All the nodes (including the soft-cores and the custom hardware accelerators) are connected to the external memory via an external memory controller.

For this architecture, the data is processed as frames, which resides in the external memory. Each frame (can also be called as a task for generalization) is

divided into constituent tiles (or subtasks), and the tiles can be processed in parallel by the nodes. The subtasks consist of multiple jobs, and each job contains a data block to process. Thus, the master node has information about the number of jobs within a task and determines the number of jobs within a subtask for each secondary node. Moreover, the master node has information about tasks' deadlines and the compute and power profiles of the secondary nodes.

The custom interface for communication among the nodes consists of a register file, which is filled by the nodes exchanging information. The master node writes to a specific address within the register file, associated with a specific secondary node. For example, the master node writes the start and end addresses of the frame memory for a specific secondary node, corresponding to jobs associated with the data blocks processed by the secondary node. A secondary node sends read request for its associated registers to a bridge controller (not shown in the figure) responsible for aligning the read requests from all the secondary nodes. If the master node has allocated a valid address and number of jobs to process, the node starts fetching this data from the external memory and starts processing. Once all the jobs assigned to a secondary node are processed by the secondary node, the secondary node writes to the appropriate status registers in the custom interface. When the master node receives this "jobs done" signal from all nodes, it signals end of frame and may gather statistics. Afterwards, the new frame processing starts whereby the master node sends new memory addresses to the secondary nodes via the custom interface.

5.2 Accelerator Allocation and Scheduling

In the previous section, architectural details about a heterogeneous system with custom communication interfaces are provided. This section provides algorithmic details about sharing a hardware architecture among different compute nodes in a heterogeneous system. Additionally, we will discuss how to extend the concept to use the same hardware architecture for concurrently processing multiple video streams.

5.2.1 Accelerator Sharing on Multi-/Many-Core

One of the major technological challenges discussed in Chap. 2 is the power-wall or Dark Silicon. A technique to solve this issue is to introduce highly efficient, custom hardware accelerators, which while running at lower frequencies (and hence resulting in lower spatial temperatures) can meet the throughput requirements. The hardware accelerators can be coupled to the compute nodes as shown in Fig. 5.3. However, allocation of the shared, loosely coupled hardware accelerator to multiple threads (or applications) requires properly weighing the computational capabilities of the nodes and their power profiles.

In this section, the following problem is addressed: how to allocate the shared hardware accelerator among the cores, such that the hardware is fully utilized, all application deadlines are met, and the power consumption of the complete system is minimized under a specific set of clock frequencies? For this purpose, an adaptive accelerator allocation (or scheduling) algorithm is discussed, to share a hardware accelerator by multiple, concurrently running independent application threads. This allocation not only accounts for meeting the deadlines of the tasks/applications running on the system, but it also helps to reduce the dynamic power by determining the voltages and frequencies of the soft-cores. Once a soft-core offloads its assigned jobs to the accelerator, the core can go into sleep state which reduces the power/temperature of the core. Summarizing, the following points form the basis of the allocation algorithm:

- *Scheduling the shared hardware accelerator,* by allocating the shared hardware accelerator among the soft-cores such that the hardware accelerator is maximally utilized when the soft-cores offload their tasks to the accelerator
- *Tuning the voltage-frequency of the cores,* such that the given throughput deadlines are met by all running tasks or applications
- *Objectively meeting the deadlines*, by distributing the workload on the programmable soft-cores and the hardware accelerator, such that the power consumed by the multi-/many-core system is minimized

For the following discussion, assume that several independent tasks are concurrently running on each compute core. In the coming text, we will only use tasks as independent entities for demonstrating this technique. However, this technique is equally applicable to concurrently running applications or threads of an application. Each task must be finished within a given deadline. Each task has an associated set of subtasks. These subtasks can either run on software or hardware. The objective is to offload the appropriate number of subtasks to the accelerator, such that the total power is minimized and system meets the deadline(s).

The outline of the heterogeneous computing architecture employing this algorithm is shown in Fig. 5.4. As seen, the "Monitoring and Control" generates the

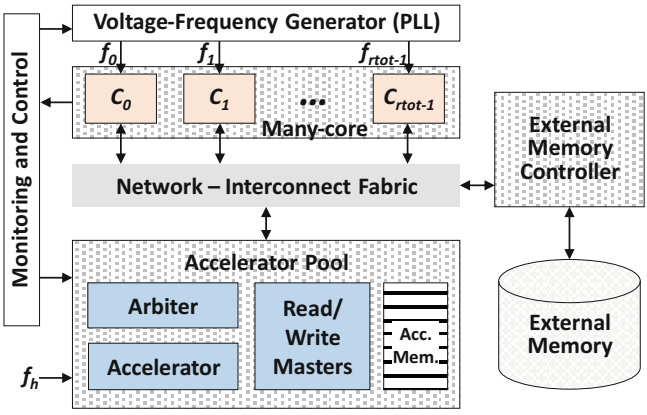


Fig. 5.4 Accelerator allocation architecture

appropriate signals to determine the frequencies of the cores and the accelerator allocation by deriving and solving an optimization problem based upon the system parameters (discussed later). Cores, accelerator, and the external memory communicate via interconnect fabric. The accelerator consists of an arbiter to regulate the core accessing the accelerator, read write control circuitries to access on-chip or off-chip memories, and internal SRAM scratchpad memory. It is assumed that the subtasks which can be offloaded to the accelerator can also be done locally by the core (via software). Further, it is also assumed that the frequencies of the cores can be independently adjusted and the clock frequency of the accelerator is constant and considerably low (i.e., the accelerator is always bright). Note that in this architecture, the accelerator populates its scratchpad by fetching the data directly from the external memory. Further, even if the size of data processed by a core and the accelerator is equal, the accelerator will still generate higher throughput, due to its custom logic implementation.

We will begin the details with modeling the system and then presenting a scheduling technique to determine the best accelerator allocation.

5.2.1.1 System Modeling and Objectives

Consider that a many-core system has r_{tot} nodes, and all these nodes can offload their subtasks to the shared hardware accelerator. The task i would consume $t_{s,i}$ seconds and $t_{h,i}$ seconds when run in software and hardware, respectively. For better understanding, an example accelerator allocation is diagrammatically shown as in Fig. 5.5. This figure shows the time consumption of each application on the programmable core and the hardware accelerator. Note that for an epoch of $t_{i,max}$ seconds, the total time for which the accelerator is engaged by all the cores is given

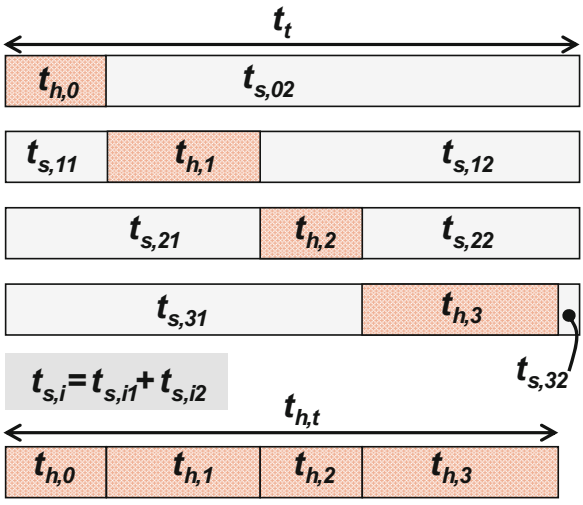


Fig. 5.5 Example breakdown of execution time on a 4-core system and a shared accelerator

by $t_{h,t} \leq t_{i,max}$. Furthermore, since a task is always run on a single core, therefore, the index i (for tasks) and k (for cores) can be used interchangeably (i.e., $i = k$).

As discussed above, the objective of the accelerator allocation technique is to minimize the power consumption of the complete system. If the power of a core k (p_k) is a function of its frequency f_k, then, mathematically, the objective is:

$$\min\left(\sum_{j=0}^{r_{tot}-1} p_j\left(f_j\right)\right) \tag{5.2}$$

At the same time, it is logical to maximize the hardware utilization which will basically increase the sleep period of the cores and thus save power and reduce the temperature. Mathematically, to maximize the accelerator utilization, the difference between the epoch time and the time for which hardware is engaged ($t_{i,max}$-$t_{h,t}$ $\in \mathbb{R}^+$, positive real) should be as small as possible. To do so, we proceed by writing the total cycles processed on the accelerator by all the cores in one second equal to:

$$c_{h,0} n_{\text{sec},h,0} + c_{h,1} n_{\text{sec},h,1} + \ldots + c_{h,k_{tot}-1} n_{\text{sec},h,k_{tot}-1} = \sum_{k=0}^{k_{tot}-1} c_{h,k} n_{\text{sec},h,k} \tag{5.3}$$

In this equation, $c_{h,i}$ is the number of cycles per subtask, and $n_{sec,h,i}$ is the number of subtasks per second for task i on the shared hardware accelerator. Therefore, if the difference $t_{i,max}$-$t_{h,t}$ needs to be minimized, the total number of cycles processed by all the cores on the accelerator per second must be matched to its clock frequency of the accelerator f_h. Note that the hardware is running at a fixed frequency. Mathematically, this constraint can be written as:

$$\sum_{k=0}^{k_{tot}-1} c_{h,k} n_{\text{sec},h,k} = f_h \tag{5.4}$$

Moreover, since the deadlines should be met, therefore, the added constraint is:

$$n_{\text{sec},s,k} + n_{\text{sec},h,k} \geq n_{\text{sec},k} \qquad \forall k \in \{0, \cdots, k_{tot} - 1\} \tag{5.5}$$

This equation shows that the number of subtasks that can be processed per second on the hardware ($n_{sec,h,i}$) and software ($n_{sec,s,i}$) should at least equal the number of total subtasks of the task $i = k$ per second ($n_{sec,k}$).

Moreover, the clock frequencies of the cores should be bound. Thus, an additional constraint is:

$$f_{k,\min} \leq f_k \leq f_{k,\max} \tag{5.6}$$

5.2.1.2 Optimization Algorithm

To optimize the function given in Eq. (5.2), we must derive $p_k(f_k)$ in terms of the system parameters, which can be tuned. If the cycles per subtask ($c_{h,k}$) and the

number of subtasks per second ($n_{sec,h,k}$) on accelerator by task $i = k$ are known, one can determine the time spent by task i on accelerator in the epoch t_t by using the formula:

$$t_{h,k} = \frac{c_{h,k} n_{\sec,h,k}}{f_h} t_t \tag{5.7}$$

Here, all quantities on the right-hand side are known. Thus, the time consumed on core k ($t_{s,k}$) can be determined by using the following relation:

$$t_{s,k} = t_t - t_{h,k} = t_t \left(1 - \frac{c_{h,i} n_{\sec,h,i}}{f_h}\right) \tag{5.8}$$

Using this equation, the frequency of the core can be determined by:

$$f_k = \frac{c_{s,k}(n_{\sec,k} - n_{\sec,h,k})}{t_{s,k}} \tag{5.9}$$

In this equation, $c_{s,k}$ is the number of cycles per subtask of task k on the associated soft-core. Here, we have used the identity given in Eq. (5.5).

Now, by inserting Eq. (5.9) in Eq. (5.2), the power consumed by a core can be written as:

$$p_k(f_k) = p_k\left(\frac{c_{s,k}(n_{\sec,k} - n_{\sec,h,k})}{t_t(1 - (c_{h,k} n_{\sec,h,k}/f_h))}\right) = p_k\left(\frac{\psi_1 - \psi_2 n_{\sec,h,k}}{1 - \psi_3 n_{\sec,h,k}}\right) \tag{5.10}$$

For readability, this equation uses ψ_1, ψ_2, and ψ_3 as constants given by:

$$\psi_1 = c_{s,k} n_{\sec,k}/t_t \qquad \psi_2 = c_{s,k}/t_t \qquad \psi_3 = c_{h,k}/f_h \tag{5.11}$$

Therefore, the objective function with constraints can be collectively written as:

$$\begin{aligned} &\min\left(\sum_{k=0}^{k_{tot}-1} p_k\left(\frac{\psi_1 - \psi_2 n_{\sec,h,k}}{1 - \psi_3 n_{\sec,h,k}}\right)\right) \\ &\text{subject to :} \\ &\sum_{k=0}^{k_{tot}-1} c_{h,k} n_{\sec,h,k} = f_h \\ &f_{k,\min} \leq \frac{\psi_1 - \psi_2 n_{\sec,h,k}}{1 - \psi_3 n_{\sec,h,k}} \leq f \quad \forall k \in \{0, \cdots, k_{tot} - 1\} \end{aligned} \tag{5.12}$$

Note that the optimization objective given in Eq. (5.12) is to find an appropriate number of subtasks which are offloaded to the accelerator ($n_{sec,h,k}$), for all cores. However, the optimization problem presented in the above equation is nonlinear, even if power and frequency approximately form a linear relationship for the given set of frequencies. In the case discussed here, the optimization (finding the value of $n_{sec,h,k}$ for all tasks) is achieved using Nelder-Mead technique [10]. Nelder-Mead is

a greedy heuristic that iteratively determines the value (v) of the given objective function for the given inputs and moves towards an optimum. The advantage of using Nelder-Mead technique is that unlike traditional mathematical methods, it does not require the derivatives of the objective function to be calculated. A brief discussion of the Nelder-Mead algorithm to traverse the optimum path is given below.

Nelder-Mead algorithm is an iterative method to find the inputs for which one gets maxima/minima of an objective function. At start, the inputs of the objective function are chosen which satisfy the given constraints. In every iteration of the Nelder-Mead algorithm, the objective function is evaluated for the set of intelligently derived set of new inputs (in this case, $n_{sec,h,k}$ for all the cores) to determine the "cost" of the objective function under these inputs. That is, for the given $n_{sec,h,k}$, the value $v{=}\sum p_k(f_k)$ is computed. Since in this case, constraints bound the search trajectory to find the optimum, therefore, the original constrained optimization problem is modified and converted into an unconstrained problem. Specifically, penalty function method is used [11]. The final form of the objective function is given in Algorithm 8. Here, the cost of the function increases if the difference between t_t and $t_{h,t}$ increases (maximum accelerator utilization, line 6). Since we try to maximize the accelerator utilization, therefore, a penalty is introduced depending upon the difference between the number of hardware operations and the total operations (lines 8-9). That is, in case the magnitude of violating the constraints increases for a given set of inputs, the penalty also increases, which in turn increases v. Therefore, the Nelder-Mead algorithm will deviate from this trajectory. Further, if the core frequencies that will support $n_{sec,s,k}$ are lesser than the minimum frequency ($f_{k,min}$), or larger than the maximum frequency ($f_{k,max}$), it will also result in increasing the cost of the function (bounded frequencies, line 10–12, Eq. (5.6)). Once the cost of the function v is determined, it is compared with the previous costs and algorithm moves in the direction of the lowest cost.

5.2.1.3 Evaluation of Accelerator Allocation

For illustrative purposes, application which computes the mean and variance of an image block is considered. This application will fetch a block of 4×4 pixels from the image storage and generate the mean and variance of the region. This type of block-based variance computation is important for texture classification and efficient image/video compression (e.g., see Sect. 4.3.3.2, where the CTU is divided into 4 × 4 blocks, and the mean and variance of each 4 × 4 is calculated). Computing the mean and variance of a block is considered as a single subtask.

For this evaluation, the Sniper x86 simulator is used along with the McPAT power simulator [12, 13] . For this subtask, our simulations show that the number of cycles per job on the soft-core $c_{s,k} = 492$ and $c_{h,k} = 70 \,\forall\, k$ cores. The frame is stored in the external memory. The cores process parts of the same video frame of different sizes, and the hardware accelerator partially shares the workload of each core, by allocating its resources according to the proposed algorithm. For

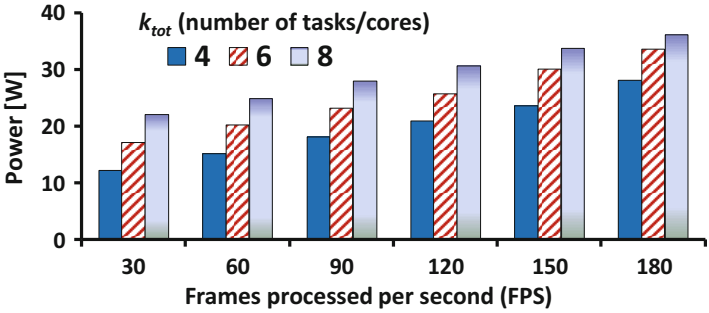


Fig. 5.6 Power consumption of the soft-cores for the given FPS requirement and the number of cores

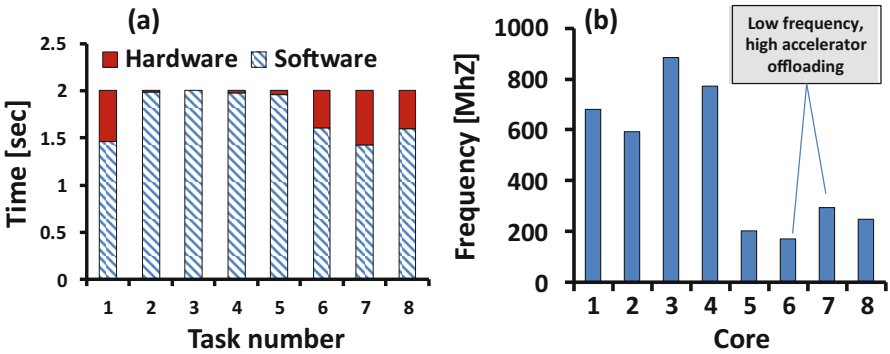


Fig. 5.7 (**a**) Time consumed per task by hardware accelerator and soft-core and (**b**) the corresponding frequency of the soft-cores for $n_{tot} = 8, fps = 120$

computational purposes, a part of video frame is brought to the internal SRAM scratchpad of the accelerator. Further, if the size of the task (i.e., number of subtask) a core needs to process or FPS requirement increases, the number of subtasks per second ($n_{sec,k}$) also increases. Therefore, more power and/or accelerator schedule should be allocated to that task.

Figure 5.6 shows the relationship of the power consumed by the system, by changing the FPS and the total number of soft-cores within the system with a single hardware accelerator. For this evaluation, a full-HD video of size 1920 × 1080 pixels is considered, and regions of this image (i.e., tasks) are distributed among the cores. The number of subtasks per second ($n_{sec,k}$) of each task is different, and thus, the accelerator demand varies for all these tasks. Further, we consider an epoch of size t_t=2 sec and f_h=100 MHz for this experiment.

Figure 5.7a shows an example allocation of the hardware accelerator to the tasks. Note that some of the cores (e.g., 2–4) mostly run their jobs on the soft-cores. Figure 5.7b shows the corresponding frequencies of the cores. As noticed, the tasks with considerable accelerator allocation are usually running at a much lower

frequency on the soft-cores. Furthermore, the frequency of a core also depends upon $n_{sec,k}$, and a larger $n_{sec,k}$ either requires more offloading or a higher core frequency, determined by the optimization program given in Eq. (5.12). Further, summation of the time consumed by the hardware accelerator will be almost equal to t_t, which shows that the accelerator is ~100% utilized.

5.2.2 *Multicast Video Processing Hardware*

In Sect. 5.2.1, the details about sharing a hardware accelerator among different soft-cores are given. The hardware accelerator processes the subtasks of the soft-cores in a round-robin fashion. However, for certain types of workloads (e.g., video encoding), processing a job or a block of video depends upon the output of the previously processed job(s) of the same task. Therefore, the hardware accelerator would need to have task-specific storage to adapt for every application (context switching), as would be the case in the multicasting scenario. Similarly, the data associated with different jobs can be coming from different sources, and therefore, a mechanism is required to efficiently provide this data to the hardware accelerator.

In the same direction, an efficient hardware architecture to realize concurrent processing of multiple videos on the same device is presented in this section. The goal of the architecture is to realize multicast scheduling of the H.264/AVC encoding loop using hardware replication and reutilization on a single device like an FPGA and to process multiple video sources in real-time, area-efficient manner. Using a single device results in lesser cost and easier management of the system. Note that the same architectural principles can be extended for HEVC and other encoders.

A high-level overview of multicast H.264/AVC intra-encoder is given in Fig. 5.8. Only a single H.264/AVC encoder is used for processing n_v video inputs. Before encoding the individual videos, the input video is preprocessed and written to the frame memories in the off-chip DDR3 memory. The encoder reads video frames from the DDR3 memory and compresses them. Encoding is achieved using custom hardware which will be further discussed in Sect. 5.3. As seen, a single soft-core is used to initialize the control registers of the VIPs and the I/O ports. It also commences the encoding, whereby the hardware coprocessors start fetching video samples from the external memory.

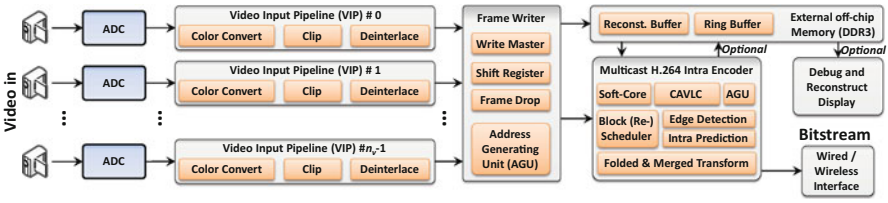


Fig. 5.8 Overview of a streaming multicast H.264/AVC Intra-only encoder

Once the current frame buffer is filled with a video frame, encoder fetches the data from the external memory, using the external memory read master. The encoder collects burst of data from the external memory and rearranges them in a "MB-FIFO." Each entry in the FIFO is one MB-row width wide (16 16-bit samples, 256-bits). Note that there is a separate FIFO for each of the n_v views. Every "MB-FIFO" is actually a set of three separate FIFOs, one for luminance and two for chrominance components (Cb and Cr). The luminance FIFO is of size $16 \times 16 = 256$ samples of 16-bits, and each chrominance component is $16 \times 16/4$ samples. Note that these MB-FIFOs are also used for clock-domain crossing as the DDR3 and the encoder may run on separate clocks. When the data in MB-FIFO is available, it is pulled by the H.264/AVC intra-encoder.

Now, the mechanisms by which video data of distinct video sources is forwarded to and collected from the video encoder are discussed.

5.2.2.1 Video Block Scheduler and Rescheduler

As mentioned earlier, processing jobs might depend upon the output of the previously processed jobs and hinder pipelined operations. Same is true for intra-encoding a block of video frame, which cannot be pipelined, and a block must wait for the previous block in the encoding loop to finish (see Sect. 2.2.1.1). Further, blocks must be processed in raster scan. Thus, the building modules of the encoding loop are idling if not in use, and only one module of the encoder is active at one time, giving rise to the so-called bubbles ($< 100\%$ hardware utilization). If we consider the impact of these bubbles on the video encoder, then large latency, area, and energy overhead are incurred, as the modules are idle most of the time.

Thus, it is proposed that instead of using n_v encoders in parallel to process n_v-independent frames, encoder's modules should be reutilized in a time-multiplexed manner. This is accomplished by a block-level scheduler, which is used to push blocks of each view into the encoding loop in round-robin fashion. The scheduler is illustrated in Fig. 5.9a, and an example schedule for $n_v = 4$ is shown in Fig. 5.9b. This will help in increasing the hardware utilization and reducing silicon area footprint. Additionally, the total energy consumption of the encoder decreases. However, to generate the bit-streams, a separate entropy coder (EC) unit for each video is required. In this architecture, context adaptive variable length coder (CAVLC) [14] is used as an entropy coder; however, other entropy coders (like context-adaptive binary arithmetic coding, CABAC) can also be used. The CAVLC units are fed via a rescheduler. Using a single EC unit for the multicast encoder is difficult to realize due to two factors.

- Firstly, a separate EC unit per video stream is essential because the EC units like CAVLC requires at least $16 \times 16 + 2 \times 8 \times 8 = 384$ cycles to process one full luminance and two chrominance blocks. This is because each quantized coefficient must be coded using the output of the previous quantized coefficient and hence the name context adaptivity. Additionally, the bits generated by these

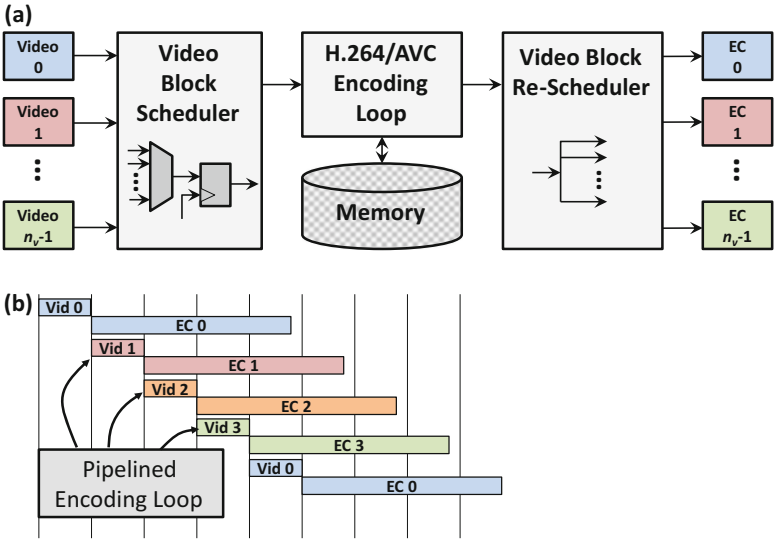


Fig. 5.9 Multicast video encoder with (**a**) block processing scheduler and (**b**) example schedule with $n_v = 4$

coefficients must be pushed into a bit-buffer in serial fashion. If we consider encoding 4 full-HD videos (4K Ultra-HD of size 3840 × 2160 pixels, i.e., four 1920 × 1080 frames) at 25 FPS, then we can maximally consume 183 cycles/block cycles if the encoder runs at a clock frequency of 150 MHz. The 384 cycles per block requirement is larger than the 183 cycles/block cycle budget.

- Secondly, the context adaptivity required in EC can only be realized by using independent data buffers for each view, which is very hard to maintain by a single EC unit and will incur large latency.

Additionally, note that the output of the scheduler is registered because of a MUX in front of the MB loop. Contrarily, rescheduler's output is directly connected with all CAVLC units. Only a "valid" signal is required which determines the CAVLC unit to which data is directed.

Instead of the block-based CAVLC units, it is possible to insert block-based CABAC units [15] seamlessly in the proposed multicast encoder. The proposed multicast video encoding technique is independent of the type of entropy coder used. Note that CABAC provides better compression efficiency compared to CAVLC (up to 15%). However, note that 15% bitrate savings by using CABAC instead of CAVLC will only occur in ideal scenarios. Further, the latency incurred by CABAC will be higher compared to CAVLC, as CAVLC is the low complexity entropy encoding alternative in H.264/AVC standard.

Moreover, the multicasting scenario is explained with the help of H.264/AVC, and it is also applicable to HEVC and other state-of-the-art video encoders. The concept of module sharing among multiple videos can be applied to other video applications as well (e.g., 3D video processing).

5.3 Efficient Hardware Accelerator Architectures

In Sect. 5.2, different techniques for allocating the shared hardware accelerator to multiple threads or applications are presented. This section deals with efficient design of some hardware accelerators, specifically for video encoding applications (H.264/AVC and HEVC).

5.3.1 Low Latency H.264/AVC Encoding Loop

A multicast solution for H.264/AVC is given in Fig. 5.8. The architectural details about the H.264/AVC video encoder modules used in that design are discussed here. The reader is advised to refer to intra-encoding discussion in Chap. 2 for better understanding. The goal of the architecture is to provide high throughput, low latency, and area efficiency and be capable of encoding full-HD views in real time. A high-level architecture of an H.264/AVC intra-encoder with its main functional blocks for a single video input is shown in Fig. 5.10. As hinted previously, the H.264/AVC intra-encoding loop has inherent sequential dependencies, which limit the throughput of the encoder. Blocks or MBs of an input frame are processed in raster scan order through the encoding loop (marked as dashed red arrow). The prediction of each MB is generated using the pixel values of the left, upper, and upper-left reconstructed MBs (intra-prediction generator in Fig. 5.10). The residue $X^{(r)}$ (i.e., the pixel-wise difference of the predicted MB X' and the

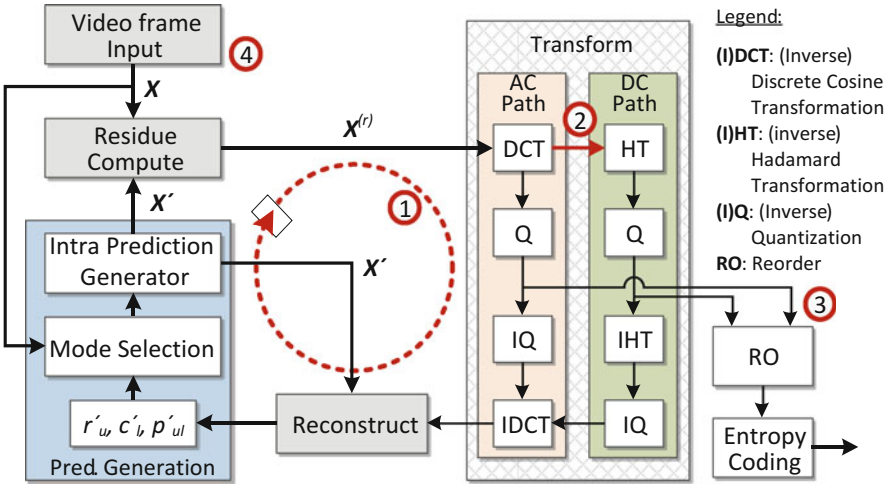


Fig. 5.10 H.264/AVC intra-encoding loop. The current MB is labeled as X, and the predicted and residue MBs are labeled X' and $X^{(r)}$, respectively; sequential data dependencies are shown by *dashed arrows* (① and ②)

actual MB *X*) is sent to the transform module. There, the residue is processed by the discrete cosine transformation (DCT), Hadamard transformation (HT), and quantization (Q) and forwarded to the entropy coding (EC) module. Additionally, the video is locally decoded, i.e., processed by inverse quantization (IQ), inverse DCT (IDCT), inverse HT (IHT), and reconstruct module. The reconstructed MBs are required as a base for the next predictions. There are various data dependencies inside this loop, and those with the highest significance for encoding performance are explained here:

- *Dependency 1*: The main dependency comes from the fact that MB processing cannot be pipelined. Before entering the encoding loop, the current MB must wait for the previous MBs in the loop to be fully encoded and then locally decoded to be used for prediction generation. The dashed arrow labeled ① in Fig. 5.10 portrays this dependency.
- *Dependency 2*: The transform module consists of two paths, one processing the AC part of the spatial frequencies and the other processing the DC part. The outputs from the DCT are fed both to HT in the DC path and Q in the AC path. However, note that algorithmically, HT cannot start execution until the complete MB is processed by DCT, but the Q and IQ modules in the AC path can start processing after a part of MB is processed. This dependency is shown as arrow labeled ② in Fig. 5.10. Additionally, to compute IDCT, data from both DC and AC paths is required. Therefore, IDCT module should stall and wait for data from the DC path.
- *Dependency 3*: The entropy coding (CAVLC or CABAC) processes the DC output coefficients before the AC coefficients, but the DC coefficients are generated later from the HT module. Label ③ denotes this dependency. Moreover, the entropy coding technique also requires reordering the data before processing it. This adds to the latency in output generation.
- *Dependency 4*: Video frame samples in the form of MBs are brought from camera or off-chip memory to the encoding modules (shown by Label ④). This transmission incurs high latency in output generation if the encoding loop's workload and efficient reshaping of video samples into MBs are not considered.

In a nutshell, low latency and high throughput for H.264/AVC encoder are obtained by addressing the dependencies presented above. The resultant architecture with hardware coprocessors is shown in Fig. 5.11. This figure depicts the design of H.264/AVC hardware accelerators and their interconnections with the video I/O (Sect. 5.1) and multicast enabling modules (Sect. 5.2.2.1). Area and computational efficiency is obtained by designing fast, area-efficient circuits. Instead of a single module of the encoding loop, the focus of this section is on the complete H.264/AVC intra-encoding system. In short, the following main techniques are employed which will be discussed in the coming text:

- *Adaptive H.264 intra-prediction technique* by utilizing a small and speedy edge extraction hardware for scheduling the generation of different intra-prediction.

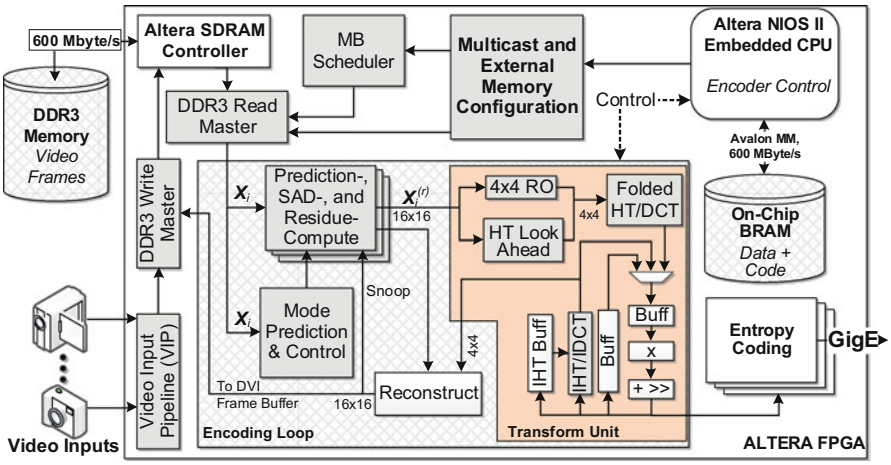


Fig. 5.11 Hardware accelerator architecture for multicast H.264/AVC Intra video encoder. This design also shows the connections of the multicast encoder with the I/O ports of the system (camera inputs and Ethernet output)

This allows the user to configure the number of predictions to test per MB, thereby configuring the throughput of the encoding loop.

- *Area-efficient transform module design* where AC and DC path of the encoding loop are decoupled to reduce latency. Moreover, folding DCT/IDCT, utilizing the same hardware resources, and interlacing of Q/IQ blocks reduce the total silicon footprint of the platform.

5.3.1.1 4×4 Reordering and HT Lookahead

To reduce the latency of the transform stage (Dependency 2 in Sect. 5.3.1), a technique is presented to decouple the AC path from the DC path in the transform module. The inner modules of the transform unit work at a 4×4 granularity. Thus, the residue 16×16 block is subdivided into 16 4 × 4 sub-blocks using the 4 × 4 reorder (RO) stage as illustrated in Fig. 5.11. The residue generator stage provides one line of residue $X^{(r)}$ (16 pixels in 1 cycle, i^{th} line of the MB given by $X^{(r)}{}_i$). Whenever four $X^{(r)}{}_i$ are stored in the input registers of the 4 × 4 RO stage, RO generates four 4×4 blocks and pushes them to the input FIFO of the 2D-DCT stage. As mentioned in Sect. 5.3.1, HT cannot begin until all 16 4×4 blocks are handled by the DCT. However, by simplifying the DCT formula, we observe that the n^{th} output DC value (which is the n^{th} input value $HT^{n}{}_{in}$) obtained from the DCT by processing the n^{th} 4×4 residue block ($X^{(r)n}$) can be calculated by:

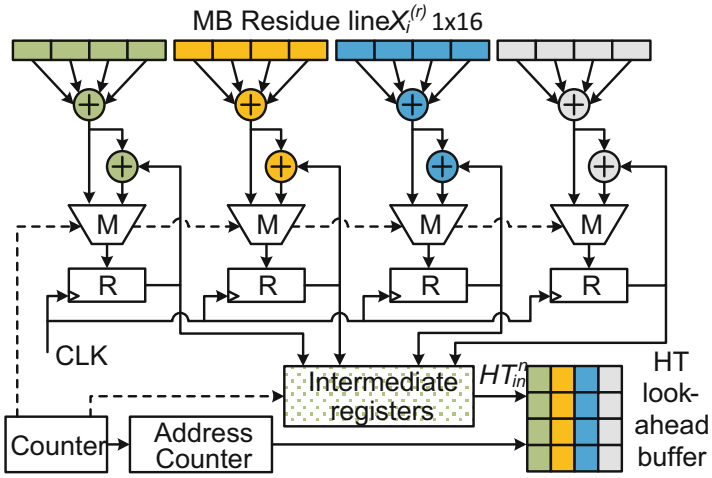


Fig. 5.12 HT-Lookahead buffer architecture, responsible for pre-computations of *HT* input

$$HT_{in}^{n} = \sum_{i=1}^{4} \sum_{j=1}^{4} X_{i,j}^{(r)n}, n = 1, \ldots, 16 \tag{5.13}$$

This shows that HT^n_{in} can be generated by adding all the residue samples in the 4×4 block. Thus, the DC outputs are alternatively generated at RO stage instead of the DCT stage, effectively decoupling DCT from HT. This computation is architectural depicted in Fig. 5.12. A residue line can be added to the current accumulators (R) of the lookahead HT memory, or it can trigger a new DC coefficient generation, controlled by the counter administering the MUXes M. Using the above technique by generating HT-transformed coefficients ahead of DCT-transformed coefficients results in reduce latency, as the entropy coding can start earlier. Since IDCT module requires IHT transformed data (Sect. 5.3.1), therefore, the output of AC inverse quantization (i.e., 16-bits per pixel, 256-bits per 4×4) must be stored for the complete MB in an intermediate buffer (256×16 = 4096 bits). However, this technique eliminates the need of the intermediate buffer because DC path is executed first, and the IDCT can process a 4×4 IQ, without needing an additional storage.

The entropy coding module needs the 4 × 4 blocks in a frequency scan or Z-fashion [14]. The RO unit presented here generates the 4 × 4 blocks for the 2D-DCT in Z-fashion on the fly, eradicating the need for an extra RO unit in front of entropy coding. This lessens the latency of the complete encoder (addressing Dependency 3 from Sect. 5.3.1).

5.3.1.2 Transform and Quantization Module

To save area, we can employ DCT and HT in the same hardware block, in a time-multiplexed manner. Since the DCT and HT equations are similar, these transforms can be implemented using butterflies [16, 17]. Both HT and DCT butterflies are composed of two stages, one for horizontal and the other for vertical transform. A butterfly is designed such that it can process four inputs simultaneously. Therefore, eight butterflies are required to process a 4×4 block. To save circuit area, the architectures can be folded, and the butterflies are reused for both horizontal and vertical transform in HT/DCT and IHT/IDCT. This architecture is illustrated in Fig. 5.13, in which a MUX in front of the butterflies determines whether to push new samples or previously computed values (i.e., second pass) to the butterflies. In this architecture, the horizontal and vertical transformation is implemented via rewiring the output. Therefore, the output of the complete transform is produced with a single cycle delay and the module is not fully pipelined, though it is utilized all the time. As shown in Fig. 5.13b, the IHT/IDCT module is also inter-ready (as it can discard inputs from the IHT buffer using an additional MUX).

The data flow graph of the output from HT/DCT butterflies till the input of CAVLC is presented in Fig. 5.14. From Point A to Point B, the flow graph denotes the rewiring of Fig. 5.13, i.e., in the first iteration of the folded HT/DCT. In the next iteration, the rewire function again transforms the data from Point B till Point C. The quantizer does not rearrange data, and therefore its impact is not illustrated in this figure. Since the data is jumbled, therefore, before feeding this to the CAVLC FIFO, it is rewired from Point C to Point D. This is because the CAVLC requires the output data organized in the Z-fashion [14]. Thus, an additional latency is saved by avoiding the rewiring of the 4×4 quantized coefficients.

Like reusing hardware in HT/DCT and IHT/IDCT, the presented architecture combines the quantizer (Q) and inverse quantizer (IQ) in DC and AC paths. The

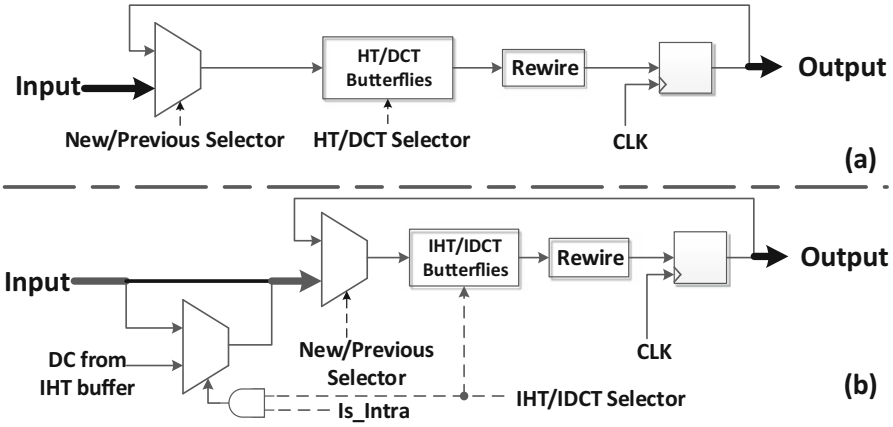


Fig. 5.13 Folded architecture of (**a**) DCT and (**b**) IDCT for H.264/AVC

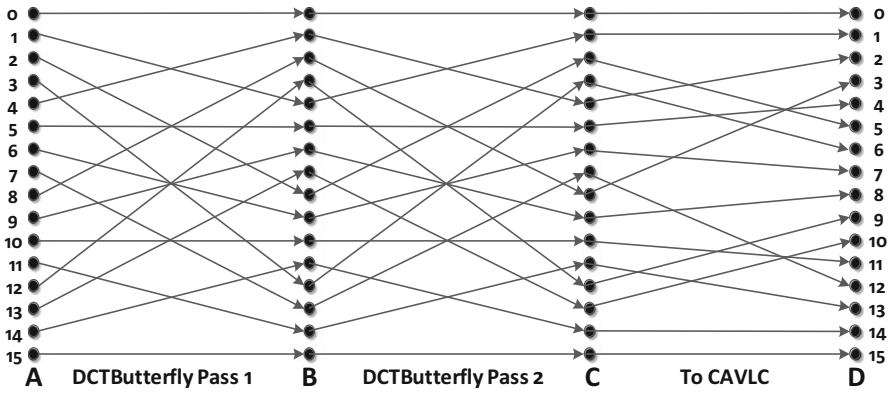


Fig. 5.14 Data flow graph from the output of HT/DCT butterflies to the input of the entropy coder (CAVLC)

resulting architecture is shown in Fig. 5.15a. The quantization and inverse quantization relationships, both for DC and AC paths, are shown in Eq. (5.14).

$$\begin{aligned} Q_{DC,i} &= \left(|DC_i| \times q_i + 2q_{i,\text{const}}\right) \gg \left(q_{i,\text{bits}} + 1\right) \\ Q_{AC,i} &= \left(|AC_i| \times q_i + q_{i,\text{const}}\right) \gg q_{i,\text{bits}} \\ IQ_{DC,i} &= \left(\left(\left(Q_{DC,i} \times dq_i\right) \ll q_{i,\text{per}}\right) + 2\right) \gg 2 \\ IQ_{AC,i} &= \left(Q_{DC,i} \times dq_i\right) \ll q_{i,\text{per}} \end{aligned} \tag{5.14}$$

Depending upon the QP value, quantization coefficients (q_i), inverse quantization coefficients (dq_i), and other values ($q_{i,const}$, $q_{i,bits}$, $q_{i,per}$) are selected. Since the goal is to reuse hardware for maximum area efficiency, the multipliers and barrel shifters are shared by interlacing the Q and IQ modules. This architecture is shown in Fig. 5.15a.

Note that the one cycle latency in the DCT's output is exploited in combined Q/IQ unit as shown by the schedule in Fig. 5.15b. This schedule represents how we can reuse the multipliers and barrel shifters for Q and IQ units by purposely delaying the DCT output by one cycle due to folding. Note that multipliers are costly in terms of area (usually implemented via DSP modules in an FPGA and there is a limited number of DSP modules available). Further, the shift operation requires barrel shifters, and for a 16-bit input, each shifter requires 64 multiplexers. However, in this architecture, an extra output buffer (delay buffer) is required for assistance in scheduling. Moreover, the folded architecture of IDCT is also provided with valid inputs in alternate cycles by the IQ stage. A single-bit shift-register circuit that mimics the valid data propagation through the registers generates the output valid signal for the complete HT/DCT, Q/IQ, and IHT/IDCT stages.

In summary, using the HT/DCT (IHT/IDCT) architecture given in Fig. 5.13, we can effectively reduce the area almost by half compared to the standard H.264/AVC transform stage implementations (see Fig. 5.10). First, only four butterflies are used

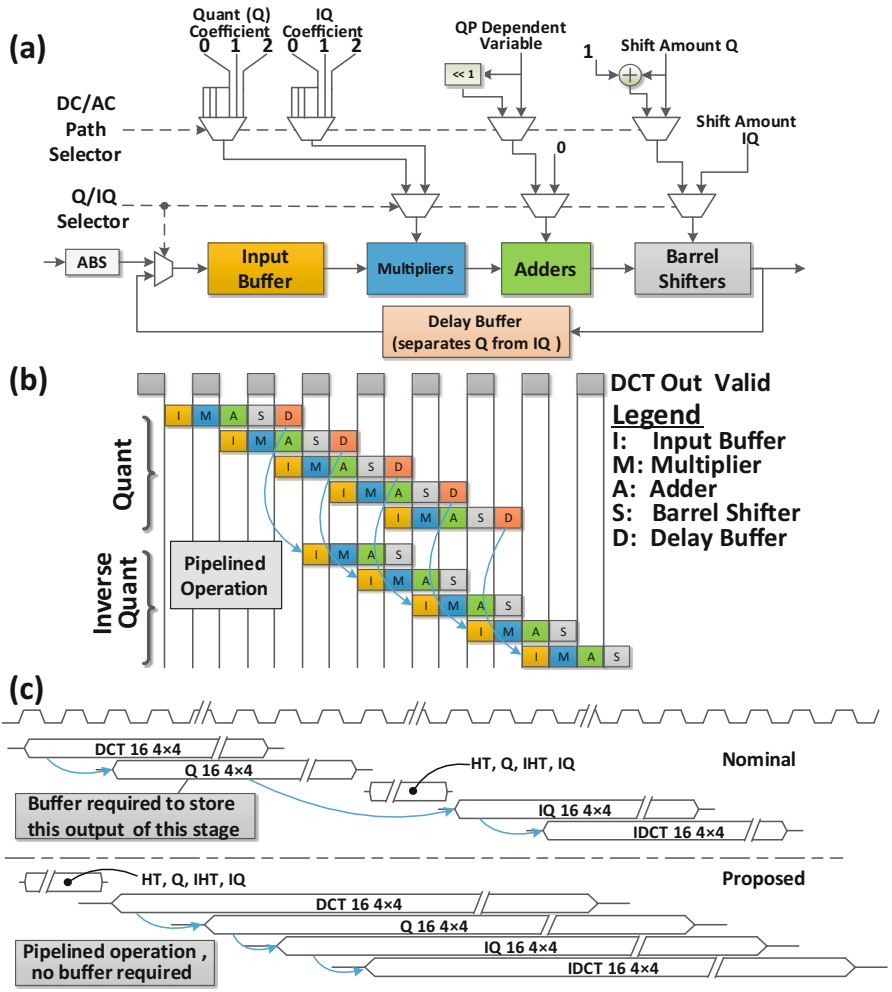


Fig. 5.15 Modules of the H.264/AVC encoding loop. (**a**) Interlaced H.264/AVC quantizer and inverse quantizer, (**b**) scheduling the quantizer/inverse quantizer on the same hardware, and (**c**) schedule of (inverse) transform and (inverse) quantization using nominal and the architectures presented in this book

instead of eight per transform due to the folded architecture. Secondly, since the 4×4 output of DCT is delayed by a single cycle, we allow Q and IQ to share the multipliers and the barrel shifters. Therefore, only 16 multipliers and 16 barrel shifters are used, instead of 32 multipliers and 32 barrel shifters. Further, folding of transform modules and interlacing of Q and IQ modules result only in a penalty of two computational cycles. This is shown in Fig. 5.15c. In the nominal case, the DCT and AC quantization is carried out for the complete MB (i.e., 16 DC coefficients are generated). Afterwards, the DC path can commence, which is followed

by pipelined AC inverse quantization and IDCT. However, in the technique presented here, DC path is executed first due to the HT lookahead technique (see Fig. 5.12). Afterwards, the DCT, quantization, inverse quantization, and IDCT can process the data in a pipelined manner.

5.3.1.3 Prediction Generation and Mode Decision Module

The mode decision module must select one prediction out of the four predictions for H.264/AVC (which are usually termed as V, H, DC, and P for 16 × 16 MB). In this section, the term "mode" refers to only one of the four modes mentioned above, and each mode has an associated prediction which is used for computing the SAD. The best mode/prediction is the one which generates the lowest SAD. Usually, one can use the term mode and prediction interchangeably.

The encoder can use a full-search approach to select the best prediction (i.e., the best X') using either SATD or just SAD between X and X'. The selection of the best prediction is slow if the generation of X' and the SAD calculation for every prediction is computed sequentially using only one hardware unit. On the other extreme, the parallel generation of all four predictions and their SAD computation would demand a significant amount of hardware. Therefore, to reduce the area overhead, only 16 adders/subtractors are used in this architecture, and thus, SAD of a luminance MB can be generated in 16 cycles. Additionally, note that generating the P prediction is computationally involved compared to the other modes.

Figure 5.16 shows the mode decision module. A fully reconstructed 16×16 MB is available from the reconstruction block (generating one reconstructed MB line) after 20 cycles. FIFO_{Xi} contains the current MB X, and this data is then written to a shared on-chip memory, which stores the MB in a line-by-line fashion. The

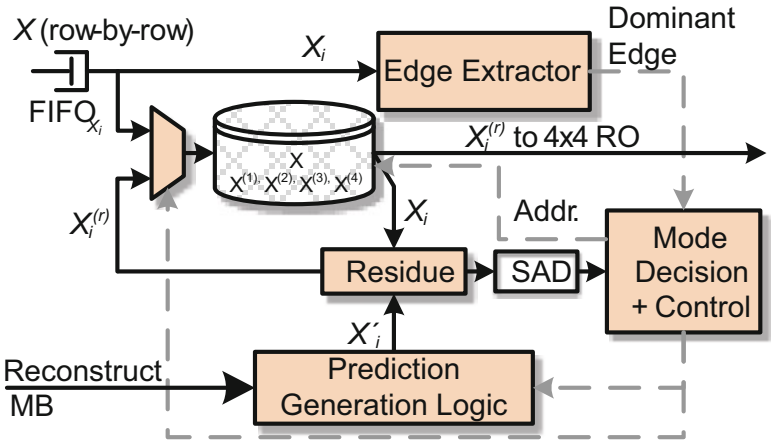


Fig. 5.16 Mode decision module for Intra 16 × 16

pipelined edge detector (discussed later in Sect. 5.3.1.4) unit predicts the order of the predictions to test by finding the most dominant edge for X. Using this order, predictions X' are generated by the prediction generator and stored in the same on-chip buffer as X at distinct address space. The quality of the prediction is evaluated by calculating the SAD of the prediction. The residue calculator generates $X^{(r)}{}_i$ and passes it to both the SAD unit and the residue memory block. After residues for all the predictors are stored, the residue resulting in the lowest SAD value is written to the 4×4 RO stage. Note that if the amount of cycle budget for encoding does not comply with the resolution and frame rate requirements, the SAD unit can be configured to use the downsampled version of the current MB for residual calculation. A downsampling factor, denoted by d_s, means that every d_s line of X is used for the SAD computation. Therefore, when $d_s > 1$, the number of cycles for SAD computation decreases. For example, for one luminance MB with $d_s = 2$, it takes 8 cycles for the SAD computation instead of 16 with $d_s = 1$. However, note that if d_s is larger than 1, $X^{(r)}$ is also downsampled, and thus it must be updated with the full residue after final mode selection.

5.3.1.4 Edge-Based Prediction Prioritization

If the encoding cycles do not comply with the allotted budget, it becomes necessary to sacrifice some encoding efficiency to fulfill the performance constraints. Since the transform loop is essential to the encoding process, then cycles from the mode decision module are scrapped to meet throughput deadlines. Unlikely predictions are not entertained as prediction candidates for residue generation by using a preprocessing stage. This procedure is not required for parallel SAD implementations, but it is useful for a sequential prediction mode decision circuit. Various algorithms and architectures are proposed in literature for estimating the best predictor and elimination of unlikely predictors [18, 19], where texture-based edge extraction information is used to determine the probable prediction candidates. Once the reconstructed data is available, the most suitable predictors are tested first, and testing the other predictors is either delayed or even skipped. One can use a Sobel-based edge extraction technique. However, an edge extraction procedure for a 16×16 block will thus require 256 iterations (requiring a $\tan^{-1}$ function and a divider) plus additional cycles to determine the dominant texture direction. Therefore, it cannot be embedded as a processing stage in the encoding loop or parallel to the encoding loop for large resolution sequences like 4K-UHD (encoding loop must finish within 183 cycles for a clock frequency of 150 MHz at 25 FPS). However, it can be realized as a separate pipeline stage outside the encoding loop. This introduces latency. In addition, the area overhead might become prohibitive if parallel edge extractors are utilized. Moreover, these algorithms work using thresholds, and usually, the edge threshold is decided at design time but it is not applicable to every video scene. In contrast to this, the technique presented here uses a lightweight and efficient prediction mode estimation process

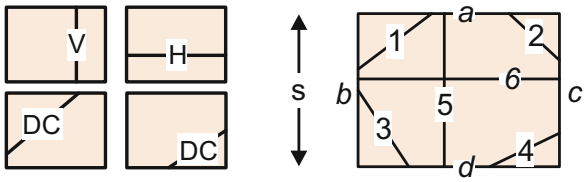


Fig. 5.17 Edge extraction technique employing running difference

with a modified version of edge extraction procedure. It extracts the dominant edge information from the input MB and does not require an edge threshold.

For the current MB X_n, the dominant or most probable mode estimation is carried out in parallel with the residue calculator and transform stage. Further, this architecture can generate the likely prediction modes before the reconstructed previous MB X_{n-1} is available at the prediction generation stage. The crux of the method involves the estimation of edge pixels at the borders of the MBs by computing a sequential running difference *rd*. Since we consider a 16 sample MB line X_i being processed within a cycle, therefore, only one subtractor and one ABS (absolute) unit are used for both the top and bottom borders. Also, one subtractor and one absolute unit each for the left and right borders are used to detect the dominant edge direction. For example, at the vertical border *b* or *c* of the MB in Fig. 5.17, rd_i is computed as:

$$rd_i = \left|X_{i,j} - X_{i+1,j}\right|, \forall i = 1, \ldots, 15, j \in \{1, 16\} \tag{5.15}$$

The pixel location *i* where rd_i is maximum is acknowledged as the point where the edge passed. Let $e_b{=}i$ be the location of the edge passing through *b* at *i*, generating the maximum rd_i given by rd_b. Similarly, edge location *e* and running difference *rd* at each border are computed, and the two borders with maximum *rd* are declared to have an edge passing through them. Thus, there are six distinct possibilities for the edges as illustrated in Fig. 5.17, and only one edge out of the six can occur.

By using this edge extraction technique, the conditional probabilities of the final mode selected by the full-search procedure for every line using various HD sequences are computed. Based upon these conditional probabilities, Algorithm 9 is devised for selecting the precedence order *m* of prediction generation. When the difference between all rd_i is less than some threshold (currently, we kept it constant at 5 as there were no observable dependencies on the input data), it is concluded that there is no edge in the block, and hence the algorithm detects $Line_0$ or no line. Additionally, the planar prediction (P) is never selected as the first mode to test in Algorithm 9 (implemented as a decision-tree), and therefore, intermediate values for P [14] can be generated in parallel to the other modes preceding it. Thus, P mode prediction/residue computations take the same amount of cycles as the other modes, although it is more complex.

A parameter θ is defined as the number of allowable modes/predictions to test. To meet the cycle budget of the encoding loop, θ can be altered to compute SADs

for only a subset of the prediction candidates. With $\theta = 1$, there is no need of SAD computations, and the most probable mode is used to generate prediction, and its residue is forwarded directly to the transform stage. For $1 < \theta \leq 4$, the algorithm starts with the most probable mode in m and computes the SADs sequentially.

5.3.1.5 Evaluating the H.264/AVC Architecture

In this section, some evaluations of the H.264/AVC encoding loop are presented. For comprehensive details, refer to Appendix C.

Encoding Loop In the encoding loop presented above, the transform stages are merged to save area and power while incurring small cycle penalties. In Ref. [20], authors have also designed a multi-transform engine, which merges all transforms (HT, IHT, DCT, IDCT) in the transform loop. Contrarily, the architecture presented in this book only merges HT/DCT and IHT/IDCT. However, the total area consumed by the two separate transform units presented here is lesser, because [20] implements large multiplexers. See Table 5.1. Further, the increased area in the merged Q/IQ unit of this book is due to the delay buffer used by our approach for interlacing Q and IQ (see Fig. 5.15). The complete transform unit proposed in Ref. [20] uses 110.29 K gates, while the architecture explained above uses 106.45K gates. For a throughput of 16 pixels per cycle, [20] corresponds to 6.89K gates per pixel and 63.5 mW per pixel. Compared to this, 6.65K gates per pixel and 61.77 mW per pixel are used by the architecture presented above. Note that the encoding loop presented here is merged within the multicast H.264/AVC video encoder for n_v parallel videos as given in Sect. 5.2.2. Therefore, the area and power consumption of [20] should be multiplied with n_v, as [20] does not provide hardware sharing for multicasting solutions. On the other hand, we use the same hardware for all the encoders. Thus, with $n_v = 4$, the architecture proposed here can theoratically reduce the area of the transform unit by ~4.14×.

Edge-Based Mode Prediction For the edge-based mode estimation, Fig. 5.18 reports the average hit rate of the likely mode precedence per frame. H correlates the prediction mode selected after full-search to the likelihood of the ordered mode candidates m generated by the Algorithm 9. In a nutshell, this plot denotes the accuracy of Algorithm 9. Here, $H = 4$ is the worst misprediction, which means that the intra-prediction mode finally selected after comparing the SADs of all four modes is the least probable mode according to prediction algorithm. H=1 denotes that the final mode selection after full search and the highest priority mode selected

Table 5.1 Area footprint of the transform unit for TSMC 65 nm technology [21] , power consumption via Altera [22]

	Baseline	This encoder	[20]
Transform	21.68	15.04 K	21.39 K
Q/IQ	67.32 K	60.52 K	51.62 K
Total	129 K	106.45 K	110.29 K
Power (mW)	1039.23	988.38	1015.93

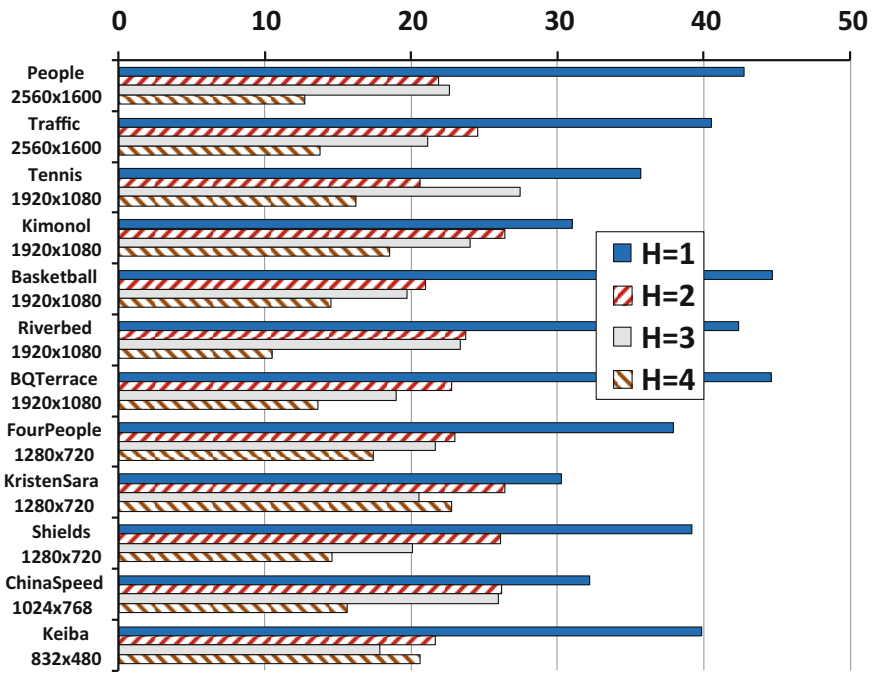


Fig. 5.18 Percentage hit rates (avg. over QPs 18. . .32, step size = 2) for different sequences; H: priority of full search within mode schedule m

by the edge extractor are the same. Notice that $H = 1$ has a higher value compared to other cases.

For evaluating the quality of the proposed edge-based most probable mode selection, plots of PSNR against the bitrate (RD curves) at $\theta = 1$ are given in Fig. 5.19. These curves suggest that the RD curve for likely mode estimation at $\theta = 1$ matches the full-search intra-mode (also called closed loop (CL)) selection closely. As a comparison to the likely mode selection procedure presented in this book, the RD curve for open loop (OL) algorithm [23] with $\theta = 1$ is also plotted. Notice that the algorithm presented here outperforms the OL algorithm.

Further, the edge detection approach presented in [18] uses 8.4K gates, while edge detection architecture presented here only uses 4.4K gates for TSMC 65nm technology. Moreover, [18] requires testing at least two intra-modes ($\theta \geq 2$), whereas the presented algorithm also allows for testing a single mode ($\theta \geq 1$).

5.3.2 *Distributed Hardware Accelerator Architecture*

Although the power efficiency (i.e., the amount of work per unit of power) of hardware accelerator is high compared to software-based solutions, it is possible to

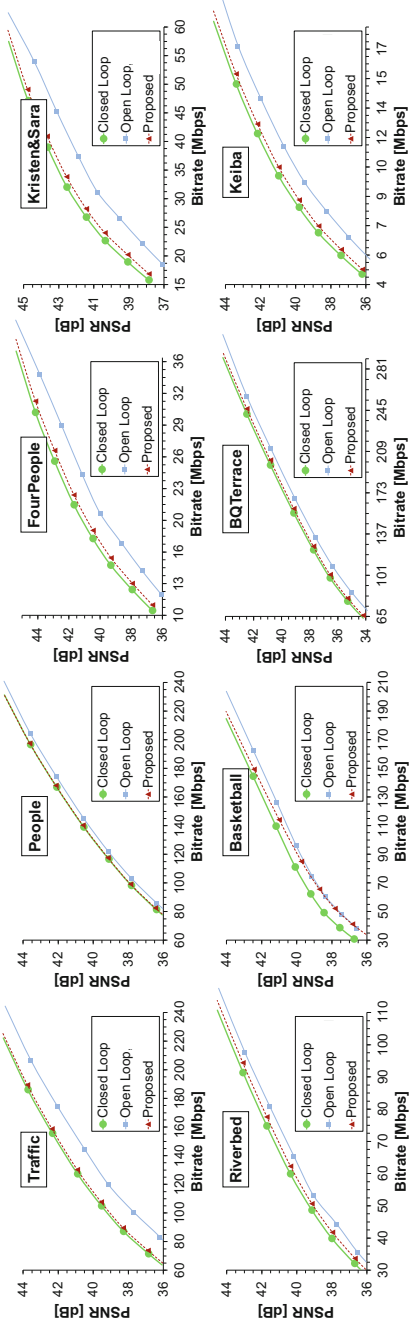


Fig. 5.19 RD curves (PSNR vs. Bitrate plots) for presented and open loop techniques with $\theta = 1$; each plot represents the average results for QP sweeps from 18 to 32 (step size = 2)

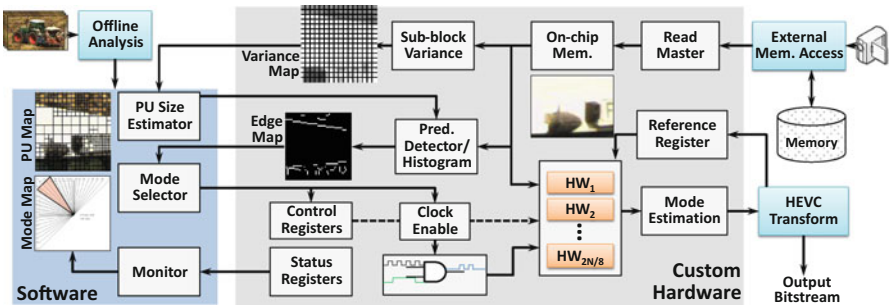


Fig. 5.20 Hardware-software collaborative control for complexity and power reduction of HEVC intra-encoding

additionally decrease the power consumption of the hardware accelerator. As discussed in Chap. 2 of this book, HEVC uses blocks of sizes ranging from 64×64 down to 4×4. Using accelerator architectures for all these block sizes is inefficient. For example, generating intra-prediction in hardware would thus require implementing intra-prediction circuitry for each of the PU sizes. However, this will incur large area overhead and a power penalty. This book presents a technique to distribute the hardware of the largest intra-prediction generating unit in HEVC into smaller components and clock- or power-gate individual components of the large hardware accelerator. This way, only a selected few constituent components of the hardware accelerator will be consuming power, while the rest would be normally turned off. The concept of distributed and power-gated hardware accelerator design can be extended to other applications.

The distributed hardware accelerator architecture is outlined in Fig. 5.20. This architecture uses a hardware-software collaborative complexity and power reduction technique for HEVC intra-encoding. Following the standard hardware-software partitioning trends, high complexity number-crunching jobs with minimal conditional executions are processed via hardware accelerators, whereas low complexity jobs mainly consisting of control decisions occur at the software layer. A block of pixels (e.g., a CTU) is fetched from the external memory for processing. To reduce the complexity, edge histogram is generated at the hardware layer, whereas the software layer uses this information to determine the most probable intra-prediction mode (Sect. 4.3.3.1). Similarly, the variance of all 4×4 sub-blocks within the CTU is computed in hardware, whereas the PU maps (PUM and PUMA, Sect. 4.3.3.2) used for reducing HEVC complexity are generated in the software.

Intra-prediction Generator The prediction generation is carried out using a distributed hardware accelerator. The prediction generation hardware can generate a prediction of the largest possible PU (i.e., 64×64) in a single cycle. However, note that a single large PU prediction generator is not employed in this architecture. Instead, PU prediction generation for large PU sizes is achieved by concatenating the output of eight, individual 8-sample prediction generators. The value of 8 samples per prediction generator is chosen because our analysis in Table 5.2 of various

Table 5.2 Percentage occurrences of PU sizes in HEVC

	RaceHorsesC		BQSquare		FourPeople	
PU Size	*QP = 22*	*QP = 37*	*QP = 22*	*QP=37*	*QP=22*	*QP=37*
4	30.064	10.131	61.623	39.344	22.75	7.793
8	54.463	64.085	35.967	50.87	56.727	58.71
16	13.168	22.523	2.386	8.802	17.295	26.749
32	2.257	3.217	0.023	0.985	3.126	6.155
64	0.049	0.043	0	0	0.102	0.539

Table 5.3 Hardware footprint for a 64×64 CTU (1 PLL, ~205K ALUTs, ~205K Registers, 736 DSP modules)

HW module	Freq (MHz)	ALUTs and registers	Memory (bits)	DSP modules	Logic Util. [%]
Variance	167.14	253,386	0	7 (<1%)	<1%
Sobel histogram	243.72	77, 86	0	1 (<1%)	<1%
Intra pred. block	243.61	203, 377	0	8 (1%)	<1%
Total CI	162.79	13409, 6934	43008	72 (10%)	10%

video sequences reveals that 8 × 8 PU is the highest occurring PU size, even with varying motion and texture properties. Moreover, note that it is impossible to have a PU not at the 8 × 8 boundary because the minimum CU size in HEVC is 8 × 8.

Clock-Gating Logic As discussed above, eight pixels of a full CTU row are associated with a single component of the intra-prediction generation hardware. Thus, when a smaller PU is processed, output from some of the intra-prediction generators is not required. Therefore, these modules can be clock-gated to save energy. The clock-gating circuit is controlled by the control register, whose individual bits are the set by the software, depending upon the PU size and location of the PU within the CTU.

5.3.2.1 Energy and Resource Evaluation

For the architecture outlined in Fig. 5.20, the area and frequency of individual modules are tabulated in Table 5.3. The Altera FPGA (EP2AGX260FF3513) used for evaluations is a midrange FPGA. Hence, by using a complete custom design, it is expected that an ASIC can further improve the throughput and area savings.

In Fig. 5.21, the energy consumption comparison between (a) no clock gating, single prediction hardware and (b) presented architecture for one frame is performed (with energy saving percentage written on top of the bars). The stimuli data generated by the HEVC reference software and ModelSim [24] is provided to the Altera's Powerplay Power Analyzer tool for determining signals' static

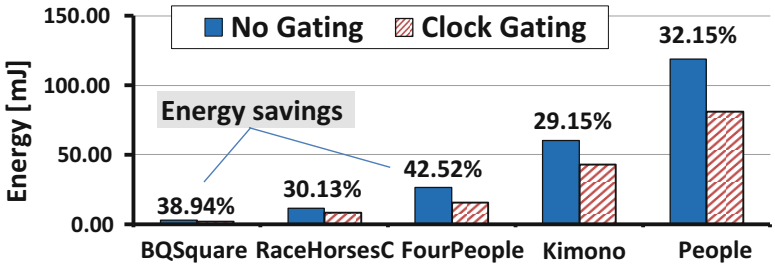


Fig. 5.21 Average energy consumption of a single frame, with and without clock gating

probabilities and transition densities, and the energy numbers are reported here. Using the architecture mentioned above, up to 42% energy savings can be achieved.

5.4 Hybrid Video Memory Architectures

An integral architectural focus of any hardware accelerator-based design is power- and compute-efficient implementation of the memory subsystem. As discussed in Chaps. 2 and 3 of this book, the high access rates of external memories and high leakage power of on-chip video memories (e.g., by block-matching algorithm) can increase the energy consumption of the video processing system considerably. Therefore, one can employ a hybrid memory architecture to exploit the advantages offered by both volatile and nonvolatile memories and reduce the energy consumption. Basic idea is to push the video data which will be read more often (i.e., will remain valid in memory for a long duration) into NVM and thus save leakage power. The opposite is true for VM, whereby the data structures overwritten considerably frequently should be placed in the memory with low write-energy (i.e., SRAM). Similarly, fast block-matching algorithms usually follow a fixed pattern (e.g., TZ search implemented in the HEVC reference software). This pattern of memory access can be used to determine the highly likely area of the memory where the next predictor under test should lie. Thus, turning *off* un-accessed memory regions also saves leakage energy.

To realize the above, an adaptive energy management for on-chip hybrid video memories (AMBER) is presented here. It is a hybrid memory architecture with integrated energy management for video applications. The architecture comprises of hardware accelerators for fetching video data from the external memory using SRAM and storing it in the on-chip high-capacity NVM memory. The inbuilt adaptive energy management technique power gates specific NVM memory sectors to save energy. The basic idea of AMBER is to leverage the characteristics of different memory types (read/write latencies, leakage energy, etc.) and the application specific knowledge to reduce the total energy consumption and processing latency of a video system. Moreover, AMBER can be seamlessly combined with

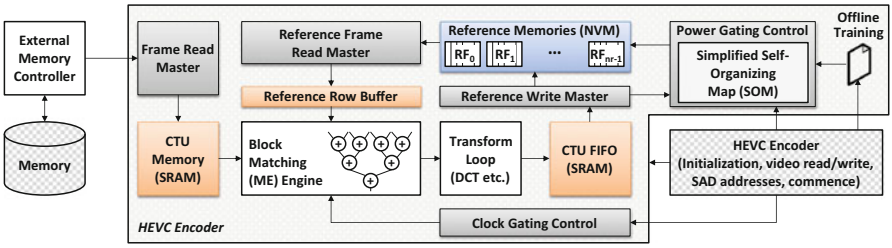


Fig. 5.22 AMBER memory subsystem architecture employed in HEVC video encoder

other orthogonal power reduction techniques for video processing systems. In a nutshell, AMBER targets:

- *Design of on-chip, hierarchical video memory subsystem*, using hybrid memories while exploiting the advantages offered by each memory type, especially for video applications.
- *Runtime adaptive power gating of selective memory sectors* to reduce leakage energy consumption of the system, by using an adaptive energy manager that exploits the video application- and video content-specific properties.

As a proof of concept, AMBER methodology is used for energy management of the memory subsystem of an HEVC encoder. In the following, the details about the memory subsystem architecture will be followed by the details of controlling the power consumption of the video processing system.

The memory energy management architecture for HEVC hybrid memories is shown in Fig. 5.22. The off-chip or external memory holds only the current video frame data. Using the external memory controller and memory managing hardware accelerators, this raw video data is brought to the on-chip SRAM CTU buffer. This buffer is used by the ME engine where the block-matching algorithm takes place. The block-matching process is controlled by the HEVC controller. After block matching, best inter-predictor is forwarded to the HEVC encoder, which generates the reconstructed CTU, used as a reference for the next frames. The reconstructed CTU is pushed into the SRAM CTU FIFO. This FIFO is read by the MRAM reference frame memories. These memories are power gated by the power-gate control module. In the following, we discuss the basic components of AMBER system in detail.

5.4.1 AMBER Memory Hierarchy

As illustrated in Fig. 5.22, in the AMBER framework, external memory holds only the current video frame. Since high-density and low read/write latency is essential for the external memory, DRAM is a suitable candidate for the external memory. It

can be replaced by a PRAM, but writing this memory by the raw video stream from the camera is both energy- and latency-wise expensive.

The read master accelerator reads the current raw video data from the off-chip memory, one CTU at a time, and places it in the on-chip CTU memory of size $b_w \times b_h$ samples, with b_w, $b_h \in \{16, 32, 64\}$ and each sample of size 8 bits. Due to a small amount of data (maximum 64×64 pixels) being written to this memory from the external DRAM, the CTU is stored in an SRAM. Thus, motion estimation can start as soon as data is available. The output of motion estimation is processed and fed to the reference frame buffers via a SRAM FIFO, also of size $b_w \times b_h$. Since SRAMs have the fastest read/write characteristics, the SRAM memories employed in AMBER hide the latencies from the external bus system and the HEVC processing engine.

The reference write master pulls the data from the SRAM FIFO and feeds it to the on-chip NVM reference frame memories. These memories hold either n_r reference frames (in case of multi-frame prediction) or a single large frame (e.g., 8K UHD, 7680×4352) in n_r separate video frame memories. The reference read master reads the predictors from these memories and forwards it to the motion estimation engine.

5.4.2 NVM Reference Memory Architecture

The NVM reference frame memories consist of memory banks and sectors. Each bank is reserved for one reference frame of size $w \times h$. Each bank is divided into multiple sectors where individual sectors are NVMs of size $b_w \times h$ and can be independently power gated using the power-gate control. These NVMs are "normally off" and only awakened for reading or writing. As discussed, the reference data generated by the HEVC encoder is fed to the SRAM FIFO. A reference write master module, with an internal AGU, writes this data to the appropriate bank and sector. The write master also collaborates with the power-gate control logic. Furthermore, all the sectors use only one address and data port that can be used for reading and writing. Moreover, the data ports of individual banks can be accessed in parallel, thus, reducing the read/write latency.

During encoding, the motion estimation engine requests a predictor from the reference frame. Its request is handled by the reference read master module. For this purpose, the motion estimation engine provides row and column addresses within the reference frame from which to read the predictor. Read master properly translates these addresses to banks and sectors and cross-checks if the sector is turned *on*. If not, then a turn *on* request to the power-gate control unit is generated. Obviously, this results in latency. Afterwards, a row of the prediction is written to the row buffer which forwards this data to the motion estimation engine. Note that the concept of search window is now replaced by the full video frame memory, and there is no need to fill a memory structure repeatedly with raw video samples (see

Fig. 2.9). Thus, in the following, a search window in the context of AMBER will mean a region in the on-chip frame buffer.

In the following, we will discuss some constraints and characteristics of using AMBER architecture within HEVC. For AMBER, the write latency of an NVM sector should be lesser than or equal to the average motion estimation computational time. Thus, for $w \times h$ dimensions of frames and f_p frames per second, the write latency of a CTU to the NVM, $t_{w,m,CTU}$, must hold the following condition:

$$t_{w,m,CTU} \leq (b_w \times b_h)/(w \times h \times f_p) \tag{5.16}$$

Further, the dynamic power in the AMBER architecture is the power consumed by:

- Two times writing into CTU SRAM
- Two times reading from CTU SRAM
- Writing to NVM frame memory
- Reading from NVM frame memory

Therefore, we can write the total dynamic power p_{dyn} as:

$$p_{\text{dyn}} = w \times h \times f_p \times \left(e_{\text{dyn},w,m} + 2e_{\text{dyn},w,s}\right) \tag{5.17}$$

Here, $e_{\text{dyn},w,m}$ and $e_{\text{dyn},w,s}$ are the dynamic write energies of NVM and SRAM, respectively. The total leakage power p_{leak} is then:

$$p_{\text{leak}} = p_{\text{leak},m} \times (b_w \times h + s_w \times h) + 2p_{\text{leak},s} \times b_w \times b_h \tag{5.18}$$

Here, $p_{leak,m}$ and $p_{leak,s}$ are the leakage powers of NVM and SRAM, respectively. While the reason for SRAM leakage power term is obvious, for the NVM, it is more involved. During write phase, only one sector is turned *on* and during read phase, a set of sectors is turned *on*, denoted by s_w, the width of the search window for block matching.

Now, we compare the advantages of on-chip NVM reference memory with the usual search window-based external memory with prefetching technique. With low leakage and high capacity, NVM on-chip video memories are feasible. For external memory-based architectures with $n_r$reference frames and search window read factor r_f, the total accesses (reads and writes) directly to the external memory results in the total pixels accessed equal to:

$$w \times h \times 8 \times f_p \times r_f \times n_r + 2 \times w \times h \times 8 \times f_p \tag{5.19}$$

Here, the second term denotes the reading of the luminance frame from the external memory and then writing back the reconstructed frame. On the contrary, if NVM on-chip reference frame memories are employed, total number of external memory accesses is reduced by the factor $2 + r_f n_r$, which is over three times external memory access savings. Moreover, on-chip memory latency is much

lower compared to the off-chip latency. Furthermore, the contention on the off-chip external memory bus is reduced using NVM frame memories.

Again, consider that n_r reference frames are stored in the external memory. The total leakage power can be therefore written as:

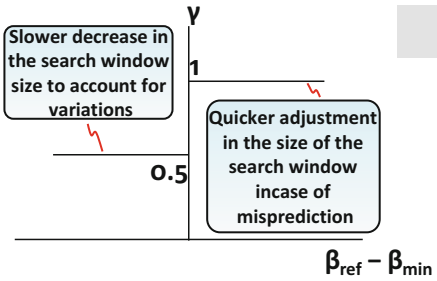
$$p_{\text{leak}} = p_{\text{leak},d} \times w \times h \times n_r \tag{5.20}$$

Here, $p_{leak,d}$ is the leakage power of external memory (e.g., DRAM). On the other hand, the NVM memories can be switched *off* to dissipate no leakage (see the leakage of NVM reference memories in Eq. (5.18)). In fact, AMBER will only turn *on* set of sectors at a time, depending upon the predictor location. Thus, the total leakage power is reduced.

Power gating is only advantageous for NVM reference memories and not for VM or on-chip SRAM. External VMs like DRAM, and on-chip VMs like SRAM, will lose their contents as soon as the power is cut off. Moreover, once the design of ME engine is fixed (chip is fabricated), the size of the search window cannot be altered. A scenario for which even smaller search windows are acceptable (e.g., low resolution, low motion video coding) employing DRAMs or SRAMs reference memories will still consume the same leakage power. However, this is not the case with the NVM-based memory subsystem. Due to the non-volatility of the NVM, reference frames will still be available, once the NVM is turned back *on*. Thus, AMBER adapts the leakage power consumption at runtime to save energy.

5.4.3 Energy Management of NVM Reference Memories

In the "sliding" search window-based video block-matching architecture, note that whether or not the current video block accesses all the samples in the search window, the next block may use these samples. Therefore, search window samples cannot be discarded. Moreover, from Fig. 3.10b, we notice that most of the search window is wasted because some of the video samples are never accessed. Considering this, block-matching algorithm accesses only a small percentage of sectors in the hybrid memory architecture presented here. However, since on-chip full-frame buffers are employed, a search window can actually be considered as the full frame. Therefore, multiple writes to this search memory are not required compared to the search window-based technique, saving dynamic write energy. On the other hand, since the search window in AMBER case is very large, it has a high leakage power. It is presented in the previous section that the leakage energy of AMBER can be reduced by appropriately power gating the unused memory sectors. For a particular bank, the power-gating control unit turns *on* set of sectors between β_1 and β_2 (collectively represented as (β_1, β_2)) and power gates the rest. The values (β_1, β_2) are predicted and adapted by analyzing the memory access pattern of the block-matching algorithm. Power-gate control unit actually controls the sleep transistor associated with a memory sector.



Neuron	Weight
1	(β_{11}, β_{12})
2	(β_{21}, β_{22})
3	(β_{31}, β_{32})
⋮	⋮

Fig. 5.23 Illustration of neuron weight-update feedback and SOM table

5.4.3.1 Memory Access-Based Self-Organizing Map

The estimation of (β_1, β_2) is computed by a self-organizing map (SOM) [25], which keeps a record of the memory access pattern and updates its map whenever there is a change in the pattern search procedure.

Fast block-matching algorithms for a block usually follow a pattern. Therefore, some simplifications can be made for designing the SOM. A table with neurons as the keys can represent the SOM. This is shown in Fig. 5.23. Further, training the SOM is fast, as we do not need to test each input against every neuron, rather, the neurons can be updated in a sequence. Each neuron of the SOM holds the minimum and maximum sector turned *on* for the current block in the form of a tuple (β_{min}, β_{max}). This tuple is called the weight of the neuron. Based upon the weight of the neuron, the power-gating control bits are set or reset.

For ease of readability, we will consider motion estimation for HEVC in this section. Each CU within the CTU will undergo the motion estimation process. Obviously, the discussion is valid for video encoders. The power-gate unit powers *on* the sectors between:

$$\beta_1 = CU_x - \beta_{\min}; \quad \beta_2 = CU_x + \beta_{\max} \tag{5.21}$$

Here, CU_x is the horizontal location of the top-left pixel of CU (or PU) under test. That is, only the sectors between β_1 and β_2 are turned *on*. Additionally, note that there is a neuron in the SOM table for every CU of the CTU. From Eq. (2.7), we notice that a total of 85 neurons are required for a CTU of 64 × 64. The weight of the neuron is set during the training phase of the SOM. However, it is also updated at runtime if there is a change in the memory access pattern. Suppose that the block-matching algorithm requests a memory sector β_{ref} to be turned *on*. Then the weight update formulas are given by:

$$\begin{aligned} \beta_{\min} &= \lfloor \beta_{\min} + \gamma(\beta_{\text{ref}} - \beta_{\min}) \rfloor \\ \beta_{\max} &= \lceil \beta_{\max} + \gamma(\beta_{\text{ref}} + b_w - \beta_{\max}) \rceil \end{aligned} \tag{5.22}$$

The learning coefficient (γ) takes the value of 1 or 0.5 for simplicity in this implementation, as shown in Fig. 5.23. If the search window size is increasing, faster expansion of the search window is allowed, by keeping $\gamma = 1$.For reducing the search window size (i.e., turning *off* some sectors), AMBER takes $\gamma = 0.5$. In the formula above, b_w is the width of the CU under test.

Typically, the memory access pattern by the block-matching algorithm does not change during the HEVC encoding. Therefore, the offline-trained SOM does not change during the runtime for a given motion estimator. However, changing the memory access pattern will require the neurons to adjust first and latency will be encountered. Moreover, a threshold-based ME early termination technique is usually employed for production systems, in which case block matching for the current CU terminates if SAD is less than a particular threshold. Even with this technique, AMBER still outperforms the search window architecture because search window will always require prefetching new CTU column from the external memory on every CTU iteration. Moreover, for slow-varying motion sequences with adaptive motion estimation, the number of NVM sectors turned *on* will reduce over time, hence further reduction in the leakage power. It will be reverse for high-varying motion sequences.

5.4.4 System Computation Flow

Algorithm 10 shows the high-level algorithm of AMBER for HEVC. For the current CTU under test, we wait before it is fully written to the CTU SRAM and the banks are ready for reading (line 2). Afterwards, the motion estimation (ME) for the current CTU commences (line 4). Before the ME launches, we forecast the bits of power-gate control lines (lines 7 and 8) given by Power Gate Control Register PG. PG bits are set to turn *on* the specific NVM memory sectors. Excess memories turned *on* due to previous CU computations are also turned *off* (line 9). Once the block matching for the CU starts, it is tested whether correct NVM sectors are turned *on* (lines 14 and 15). In case a misprediction in the forecasting the right sectors occurs, we need to turn *on* the NVM sectors which are erroneously kept *off* (line 16). In parallel, we update the estimator (line 17 and Eq. (5.22)). Afterwards, ME can start (line 19) and we repeat the process to the next location in the block-matching pattern for the current CU. Once the current CU is processed, it is subdivided into four equal CUs (lines 21 and 22) and the process is repeated.

We now examine the impact of the wrong decision latency on the performance of HEVC. For a frame with total n_{frm}CTUs, the total number of CUs searched is (see Eq. (2.7)):

$$\xi = n_{\text{frm}} \times 13 \times \sum_{i=0}^{\log_2 bw-3} 2^{2i} \tag{5.23}$$

Therefore, for a 30 FPS video, total number of CUs searched in 1 s is equal to 30ξ. Now consider that we turn *off* memory forecasting and the wake-up time for a NVM is supposedly ψ cycles, a total of $30\xi\psi$ cycles per second are wasted. As an example, for a CTU of size 64×64 of a full-HD frame (1920×1088), it will incur an additional latency of about 16.9ψ megacycles per second. Therefore, guessing the correct location of the predictor is important in managing the energy consumption of the prediction process.

5.5 Energy-Efficient Anti-aging Circuits for SRAM

In the previous section, NVM-based memories are considered to store the raw video data. However, SRAMs are the most common type of on-chip memories, which are universally available on embedded systems. By only using SRAMs to store video data, system design can achieve low latency and eliminate the extra circuitry/maintenance required by the NVM-based hybrid memories. However, using SRAM comes with additional challenges, and one of the challenges is time-based degradation (called aging) of SRAM cells.

This book targets aging analysis and configurable aging optimization of SRAM-based on-chip memories deployed in application-specific architectures (like camera-based video processing systems). In this section, details about the aging-resilient architecture for a memory composed of 6T-SRAM-based cells are presented. In order to reduce the aging rate of the SRAM video memories (see Sects. 2.3.2 and 3.3.3), microarchitectural enhancements are employed at the memory subsystem, which modulates or transduces the video data written to and read from the SRAM memory. Some of these circuits are also discussed in Chap. 3. The SRAM memory is assumed to have capacity sufficient to store a large chunk of data, e.g., multiple images/video frames. Figure 5.24 illustrates the overall architecture of the aging-resilient SRAM memory. The operational workflow of writing and reading video data is as follows:

- Raw video samples from the streaming video source are written to a FIFO, controlled by the FIFO controller, which also provides appropriate data-valid signals to the memory subsystem. This FIFO and its associated control circuitry are similar to the ones previously discussed in this book.
- The aging controller snoops the data from the data FIFO and generates proper control signals for the best aging resilience and power trade-off (see details in Sect. 5.5.3).
- The signals generated by the aging control configure the Memory Write Transducer (MWT) to adapt input video samples before they are written to the memory (see details in Sect. 5.5.1). Specific bits of the samples are selected for inversion by setting the appropriate enable signals of the configurable Inverter Switches. In addition, the write address of the data is changed at runtime using the Write AGU (see details in Sect. 5.5.2), to fully utilize the memory

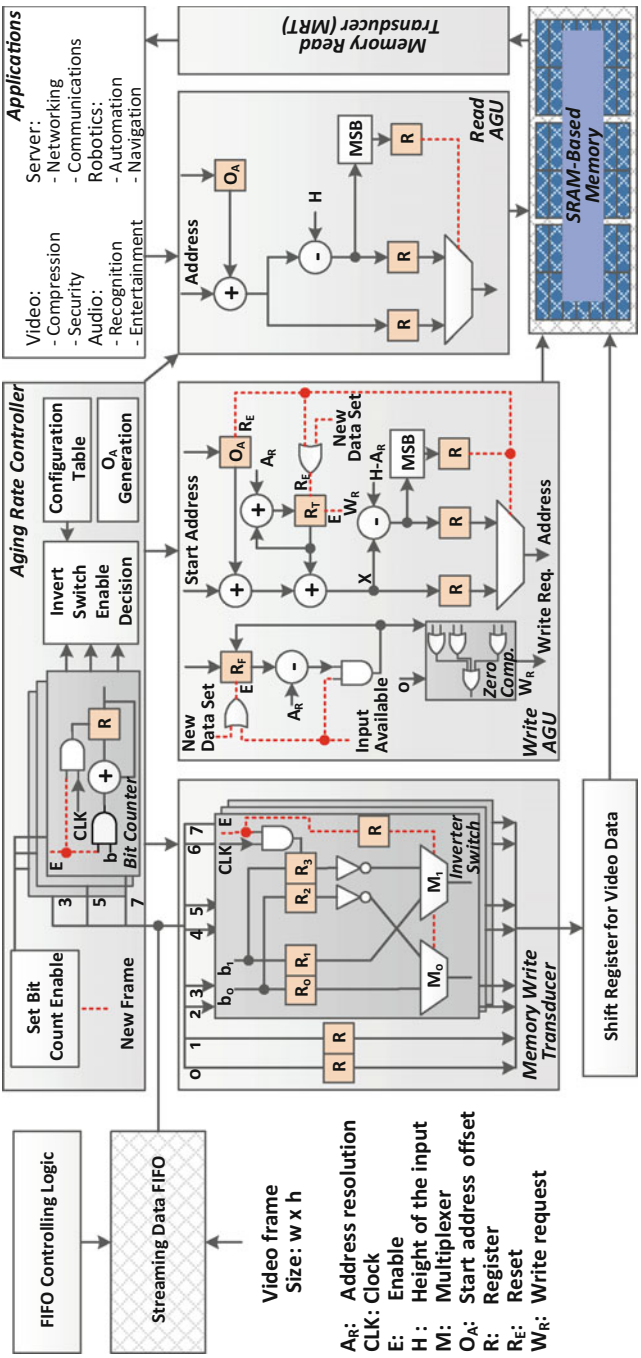


Fig. 5.24 SRAM memory subsystem anti-aging architecture. The Memory Write Transducer, Write AGU, Read AGU, and Memory Read Transducer are connected to the SRAM. The aging controller configures these modules by generating control signals to adapt input data at read and write ports of the memory

address space and introduce stress relaxation at the SRAM cells holding less frequently changing data samples (i.e., pixels of static background regions in an image).
- Before the video samples are read by the application, the Read AGU appropriately converts the logical address to the physical address, and the Memory Read Transducer (MRT, an exact replica of MWT) readapts them.

As pointed out, the configurable design presented here explores the trade-off between aging resilience and the associated power overhead. Further, the configurable aging resiliency techniques presented here are scalable, because they are orthogonal to the number of memory ports and memory size. In the following, the memory write modules are discussed in detail. However, note that the working principles of the memory read modules can be easily deduced as they perform the exact opposite operation of the writing modules.

5.5.1 Memory Write Transducer (MWT)

The MWT is used to invert specific bits of the raw video samples in order to toggle less frequently changing samples and release stress on the 6T-SRAM cells storing these bits. The data bits are grouped in pairs, and each bit-pair can be adapted separately to target the specific SRAM cells for releasing stress. For this purpose, the higher-order data bit-pairs (bits 2-7) are passed through controlled Inverter Switches. Figure 5.24 shows that the first two bits (0 and 1) are not adapted. This is due to the fact that the SRAM cells storing the first two LSB bits always have the lowest degree of stress, due to high degree of variation and hence a balanced duty cycle (as shown in the case study of Fig. 3.13j). Therefore, the anti-aging architecture presented here only uses three Inverter Switches to control six MSB bits of a data sample. The input control lines (E) act as a clock-gating signal to the registers R_2 and R_3 and as a "select" signal to the multiplexers M_0 and M_1. All the registers store the original unmodified bits (b_0 and b_1). The registers R_0 and R_1 are directly connected to the multiplexers, whereas R_2 and R_3 are inverted and connected to the multiplexers. As seen, for every bit-pair, five 1-bit registers, two inverters, and two 1-bit multiplexers are required. For example, for 8-bit data samples, a total of 15 1-bit registers, 6 inverters, and 6 1-bit multiplexers are required.

If the control signal E of an invert switch is set (high), registers R_2 and R_3 latch the bits b_0 and b_1, and thus, the inputs and outputs of the inverters are updated. In this case, both input bits are inverted and the inverted bits are generated at the output. If the "enable" signal is reset (low), bits b_0 and b_1 are forwarded to the shift register unaltered. Therefore, no dynamic energy is consumed by R_2 and R_3 and the inverters. The signal E is controlled by the aging controller.

5.5.2 *Aging-Aware Address Generation Unit (AGU)*

The Write AGU is responsible for selecting the appropriate frame memory (e.g., for writing an incoming video frame from a camera device) and generating addresses for writing the data words to the SRAM memory from the shift register. Selecting the appropriate frame memory for writing a complete video frame follows a round-robin technique (see the ring buffer discussed in Sect. 5.1.3). In addition, before overwriting a frame memory, it is required that frame memory is no more required by the executing application(s). An application signals this by requesting the address of a new data set (see Sect. 3.3.3 for details). A signal ("New data set request" in Fig. 5.24) prompts to deliver a new starting address to the Write AGU.

As discussed before, less frequently changing data will incur the most NBTI-induced stress on the 6T-SRAM cells. If the most frequently changed data words are identified, they can be distributed in memory in a spatial round-robin fashion. However, this may require information from the application and additional analysis at runtime, which is power- and area-wise inefficient. Therefore, a simple technique is introduced for aging resiliency. For every new video frame written to the memory partition, the video frame is circularly shifted. This corresponds to changing the starting address of the data set. With each new video frame, an offset in the starting address (o_A) is generated (by the aging controller) to the Write AGU as the starting address of the video frame. With every frame, o_A is accumulated and the starting address is shifted by o_A. This guarantees that the frame memory containing less frequently changing data is interchanged with the more frequently changing data at regular intervals, thus relieving stress from the SRAM cells holding bits of the less frequently changing data. Figure 5.25 illustrates the example of video frame writing in the memory with $o_A = 119$.

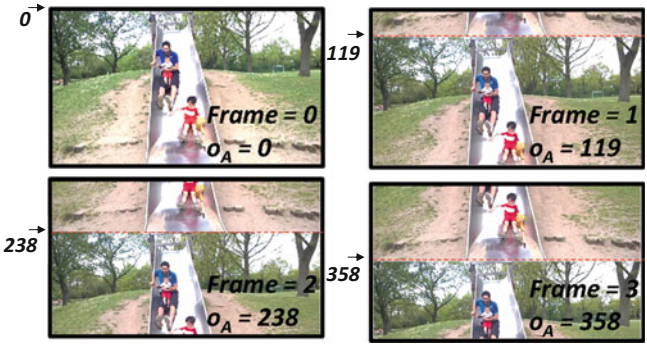


Fig. 5.25 Write AGU frame writing technique with $o_A = 119$. The moving region (sliding people) is overwriting the static background with every frame

5.5.3 Aging Controller

As illustrated in Fig. 5.24, the aging controller constructs the control signals of the MWT and supplies the start addresses of the video frame in the Write AGU. Figure 5.26 shows the comprehensive flow of the aging controller for enabling Inverter Switches. For every Inverter Switch, such a controller exists; therefore, three such controllers are used in the SRAM anti-aging system. To generate the control signals for MWT, two decisions must be made: (1) at what time instant the Inverter Switch circuit should be activated and (2) on which SRAM cells aging balancing should be applied.

Decision – 1: Triggering Aging Balancing Circuit at a Particular Time Instant For this, the Inverter Switches are actuated for a complete video frame after specific time period, i.e., a certain number of video frames are written without adaptation and are processed by the application. For example, consider that a specific bit-plane of a frame $2i$ is stored without inversion ($E = 0$) and for the frame $2i+1$, it is stored with its bits inverted ($E = 1$). This corresponds to the data adaptation rate (f_R) of 1. That is, the respective bit-planes in every second video frame are inverted. Formally, f_R denotes the number of video frames stored in the SRAM memory without inversion of the video frame. In the definition above, a bit-plane is defined as the collection of the bits at the same bit location, in all the samples of a video frame. In case two neighboring video frames have high correlation, it is expected that the inversion of frame $2i + 1$ will overwrite most of the bit locations of frame $2i$ with inverted bits. This will therefore relieve stress on the SRAM cells and reduce the NBTI introduced aging. Note that there is a separate f_R for each Inverter Switch, and each Inverter Switch is independently activated. Additionally, f_R also plays an important role in deciding about the power

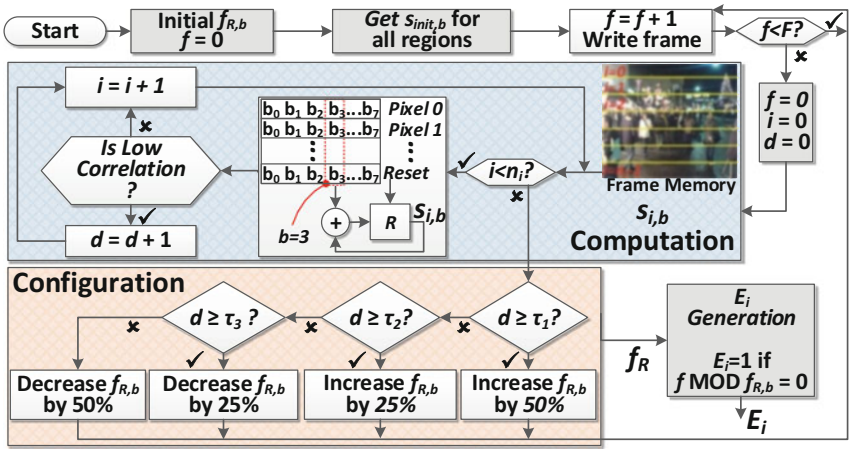


Fig. 5.26 Inverter Switch enabling decision logic. For the anti-aging memory architecture discussed here, there are three such circuits, one for each Invert Switch

consumption and aging rate of the SRAM memory. A high f_R will reduce the power consumption but will be less resilient to aging. On the contrary, a low f_R will reduce the aging rate, but will also increase the power penalty.

Decision – 2: Picking SRAM Cells for Aging Balancing At runtime, the MWT enable signals (E, which is generated based upon f_R) can be turned *on* or *off* depending upon the expected aging and the power constraint of the system. When all the enable signals are inactive ($N = 0$), power penalty of this technique is the lowest because no inversion takes place. This relates to no toggling activity occurs at the inputs of the inverters of MWT because the registers associated with the inverters are unchanged. However, the rate of SRAM aging is the highest because next f_R number of video frames are written without adaptation and thus, increase the amount of stress a single transistor of a 6T cell will endure. Similarly, when all the enable signals are active ($N = 3$), SRAM aging rate is the lowest at the cost of the highest power penalty. Basically, when only one enable signal must be active ($N = 1$), the SRAM anti-aging architecture inverts the two MSB bits (bit 6 and 7) as the SRAM cells storing these bit encounter the most stress. If the power configuration allows for two enable signals to be active (i.e., $N = 2$), the four MSB bits (bits 4 till 7) are inverted.

A method can determine the value of N depending upon f_R of all Inverter Switches. If f_R of all Inverter Switches is selected such that adaptation is active for all Inverter Switches for the same video frame, then $N = 3$. Otherwise, if two Inverter Switches are active for the same frame, then $N = 2$. Selection of appropriate f_R is a control problem, which must be computed at runtime by analyzing the characteristics of the input data (i.e., computation of duty cycle for different bits). In the technique presented here, a simple and efficient microarchitecture to estimate duty cycle online is given, as outlined in Fig. 5.26. This circuit is used to determine f_R for a single bit-plane. A bit-plane is divided into n_i parts, and the corresponding bits are accumulated for each part. This accumulation also denotes the duty cycle of the bit-plane in the specific frame part. For every part, if the number of 1s differs significantly than its previous stored value (depending upon the lower $s_{L,i,b}$ and upper $s_{H,i,b}$threshold), the part has a different texture, and a difference counter d is incremented. Afterwards, d is tested and f_R of the bit-plane is increased or decreased. The thresholds τ_1, τ_2, and τ_3 ($\tau_1 \geq \tau_2 \geq \tau_3$) determine the change in f_R and can be set at design time.

However, calculation of duty cycle online incurs a power penalty. In the technique presented here, a counter-based logic is used to activate the duty cycle generation circuit. The set bit counter's register and the input bit to the adder are only updated if the enable signal is high. Thus, this will save dynamic power consumption. If online duty cycle computations are more frequent (finer control with smaller F), the power consumption of this technique will be high and vice versa.

5.5.4 *Generalization and Applicability*

In general, the aging-mitigation design methodology and architecture presented here are orthogonal to the type of application and the low-level aging models. In fact, a video frame can be replaced by any data set in the previous discussion. However, this may require several design optimizations to get the best power-efficient aging-mitigation design, for example:

- In addition to inverter, some rotation or swapping modules can also be combined in the Memory Read and Write Transducers.
- The values of controller variables to adapt the aging balancing might need to be reevaluated.

This will thus require an application- and content-aware analysis, similar to the one performed for the camera-based application in this book. To illustrate the varying duty cycle behavior of a non-video-based application, an audio sample storage is evaluated, as discussed below.

Figure 5.27 shows a 16 KHz, 16-bit linear pulse code modulated (LPCM) audio signal and the duty-cycle box plot of the memory used for storing this signal. Without using any aging balancing circuit, the duty cycle of all the bits is already balanced, which is visible from the concentrated spread of the box plot. This is due to the fact that audio signal swings around 0, and the number of negatives (with MSB bits storing 1s) and the number of positives (with MSB bits storing 0s) are similar, unlike the video data. Moreover, the temporal correlation of audio data is low, suggesting that it is highly likely that the new audio sample, which overwrites the previous one, will have different characteristics (i.e., different sign). However, this is the not the case with video data, which will have high temporal correlation (e.g., the background region, which will be static in many consecutive video frames), leading to biased duty cycles and, therefore, higher stress on the 6T SRAM cells.

Moreover, the analysis carried out in this book only considered integer values. An interested reader can perform similar analysis for floating-point storage. Large floating-point storages are becoming increasingly popular due to large matrix operations in graph theory, quantum mechanics, machine learning etc.

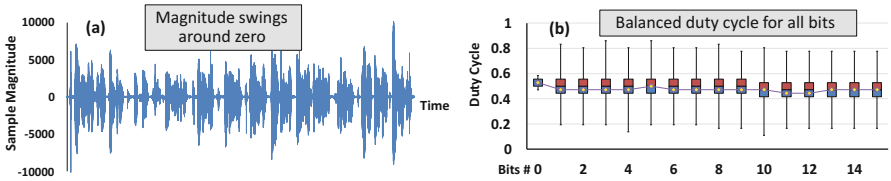


Fig. 5.27 (**a**) Magnitude swings of a 16-bit per sample, 16 KHz audio signal, and (**b**) aging profile as box plot

5.5.5 Sensitivity Analysis of SRAM Anti-aging Circuits

The information on the duty cycles can be transformed to respective SNM over time using any available data mapping of duty cycle (Δ) to SNM degradation. However, the technique presented in this book is independent of such a mapping. In fact, any table that relates the Δ to SNM degradation can be used, because a balanced Δ will always result in smaller SNM degradation (lower aging rate). For evaluation purposes, another variable called aging map (δ) is defined, which connects the Δ ranges to respective δs as shown in Table 5.4. As seen, if the duty cycle is balanced, ($0.4 \leq \Delta \leq 0.6$), the value of δ is low. This should be the goal of an aging balancing approach – to reduce δ as much as possible. Moreover, all the 6T SRAM cells with Δ more than 0.95 or less than 0.05 result in high SNM degradation and therefore are represented by a higher δ value. To concisely present the duty cycle of the whole frame memory, δ for the complete frame memory can be represented in form of a δ histogram. Afterwards, the information contained in this histogram can be presented in form of a scalar metric π given as:

$$\pi = \frac{\sum_{\forall \text{bins}} \left[\left(v_{\text{bin}} - v_{\text{bin,min}} \right) \times n_{\text{bin}} \right]}{\left(v_{\text{bin,max}} - v_{\text{bin,min}} \right) \times n_{\text{samples}}} \tag{5.24}$$

In this equation, v_{bin} is the value of a bin of the histogram, on x-axis of histogram (δ). n_{bin} is the number of values in the bin, on y-axis of histogram (total number of 6T SRAM cells which are stressed under the particular δ). $v_{bin,\text{min}}$ and $v_{bin,\text{max}}$ are the minimum and maximum values of all the bins, respectively. n_{samples} corresponds to the total samples in the histogram. Thus, for SRAM storing an 8-bit per pixel video frame, $n_{samples}$ equals $w \times h \times 8$, i.e., the size of the frame memory. If the number of values in the bins is closer to $v_{bin,\text{min}}$ ($\delta = 0$ in this case), we would expect π closer to 0. Otherwise, π will have a larger value. Therefore, an aging resiliency technique should strive to reduce π as much as possible. Further, calculation of π considers the distance between the local degradation and the least possible degradation (i.e., $v_{bin} - v_{bin,min}$).

The aging analysis of SRAM cells for different video sequences is tabulated in Table 5.5. The column "Base" denotes the amount of aging without using any MWT, i.e., without applying any aging balancing techniques. Since different video sequences lead to different amount of stress as a result of varying distribution of

Table 5.4 Duty cycle (Δ) to aging map

Duty Cycle (Δ)	Aging map ($\delta = \|0.5-\Delta\|$)
$\Delta \leq 0.05$ or $\Delta \geq 0.95$	0.45
$\Delta \leq 0.1$ or $\Delta \geq 0.9$	0.4
$\Delta \leq 0.2$ or $\Delta \geq 0.8$	0.3
$\Delta \leq 0.3$ or $\Delta \geq 0.7$	0.2
$\Delta \leq 0.4$ or $\Delta \geq 0.6$	0.1
$\Delta = 0.5$	0

Table 5.5 Duty cycle presented as aging parameter (π in 10^{-2}) for different video sequences, with no inversion (f_R=∞), no controller

		f_R=1			$N = 0, f_R = \infty$		
Seq.	Base	Invert	Swap	Rotate	o_A=17	$o_A = 41$	$o_A = 61$
Basketball	37.9	3.9	32	3.7	14.5	14.4	14.4
BQTerrace	14.1	0.0	9.3	0.6	0.6	0.6	0.6
ChinaSpeed	38.9	0.0	30.1	22.1	21.3	21.3	21.3
FlowerVase	14.5	0.0	11.8	6.4	10.4	10.4	10.4
FourPeople	49.8	0.0	32.9	7.4	9.2	9.2	9.2
Johnny	52.6	0.0	34.8	5.6	25.5	25.5	25.5
Keiba	8.4	0.0	6.1	0.1	7.0	6.5	6.5
People	27.6	0.1	18.9	2.7	7.4	7.2	7.1
Traffic	42.6	0.1	29.4	8.0	8.8	8.9	8.5

"zeros" and "ones," the aging imbalance results in undesirable varying degradation of different SRAM cells. Sequences with large static structures in video frames (e.g., "Johnny") bring the most amount of stress on the SRAM cells because of static sample values. Sequences of this type are common for video security and communication applications, which are employed for a long duration. Camera panning and zooming sequences (like "BQTerrace," "FlowerVase," and "Keiba") usually have a low aging impact on SRAM cells. Largely static video sequences exhibit high aging due to less frequently changing data values (e.g., "Basketball," "ChinaSpeed," "FourPeople," and "Johnny").

This table also presents the aging parameters for the MWTs given in Fig. 3.13. For a comparison with these MWTs, results for using $N = 0$ (no Inverter Switch active) and only using the Write AGU to circularly write the frames in the frame memory are also given. Write AGU is activated by having $o_A \neq 0$. Using three different values for o_A, we notice considerable aging balancing achieved by only adapting the start addresses of the video frames in the SRAM memory. Specifically, largely static sequences get the most benefit. However, an interesting observation for the "Johnny" sequence can be made where we notice low aging improvement using Write AGU as compared to a sequence with similar aging profile (i.e., "FourPeople"). This is due to the fact that the static regions in the "Johnny" video frames are similar throughout the height of the frame, and the video samples of the new frame that overwrite the video samples of the previous frame have similar characteristics, thus only marginally contributing to stress relaxation. Hence, inversion is a better option in such a case. However, one can extend the Write AGU to start writing the next frame somewhere in middle of the row storing the previous frame samples. Further, o_A adaptation results in better aging resiliency as compared to nibble swapping, with negligible power penalty. Note that o_A is chosen as a prime number to make the cycle of start address of the video frame to be as large as possible.

The impact of parameter f_R and N on aging for different video sequences is tabulated in Table 5.6. Note that increasing f_R causes more frames to be inserted

Table 5.6 Duty cycle presented as aging parameter (π in 10^{-2}) for different video sequences, with $o_A = 0$, no controller

		f_R=1			f_R=3			f_R=7		
Seq.	Base	N =1	N =2	N =3	N =1	N =2	N =3	N =1	N =2	N=3
Johnny	52.6	29.9	11.8	1.1	39.9	29.6	23.2	45.1	39.2	35.7
Keiba	8.4	1.1	0.0	0.0	3.9	2.9	2.9	5.9	5.2	5.2
BQTerrace	14.1	4.3	0.4	0.0	8.0	5.2	4.9	11.0	9.5	9.3
Traffic	42.6	23.9	8.8	0.9	32.0	23.3	18.5	36.6	31.8	29.4

between two adapted frames at the same memory location. However, for static sequences like "Johnny," aging is accelerated due to increase in duty cycle bias. Sequences with camera panning, zooming, and frequent scene changes exhibit lesser sensitivity to changing f_R, mainly because of the video frame memory is overwritten continuously with dissimilar video samples. For example, the sequence "Keiba" and "BQTerrace" exhibit lower sensitivity to increasing f_R. Similarly, introducing more inverters by enabling the control signals of the Inverter Switches in the MWT will largely balance the aging of video memory. For slow moving, static sequences like "Traffic" and "Johnny," $N = 3$ results in a significantly better aging compared to $N = 2$ or 1. Highly dynamic sequences can still achieve the same aging with $N = 2$ or $N = 1$.

Further, experiments also reveal that using multiple frame memories result in almost the same aging balancing in all the frame memories. This is because frames are highly correlated, and instead of always overwriting a particular frame memory with the next frame, writing the second or third frame results in nearly the same aging statistics.

References

1. Altera. External memory interface handbook. June 2011. [Online]. Available: http://www.altera.com/literature/hb/external-memory/emi.pdf. Accessed 29 Sept 2015.
2. Juice Encoder– 4 in 1 MPEG-4 AVC/H.264 HD encoder. Antik Technology, [Online]. Available: http://www.antiktech.com/iptv-products/juice-encoder-EN-5004-5008/
3. Marvell 88DE3100 High-Definition Secure Media Processor System-on-Chip (SoC). [Online]. Available: http://www.marvell.com/digital-entertainment/armada-1500/assets/Marvell-ARMADA-1500-Product-Brief.pdf/
4. Cuomo, S., Michele, P. D., & Piccialli, F. (2014). 3D data denoising via nonlocal means filter by using parallel GPU strategies. In *Computational and Mathematical Methods in Medicine*.
5. Shafique, M., Zatt, B., Walter, F. L., Bampi, S., & Henkel, J. (2012). Adaptive power management of on-chip video memory for multiview video coding. In *Design Automation Conference*.
6. Sze, V., Finchelstein, D. F., Sinangil, M. E., & Chandraksan, A. P. (2009). A 0.7-V 1.8-mW H.264/AVC 720p video decoder. *IEEE Journal of Solid-Sate Circuits, 44*(11), 2943–2956.
7. Ma, Z., & Segall, A. (2011). Frame buffer compression for low-power video coding. In *International Conference on Image Processing*.

8. Ning, K., & Kaeli, D. (2005). Power aware external bus arbitration for system-on-a-chip embedded systems. *High Performance Embedded Architectures and Compilers, 3793*, 87–101.
9. Nios II Custom Instruction User Guide. Altera, (2011).
10. Nelder, J. A., & Mead, R. (1965). A simplex method for function minimization. *The Computer Journal, 7*(4).
11. Snyman, J., Stander, N., & Roux, W. (1994). A dynamic penalty function method for the solution of structural optimization problems. *Applied Mathematical Modelling, 18*(8), 453–460.
12. Carlson, T., Heirman, W., & Eeckhout, L. (2011). Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation. In *High Performance Computing, Networking, Storage and Analysis*.
13. Li, S., Ahn, J. H., Strong, R., Brockman, J., Tullsen, D., & Jouppi, N. (2009). McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Microarchitecture*.
14. Ostermann, J., Bormans, J., List, P., Marpe, D., Narroschke, M., Pereira, F., Stockhammer, T., & Wedi, T. (2004). Video coding with H.264/AVC: Tools, performance, and complexity. *IEEE Circuits and Systems Magazine, 4*(1), 7–28.
15. Tsai, C. H., Tang, C. S., & Chen, L. G. (2012). A flexible, fully hardwired CABAC encoder for UHDTV H.264/AVC high profile video. *IEEE Transactions on Consumer Electronics, 58*(4), 1329–1337.
16. Shafique, M., Bauer, L., & Henkel, J. (2010). Optimizing the H.264/AVC video encoder application structure for reconfigurable and application-specific platforms. *Journal of Signal Processing Systems (JSPS), 60*(2), 183–210.
17. Malvar, H., Hallapuro, A., Karczewicz, M., & Kerofsky, L. (2003). Low-complexity transform and quantization in H.264/AVC. *IEEE Transactions on Circuits and Systems for Video Technology, 13*(7), 598–603.
18. Wang, J.-C., Wang, J.-F., Yang, J.-F., & Chen, J.-T. (2007). A fast mode decision algorithm and its VLSI design for H.264/AVC intra-prediction. *IEEE Transactions on Circuits and Systems for Video Technology, 17*(10), 1414–1422.
19. Pan, F., Lin, X., Rahardja, S., Lim, K. P., Li, Z. G., Wu, D., & Wu, S. (2005). Fast mode decision algorithm for intraprediction in H.264/AVC video coding. *IEEE Transactions on Circuits and Systems for Video Technology, 15*(7), 813–822.
20. Kuo, H. C., Wu, L. C., Huang, H. T., Hsu, S. T., & Lin, Y. L. (2011). A low-power high-performance H.264/AVC intra-frame encoder for 1080pHD video. *IEEE Transactions on Very Large Scale Integrated Systems (TVLSI), 19*(6), 925–938.
21. Taiwan Semiconductor Manufacturing Company Limited. TSMC, [Online]. Available: http://www.tsmc.com/. Accessed 7 Oct 2015.
22. Altera. Nios II Process Reference Handbook. [Online]. Available: https://www.altera.com/literature/hb/nios2/n2cpu_nii5v1.pdf. Accessed 08 June 2015.
23. Fonseca, T. A., Liu, Y., & Queiroz, R. L. D. (2007). Open-loop prediction in H.264 / AVC for high definition sequences. In *SBrT*.
24. ModelSim – Leading Simulation and Debugging. Mentor Graphics, [Online]. Available: http://www.mentor.com/products/fpga/model/. Accessed 7 Oct 2015.
25. Kohonen, T. (1990). The self-organizing map. *Proceedings of the IEEE, 78*(9), 1464–1480.

Chapter 6
Experimental Evaluations and Discussion

The experimental evaluation of the techniques presented in Chaps. 4 and 5 are discussed in this chapter. In the previous chapters, we have already included the sensitivity analysis of the individual parts within the algorithmic and architectural details, whenever deemed useful. Here, the main results and comparison with other state-of-the-art techniques are presented, to provide an overview to the reader about gains and drawbacks of these techniques. Major emphasis of the results is video encoding, specifically H.264/AVC and HEVC video encoders. It must also be noted that these encoders have much more modules and higher complexity than many benchmark applications available in Parsec [1], MediaBench [2], Cosmic [3], and MiBench [4] benchmark suites.

The outline of this chapter is as follows. First, the parallelization of video application is discussed by evaluating the performance of compute and application configurations given in Chap. 4. Afterwards, the discussion regarding the resource budgeting (i.e., compute configuration) for multiple, multithreaded applications is carried out. In the end, architectural enhancements presented in Chap. 5 pertaining to implementation of highly efficient memory subsystem of the video system including AMBER and SRAM anti-aging circuits are evaluated.

6.1 Parallelization and Workload Balancing

This section discusses the evaluation of parallelization and workload balancing techniques for multithreaded, video processing benchmarks.

The authors would like to point out that this work was carried out when all the authors were in Department of Computer Science, Karlsruhe Institute of Technology, Karlsruhe, Germany.

M.U.K. Khan et al., *Energy Efficient Embedded Video Processing Systems*,
DOI 10.1007/978-3-319-61455-7_6

6.1.1 Software Architecture and Simulation Setup

The architecture of the video system with design methodology employing multiple hierarchies of application parallelization is shown in Fig. 6.1. A library written in any high−/mid-level language (e.g., C++) for generating the compute configuration produces the compute configuration (Sect. 4.2) and sets the application configuration (Sect. 4.3) given the initial configuration matrix A (Fig. 4.7). This ensures that there is little or moderate effort from the application designer to incorporate the architectural enhancements in the video application. This library also handles the frequency generation and frequency model adaptation (Sect. 4.2.5.2) at runtime. Similarly, another library handles parallelization routines of the application. This library implements a workload queue, which is filled with the callback function (i.e., the tile processing function) and the associated arguments to that function. Although this library can utilize OpenMP at the back end, more control and options are available if pthread [5]-based solutions are employed. However, the interface of this library should just accept the callback function and a parameter structure for portability and scalability. Afterwards, a start signal initiates the parallel processing where all the callback functions are executed.

The applications can embed these libraries with minimal effort. Note that for the application configuration case study of HEVC, the reference encoding software (named HM encoder [6]) does not provide parallelization and has a large memory footprint. To employ parallelization and other functionalities missing in the reference software, we developed our C++ -based open-sourced multithreaded HEVC intra-encoder, ces265 (for Windows/Linux), in Chair for Embedded Systems

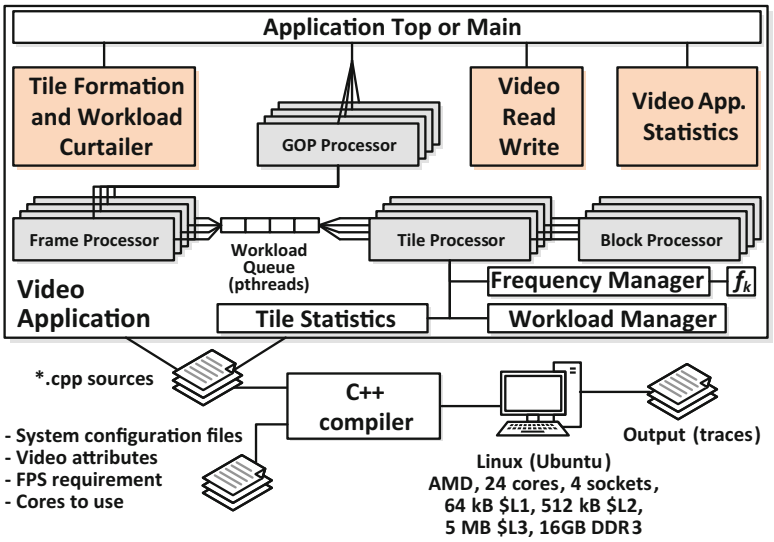


Fig. 6.1 Reference multithreaded video application architecture, employing multiple thread hierarchies

(CES), Karlsruhe Institute of Technology (KIT), Germany. A single thread of ces265 is ~13× faster than HM software (more information in Appendix B). Other open-source HEVC encoders (e.g., x265 [7]) are also available; however, they do not include the required adaptations, and neither do they lend themselves easily for the architectural enhancements discussed in this book. Further, due to a small memory footprint, no platform-specific intrinsic and SIMD instructions, ces265, can be employed and tested in small embedded systems.

For evaluations, a 4× Six-Core AMD Opteron processor [8] (also called "Istanbul" processor) is used. During the experiments, the frequency scaling of all cores is disabled, and using a fixed frequency, the number of cycles per block ($c_{k,\boldsymbol{\alpha}}$, see Eq. (4.12)) can be calculated given the computation time. Note that multiple programs are running in parallel on this computer; therefore, it is expected that the load on the computer fluctuates, and therefore, it mimics the scenario targeted in this book. Thus, it corresponds to the changing workload of the system running the application. The set of supported frequencies ($\boldsymbol{f}_{set}$) used for these evaluations is:

$$\boldsymbol{f}_{\text{set}} = \{1.0, 1.2, \ldots, 3.0\}_{\text{GHz}} \tag{6.1}$$

The frequencies of the cores are used to estimate the power consumed by the application, by running the application on the Sniper many-core simulator [9] and McPAT [10]. Afterwards, a similar technique [11, 12] is used to estimate the final power consumption of the system. For the techniques presented here, it is assumed that no prior knowledge about the frequency estimation model constants $\boldsymbol{\omega}_k$ and matrix E is delivered (see Eq. (4.12)). Hence, at the start of video processing, all elements of $\boldsymbol{\omega}_k$ are randomly chosen, and E is initialized with $999I$, where I is an identity matrix.

6.1.2 Compute and Application Configuration for Uniform Tiling

Table 6.1 evaluates the compute configuration technique for the ces265 HEVC encoder as benchmark, using different test video sequences with varying dimensions. This table relates the number of given cores (r_{tot}), actual used cores (k_{tot}) after frequency-estimation-model stabilization, and average time per frame (t_{frm}) used for encoding. As noticed, $k_{tot} \leq r_{tot}$, denoting that the RLS-based model tuning technique will derive the correct number of cores to sustain the workload, irrespective of initial settings.

Power Savings Figure 6.2 shows the power and video quality comparison when the optional runtime adaptation of the workload (application configuration in Sect. 4.3) is activated. These results are generated using $f_p = 5$ and $\varepsilon = 0.05$ or 5% tolerance to the number of compressed output bytes. As seen in Fig. 6.2a, significant power saving (up to 42.48%, average 39.21%) is obtained if the workload tuning

Table 6.1 For given (r_{tot}), the used (k_{tot}) number of cores and average time per frame (t_{avg}, in msec) for different video sequences, using $f_p = 5$

Sequence number	Sequence	Resolution $w \times h$	Given cores (r_{tot})	Used cores (k_{tot})	t_{frm} [msec]
A	*Ballroom*	640 × 480	4	2	180
B	*BQMall*	832 × 480	8	3	197
C	*BQTerrace*	1920 × 1080	24	16	206
D	*ChinaSpeed*	1024 × 768	8	3	191
E	*FourPeople*	1280 × 720	12	6	193
F	*Johnny*	1280 × 720	12	6	193
G	*Keiba*	832 × 480	4	2	183
H	*RaceHorses*	832 × 480	16	2	177
				Average	**190**

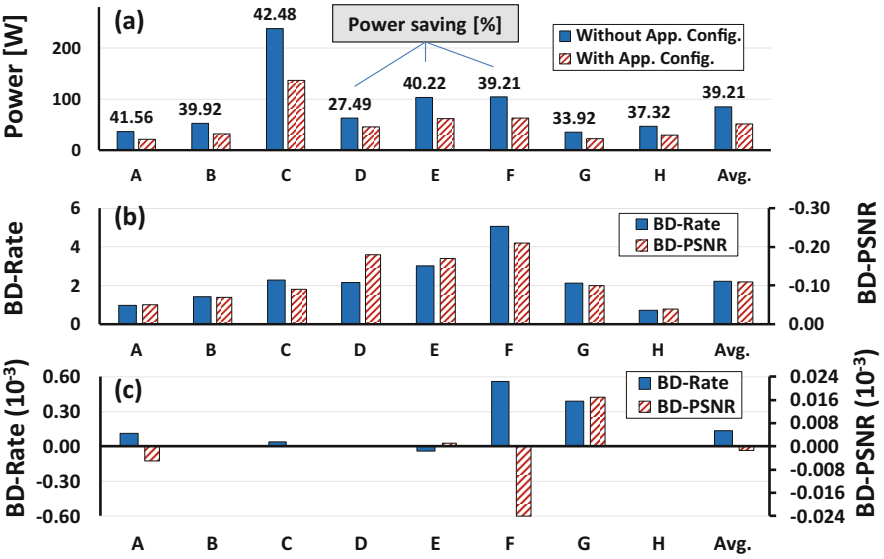


Fig. 6.2 For the video sequences given in Table 6.1 and using $f_p = 5$, $\varepsilon = 0.05$ or 5%, (**a**) power, (**b**) unavoidable video quality loss in terms of BD-Rate and BD-PSNR without any application configuration, and (**c**) with application configuration

techniques presented in this book are used, which shows that the application configuration by leveraging the application knowledge results in high power savings. For this graph, the power savings are computed by using the following relation:

$$\text{PowSave}[\%] = \left(1 - \text{Pow}_{\text{AppConfig}}/\text{Pow}_{\text{NoAppConfig}}\right) \times 100\% \qquad (6.2)$$

Video Quality Comparison For the sequences given in Table 6.1, Fig. 6.2b, c present the BD-Rate and BD-PSNR [13], with and without application configuration. BD-Rate denotes the percentage increase in the average bitrate of an encoder under test compared to an anchor/reference encoder, and BD-PSNR denotes the reduction in PSNR (in dB) against the same anchor encoder. The formula to compute BD-Rate and BD-PSNR requires to compute PSNR at a specific QP value. Usually, the following formula is used to compute PSNR which is also employed in generating these results:

$$\text{PSNR}_{\text{QP}} = \frac{4 \times \text{PSNR}_{\text{Y}} + \text{PSNR}_{\text{Cb}} + \text{PSNR}_{\text{Cr}}}{6} \tag{6.3}$$

For the case of Fig. 6.2b, the anchor encoder is the baseline encoder, using only a single tile per frame. However, this single-tile encoder is unrealistic as it is not possible to support large HEVC workload by using a single core. The encoder under test uses the compute configuration technique (multiple tiles per frame) without any application configuration. Therefore, the BD-Rate and BD-PSNR for the encoder under test in Fig. 6.2b only show the unavoidable overhead that must be paid while satisfying the throughput constraints (frame rate). In short, this figure shows quality degradation due to mandatory tiling. On average, BD-Rate increases by 2.22% and BD-PSNR degrades by 0.11 dB. For Fig. 6.2c, the anchor encoder is the multi-tile encoder used as the encoder under test in Fig. 6.2b. In this evaluation, the encoder under test employs both compute and application configuration. The degradation in bitrate and PSNR shown here accounts for the degradation produced due to workload tuning (Sect. 4.3.1). However, note that the encoder under test in Fig. 6.2c has both the BD-Rate and BD-PSNR expressed in 10^{-3}, which means that there is negligible video quality degradation produced by the workload tuning techniques discussed in Chap. 4. On average, the BD-Rate is negligibly reduced by 0.13×10^{-3}% and BD-PSNR reduces by 1.4×10^{-6} dB.

Comparison with State-of-the-Art Techniques Figure 6.3 compares the power efficiency of the compute and application configuration technique with those presented in [14]. In [14], authors try to maximize the throughput of a divisible workload and adapt the number of cores to process this workload, whereas compute and application configuration techniques in this book try to meet the application's

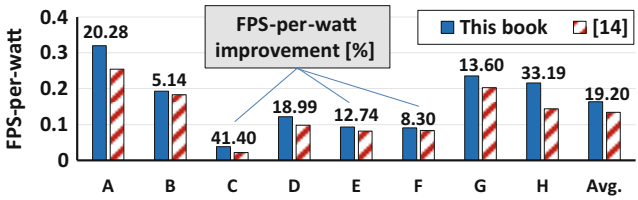


Fig. 6.3 Comparing the power efficiency (in terms of FPS-per-watt) of the workload balancing technique employing compute and application configuration compared to [14]. The percentage improvement of the technique presented in this book is written on top of the bars

deadline while minimizing the power consumed by the system. Therefore, Fig. 6.3 compares the FPS supported per unit of power (FPS-per-watt) for both techniques. One can imagine FPS-per-watt being a representative for throughput-per-watt. For sequences given in Table 6.1, techniques outlined in this book result in significant improvements. Note that for this analysis, the power consumption and FPS (reciprocal of average time to process a frame) are taken after the first retiling event. On average, the techniques presented here increase the FPS-per-watt metric by 19.20% compared to [14].

Runtime System Dynamics Figure 6.4 shows the power, time to process a video frame (t_k), and frequencies (f_k) of all cores for the HEVC encoding process. For these experiments, $1/f_p = 1/5 = 200$ msec, $\varepsilon = 0.05$, and $z = 8$ (see Sects. 4.2 and 4.3 for more information). Since in the first few epochs, the frequency estimation model is untrained (see Sect. 4.2.5.2), therefore, the number of tiles/cores (k_{tot}) and the initial estimated frequency are considerably off the required values in these epochs. In addition, t_k is unnecessarily small resulting in high power consumption (Fig. 6.4a–d). Of course, the initial untrained estimation model might result in extremely large t_k. After a few epochs, the frequency model stabilizes, and t_k becomes closer to the required 200 msec range. Therefore, there is some power inefficiency at start. However, the advantages of determining the constants online outweigh the disadvantage of small power wastage that occur only in the initial processing stage. If the video system runs for a long duration, this initial overhead might be negligible.

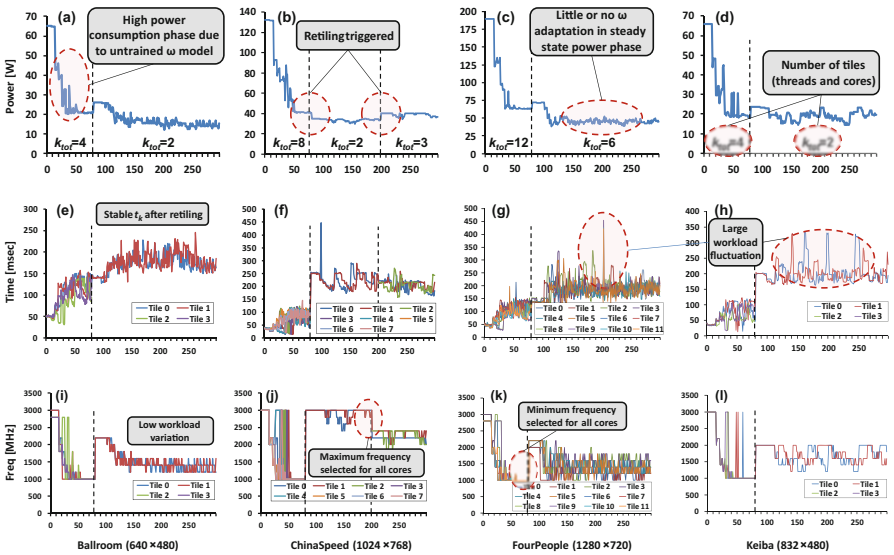


Fig. 6.4 Using $f_p = 5$ and $\varepsilon = 0.05$ or 5%, runtime adaptation of (**a–d**) power, (**e–h**) time per tile, and (**i–l**) frequency per core for different video sequences

Simulations with Sniper many-core simulator [9] and McPAT [10] show that only about 2.8 µsec and 133 µJ energy is spent in calling the RLS filter once after an epoch, which is negligible overhead when compared to processing a single frame (e.g., the "Ballroom" sequence on a single core takes around 858 msec with 16.81 J of energy). Moreover, the power wastage in the $\boldsymbol{\omega}_k$ training phase can be avoided by determining the constants offline and then tuning them online (via RLS). In addition, the overhead of retiling itself is negligible, because it is invoked only once after 80 frames are encoded via HEVC. We note that f_k gradually moves towards a stable value, due to the adjustment of frequency estimation model constants ($\boldsymbol{\omega}_k$, Eq. (4.12)). After retiling, the number of cores used for encoding (k_{tot}) and their frequencies are adjusted, and thus, t_k becomes considerably closer to $1/f_p$ as shown in Fig. 6.4e–h. This results in a steady power consumption after the adjustment phase.

Note that retiling is done when frequency of all the cores is stuck at maximum or minimum value (Fig. 6.4j, k, see Sect. 4.2.6). An interesting case is demonstrated by the "ChinaSpeed" sequence, where retiling is done twice during the encoding process. Further, the workload of "Keiba" sequence is highly fluctuating, and it results in large variations in frequency and power. For the "FourPeople" sequence, the average power before retiling is 99.08 W, and after retiling it reduces to 48.32 W (an improvement of ~2.05× after frequency model adjustment).

In Fig. 6.5, f_k, θ_k, d_k, and b_k update of different tiles for distinct sequences using HEVC are shown. The optional tuning of workload (θ_k and d_k) due to the allowable increase in bitrate ($\varepsilon = 0.05$) causes a change in the allocated frequencies of the cores. Note that the large jump in b_k is due to retiling, because retiling results in a larger tile. Hence, now we have more b_k per tile but smaller number of total bytes. Note that the workload adaptation techniques presented in this book can handle these situations. Further, in Fig. 6.5 (c), $b_k is$ progressively increasing, thereby resulting in the increase of the workload (θ_k and d_k) and increasing amount of $f_{k,h}$ in the epoch (see Fig. 4.8). Figure 6.5d is the opposite case, and the workload is steadily declining, with increasing number of $f_{k,l}$ in the epoch.

To display the effect of throughput requirements, Fig. 6.6 demonstrates the impact of FPS (f_p) for the frame interpolation application, with four available cores ($r_{tot} = 4$) and retiling test after every 80 frames. Note that these experiments only utilize the compute configuration, i.e., selecting the required number of cores and maximum workload per core. At start, the frequency estimation model has not

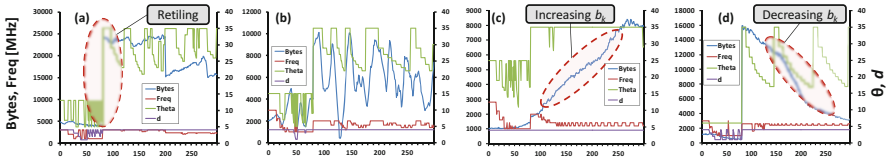


Fig. 6.5 Using the same settings as Fig. 6.4, HEVC frequency and workload tuning for (**a**) Tile 1 of "ChinaSpeed," (**b**) Tile 1 of "Keiba," (**c**) Tile 0, and (**d**) Tile 7 of "BQTerrace"

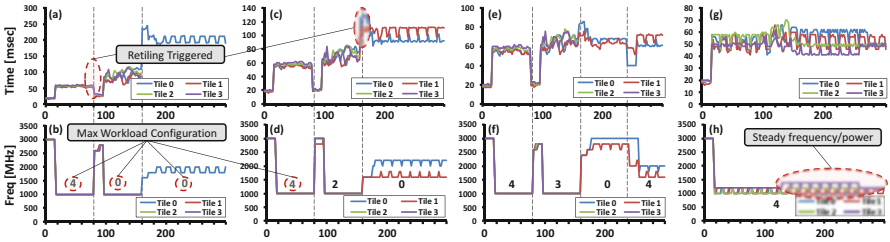


Fig. 6.6 Time, frequency, and maximum configuration of all the cores for frame interpolation application with allowed number of cores (r_{tot}) = 4: (**a**, **b**) FPS = 5, (**c**, **d**) FPS = 10, (**e**, **f**) FPS = 15, (**g**, **h**) FPS = 20

tuned to the current application scenario, and thus, the number of threads and the frequencies of the cores are higher while supporting a lower maximum workload ($\boldsymbol{\alpha_{k,m}}$) denoted by a higher tuple number within configuration matrix A. As seen, the frequency estimation function is gradually stabilizing, and the execution time of the frame is steadily reaching the required throughput demands. In addition, the maximum supportable workload is also increasing (denoted by a lower tuple number). Different FPS requirements however do play a role, whereby we notice that Fig. 6.6d with highest FPS = 20 does not get its workload increased due to the limitation of resources.

6.1.3 *Compute Configuration with Non-uniform Tiling*

Simulating the ces265 video encoder via Sniper and McPAT, the power and resource utilization of both uniform and non-uniform tiling (Sect. 4.2.2) is tabulated in Table 6.2. This table presents video quality in terms of BD-Rate, BD-PSNR, and the total power savings in percentage for non-uniform tiling against the uniform tiling technique. For these experiments, a many-core system with no DVFS capabilities and with the following model (derived via offline regression analysis) is used:

$$t = 29.65 - 1.28QP + 0.05n_{\text{frm}} + 3.38n \tag{6.4}$$

In this equation, t is the time (in msec) for HEVC intra-encoding n CTUs of a video frame with n_{frm} total CTUs, at 2.6 GHz. Using this equation and those given in (4.5)-(4.7), one can determine the tile structure. For the uniform tiling technique, the number of cores and tiles is equal. Non-uniform tiling uses the bin-packing heuristic shown in Fig. 4.5 to determine the number of cores actually required to sustain the required throughput.

As seen, various sequences with varying frame rates are tested, which can mimic different throughput requirement scenarios. Note that non-uniform tiling, on average, saves three cores as compared to the uniform tiling. Further, the average video

Table 6.2 Number of cores, tiles, and video quality in terms of BD-Rate and BD-PSNR, along with the system power savings for non-uniform tiling technique (ΔP), using an x86 many-core system running at 2.6 GHz for 45 nm technology

		Uniform tiling				Non-uniform tiling				
Video Sequence	FPS	Cores	Tiles	BD-Rate	BD-PSNR	Cores	Tiles	BD-Rate	BD-PSNR	ΔP
Ballroom (640 × 480)	**20**	35	35	8.961	−0.449	30	32	8.747	−0.4381	+10.5
Flamenco (640 × 480)	**20**	35	35	12.756	−0.719	30	32	12.078	−0.682	+10.3
Vassar (640 × 480)	**15**	20	20	6.904	−0.237	20	20	6.895	−0.238	−0.30
Vassar (640 × 480)	**20**	35	35	11.375	−0.387	30	32	9.862	−0.336	+10.3
Keiba (832 × 480)	**15**	35	35	9.069	−0.464	30	32	7.452	−0.382	+14.0
RaceHorses (832 × 480)	**15**	35	35	1.167	−0.067	30	32	2.050	−0.118	+14.4
BasketballDrill (832 × 480)	**10**	20	20	5.896	−0.276	17	20	6.120	−0.286	+4.0
BasketballDrill (832 × 480)	**18**	35	35	8.580	−0.400	33	40	10.524	−0.487	−0.70
Average	**17**	**31.25**	**31.25**	**8.088**	**−0.375**	**27.50**	**30**	**7.966**	**−0.371**	**+7.81**

output quality is better, the number of tiles is reduced, and usually power savings are obtained when non-uniform tiling is used. Additionally, note that growing frame rates or increasing resolution of video sequences generally results in higher power savings by the non-uniform technique. This is mainly because a better workload balancing is achieved in case of non-uniform tiling technique compared to the uniform technique. Further, an added advantage of non-uniform tiling technique is that it can be employed for workload balancing on a system without DVFS.

6.1.4 Workload Balancing on Heterogeneous Platforms

Using the same heterogeneous cores and benchmarks given in Table 3.1 and Fig. 3.9, a heterogeneous system with multiple compute nodes is used to evaluate the workload balancing technique given in Sect. 4.4. Specifically, a four-core multi-core heterogeneous system ($r_{tot} = 4$ with two "tiny," one "medium," and one "large" core given in Table 3.1) is tested. A throughput constraint of $t_{i,max} = 30$ msec is set by the user for all the benchmarks. The quality metric is throughput-per-watt and is given by the relation $1/(t_{frm} \times p_{tot})$, where t_{frm} is the time to process a task and p_{tot} is the power consumed by the heterogeneous multi-core system.

The average throughput-per-watt with different benchmark applications (i.e., H.264/AVC DCT and quantization and HEVC intra-encoding) is reported in Fig. 6.7. As noticed, for these benchmarks, the efficiency indices $\phi_{k,1}$ and $\phi_{k,2}$ outperform the load balancing technique given in [15] that uses a bin-packing heuristic to distribute the load among cores. Although the efficiency index $\phi_{k,3}$ does not perform as well as the other indices and even beaten by [15]. Hence, this shows that only power-aware load distribution will not result in higher performance for metrics like throughput-per-watt. It marginally performs better for the HEVC benchmark, since number of subtasks n_i for HEVC are considerably lesser (each subtask is a tile to processes, so n_i is from 1 to 10) compared to the DCT and quantization benchmark (see Fig. 6.8). When n_i increases, the freedom to map these subtasks also increases, and the performance of the load balancing improves [16].

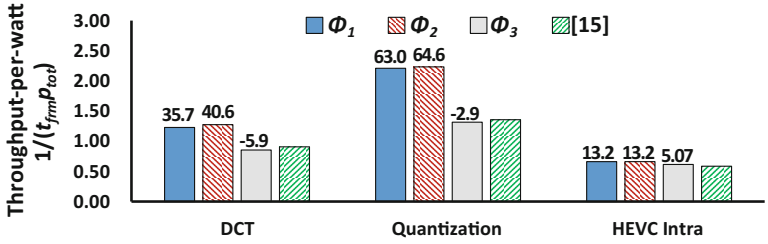


Fig. 6.7 Average throughput-per-watt $1/(t_{frm} \times p_{tot})$ for different efficiency indices presented in this book and [15]. The performance improvement (in percentage) against [15] is written on top of the bars

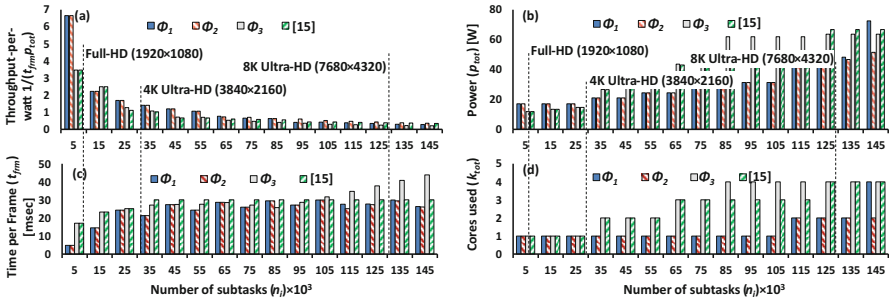


Fig. 6.8 (**a**) Comparison of the techniques presented here and [15] for the "DCT" benchmark. The number of DCTs (or subtasks) for different image resolutions are portrayed on top of the graph. For the "quantization" benchmark, (**b**) shows power consumption and (**c**) time per frame and (**d**) denotes the number of cores used for different n_i, using the efficiency indices and the technique in [15]

Figure 6.8a breakdowns the throughput-per-watt performance for the DCT benchmark, for all efficiency indices and the technique proposed in [15]. To evaluate the performance of these techniques for a wide range of throughput requirements, the throughput-per-watt metric is plotted for the increasing number of subtasks (i.e., n_i). For clarity, the image resolution pertaining to a specific number of DCTs is written on the graph as well. As seen, the performance of efficiency index $\phi_{k,1}$ and $\phi_{k,2}$ is considerably better than $\phi_{k,3}$ and [15].

A deeper breakdown for the quantization benchmark is shown in Fig. 6.8b–d. This figure plots the power, time (i.e., throughput), and number of cores actually used for processing (k_{tot}). As seen, for efficiency index $\phi_{k,1}$ and specially for $\phi_{k,2}$, the load distribution is not only based upon the power of the node but also the amount of time the node takes to process a subtask. Index $\phi_{k,1}$ uses the combination of power and cycles consumed in processing the load, while $\phi_{k,2}$ uses only the number of cycles. Therefore, these two indices and [15] always result in throughput being satisfied ($t_{i,k} \leq t_{i,max} = 30$ msec) as shown in Fig. 6.8c. However, $\phi_{k,3}$ starts to miss the deadline once the load on the system increases considerably. This is expected because $\phi_{k,3}$ does not consider the complexity of a subtask (in terms of cycles or time). Moreover, as shown in Fig. 6.8d, the number of cores used for processing by $\phi_{k,1}$ and $\phi_{k,2}$ is also lesser than $\phi_{k,3}$ and [15].

6.2 Resource Budgeting

This section provides experimental evaluation of resource (cores and power) budgeting for mixed multithreaded applications, while maximizing the throughput of all parallel running applications. For these experiments, the focus is on software level multicasting (see Sect. 2.2.2) of parallelizable HEVC encoders. However, it should be noted that the resource budgeting techniques presented in Chap. 4 and evaluated here are also applicable to other parallelizable/malleable applications.

6.2.1 Experimental Setup

In these simulations, epoch size is chosen as one second, i.e., a total of FPS number of frames. Therefore, the inter-cluster resource and power adaptation takes place after every FPS frames have been processed. Following the methodology given in [17], the 22 nm results are scaled to 11 nm using ITRS-provided technology scaling factor [18] to have representative scenarios involving power wall. These experiments consider a many-core chip with number of cores $k_{tot} = 24$. In this evaluation:

$$\boldsymbol{f}_{\text{set,GHz}} = \{1.0, 1.2, \ldots, 3.0\} \tag{6.5}$$

Experiments show that the power of a core at 3.0 GHz is 4.94 W. Hence, the chip's maximum power with 24 cores is 118.56 W. Note that the power exchange in Eq. (4.37) uses $\psi_1 = 8$ and $\psi_2 = 2$ in Eq. (4.38). These parameters are obtained using empirical analysis of HEVC video encoding. For other applications, similar application-specific constants can be derived.

In order to evaluate the technique presented in this book, a trace-based simulation method is used. By disabling the frequency scaling on a real system (core i7, 3.4 GHz, 16 GB RAM, Ubuntu Linux), HEVC encoding is run, and the actual time consumed for encoding each subtask (i.e., a video tile) is written to a text file. While simulating, this file is read to extract the time associated to process a video tile. For scenarios involving multichannel HEVC encoding, video sequences with diverse texture and motion characteristics are utilized in these experiments, as tabulated in Table 6.3. Here, the term "set" refers to the set of video sequences, which are encoded concurrently on the same chip. One can imagine that at one time, all the videos in a row of this table are processed on the chip. An encoder processes a single video in the set. These experiments use ces265 HEVC video encoder [19] for all the encoders.

For these analyses, the technique proposed in [11] is also chosen for assessment. Note that [11] does not consider application-level resource allocation and assumes cluster sizes are available. Further, there is no adaptation of cluster size at runtime. In addition, the two-level, temporal power budgeting (among applications after

Table 6.3 Test video sequences sets used for resource budgeting experiments. The display format is: "Name (fps_{min}) ($w \times h$)"

Set	*Enc-0*	*Enc-1*	*Enc-2*	*Enc-3*
Set-0	*Ballroom* (20) (640 × 480)	*Basketball* (10) (832 × 480)	*Exit* (20) (640 × 480)	*Vassar* (10) (640 × 480)
Set-1	*Ballroom* (10) (640 × 480)	*Football* (10) (352 × 288)	*Foreman* (20) (352 × 288)	*FourPeople* (5) (1280 × 720)
Set-2	*Bubbles* (10) (416 × 240)	*Coastguard* (20) (352 × 288)	*Keiba* (10) (832 × 480)	*RaceHorses* (5) (832 × 480)
Set-3	*BQMall* (5) (832 × 480)	*ChinaSpeed* (10) (1024 × 768)	*Flamenco* (10) (640 × 480)	*Vassar* (5) (640 × 480)

epochs and among threads after processing a data frame) contained within the techniques presented here is not incorporated by [11]. Moreover, note that [11] allocates more power budget to the cluster with high relative performance-to-power ratio. For the video encoding scenario discussed above, this would mean that the video encoder, which generates the highest FPS, is allocated more power by [11]. This can result in the following outcome. If all the cores of the cluster are already running at the minimum frequency, and a lower power is allocated to the cluster in the next epoch, it will not be possible to reduce the frequencies of the cores further. Therefore, the power budget of the complete chip (p_{tot}) will be exceeded because the cluster having cores running at minimum frequency is still consuming the same power instead of the new (lower) one. However, this is problem is circumvented by the resource budgeting technique presented in Chap. 5.

6.2.2 *Results and Discussion*

Figure 6.9 shows the total achieved FPS by the technique presented here and in [11] for different power configurations. As discussed before, the total theoretical power the chip can consume is 118.56 W. However, to make representative power-wall or dark silicon (DS) scenarios, only a part of this power is pumped into the system, which is marked as the TDP of the system. Every bar shows the stacked FPS for each set of Table 6.3. When a larger TDP is available (e.g., at $p_{tot} = 100$ W or 15% DS), the frequencies of the cores have a higher degree of freedom, and therefore, the technique outlined in this book achieves 30.86% higher FPS than [11]. However, reducing TDP (e.g., when $p_{tot} = 42$ W or 65% DS) also reduces the degree of freedom, and therefore, the FPS improvement also declines (still 15.6% higher than [11]). Note that for these experiments, Eq. (4.40) is used for determining the initial cluster size.

Figures 6.10 and 6.11 breakdown cluster size (i.e., number of cores k_i) and inter-cluster power p_i for all video encoders at TDP of 100 W (~15% DS). As seen,

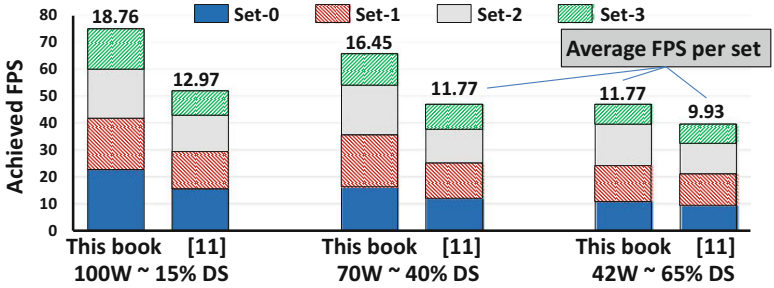


Fig. 6.9 Stacked bar chart for achieved FPS using the resource budgeting technique presented in this book and [11] for different amount power pumped into the complete chip. The average FPS achieved per set is written on top of the bars

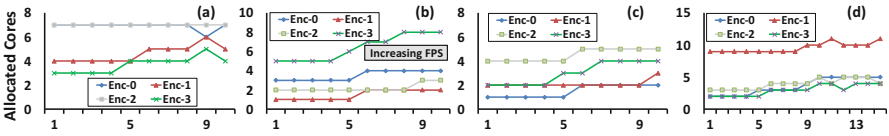


Fig. 6.10 Core allocation at runtime for encoders, at TDP $p_{tot} = 100$ Watts (i.e., ~15% dark silicon, DS), of (**a**) Set-0, (**b**) Set-1, (**c**) Set-2, and (**d**) Set-3. The total number of cores used by all the encoders is always less than or equal to the total available cores

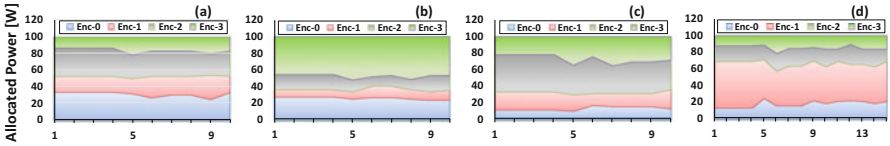


Fig. 6.11 Stacked area plot for cluster level allocated power at runtime for encoders, with TDP $p_{tot} = 100$ Watts (~15% DS), of (**a**) Set-0, (**b**) Set-1, (**c**) Set-2, and (**d**) Set-3. Total power consumed by the chip is always less than or equal to the TDP

encoders that process larger resolutions and higher fps_{min} requirement video sequences require more cores k_i and power p_i allocation to their associated cluster. For example, Enc-3 of Set-1 is allocated higher k_i and p_i compared to the other encoders. Note that the resource budgeting algorithms presented here will adapt the size of the cluster by including or excluding cores from a cluster. This is shown in the runtime adaptation of both k_i and p_i in these figures, where the clusters can exchange cores and power among themselves. Furthermore, note that k_{tot} and p_{tot} of the system never exceed the thresholds ($k_{tot} \leq 24$ and $p_{tot} \leq 100$) and p_{tot} is instantly utilized completely unlike the control-based scheme of [11].

The intra-cluster power profiles of the encoders associated with videos of Set-3 at 100 W (~15% DS) are shown in Fig. 6.12. The increase in number of cores/threads by adjusting the FPS requirement at runtime is shown in Fig. 6.12a. Figures 6.11a and 6.12b jointly show the inter-cluster power exchange between encoders, whereby Encoder-1 transfers its power to Encoder-0. Figure 6.12c shows the variation in allocated power to Encoder-2 at runtime. Furthermore, the allocated power to the individual cores is also adapted. The ripple in Fig. 6.12 (clearly visible in Fig. 6.12d) is due to intra-cluster power exchange (i.e., power exchange among cores and their frequency adaptation) given in Algorithm 7.

6.3 Memory Subsystem

This section demonstrates the experimental evaluation of the memory subsystem architecture described in this book. Firstly, this section presents the impact of hybrid memories on the system performance, and afterwards, power-efficient SRAM aging balancing techniques are evaluated when used in video processing systems.

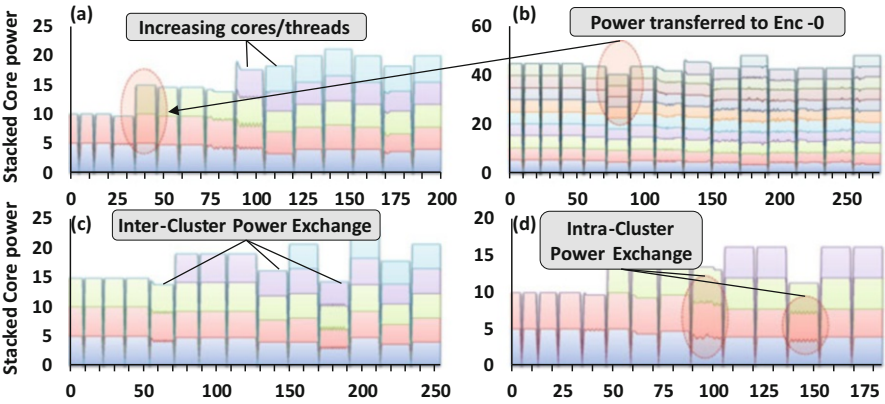


Fig. 6.12 Intra-cluster power profile of (**a**) Enc-0, (**b**) Enc-1, (**c**) Enc-2, and (**d**) Enc-3 of Set-3 for each frame, shown as a stacked power plot per core of the cluster. Example power exchanges are also highlighted

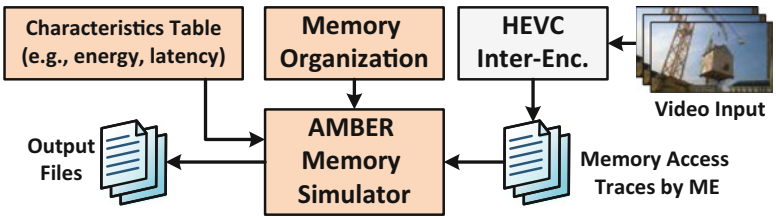


Fig. 6.13 Simulation setup for testing hybrid memory subsystem

6.3.1 AMBER: Hybrid Memories

This section discusses the evaluation of conjugating NVMs with SRAM-based memories. For these results, a MRAM-based NVM is considered, which stores the video frames on the on-chip frame memories. In order to test the AMBER architecture of using hybrid memory structures, the motion estimation engine within HEVC reference software (HM-9.2 encoder [6]) is augmented to include the latency, energy, and power profiles of the memories. The AMBER architecture is implemented via SRAMs and MRAMs for which the numbers are taken from Table 3.2 using a 65 nm technology [20]. Since the MRAM and SRAM read energy and latency numbers are similar and dependent upon the motion estimation algorithm, therefore, they are not included in the results.

Figure 6.13 illustrates the setup used to evaluate AMBER. The inputs to the AMBER memory simulator are:

- Memory traces for inter-encoding (i.e., motion estimation) generated using the HEVC reference software

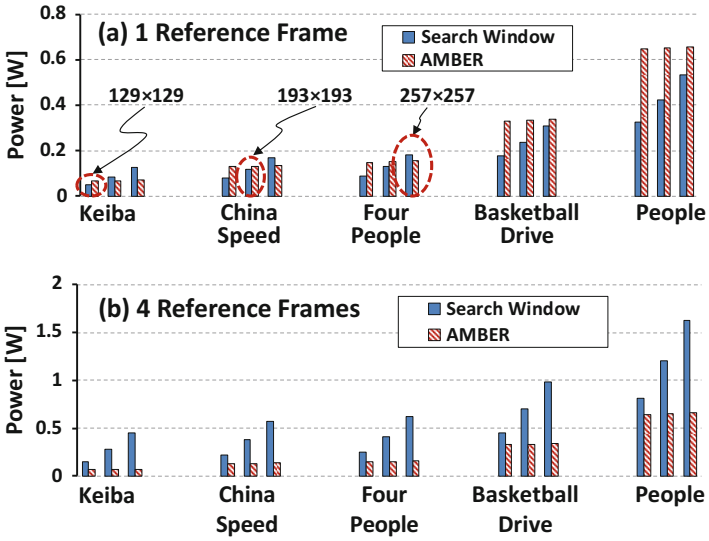


Fig. 6.14 Power consumption for the search window technique and AMBER, using (**a**) one reference and (**b**) four reference frames for motion estimation of HEVC inter-encoding using the reference software

- Memory architecture (i.e., hierarchy) and organization
- Memory characteristic table that contains energy, latency, and area of different memory types

In Fig. 6.14, the power consumption of the usual search window-based technique with prefetching (outlined in Chap. 2) and AMBER is presented for three different search window sizes, used in ME for encoding HEVC inter-frames. For evaluation, various video sequences recommended by the JCT-VC [21] are used, and details about the video sequences can be inferred from Tables 6.1 to 6.2. The small search window size is 129 × 129 pixels, medium search window size is 193 × 193 pixels, and for large search window, 257 × 257 pixel size is chosen. Note that these sizes produce best results according to the video frame dimensions. That is, a larger frame requires a bigger search window, and a larger search window might not be efficient for video sequence with smaller resolutions.

For only a single reference frame (Fig. 6.14a), the energy consumed by the search window technique is better. This is because only a single reference frame is used, and the total dynamic read-and-write energy consumed by the search window is small. However, increasing the size of the search window introduces more leakage and dynamic power consumption, and the power of search window-based technique increases and surpasses that of AMBER. Similarly, larger frame dimensions cause more leakage power consumption in AMBER because the sector height increases and, therefore, the total power consumption of AMBER system increases.

On the other hand, AMBER produces better results for multiple reference frames (Fig. 6.14b). The reason is that multiple reference frames are being read and written to multiple search windows, and the total power consumption of search window technique therefore also increases. On average, employing AMBER results in 43% energy savings compared to the search window-based technique.

6.3.2 SRAM Anti-aging Circuits

This section presents the detailed results of analyzing the aging of SRAM-based memories, using different state-of-the-art architectures and the SRAM anti-aging architecture presented in Chap. 5. The sensitivity analysis of these architectures is already discussed in Sect. 5.5.5. Here, we will focus on experimental setup and evaluation of the complete system.

6.3.2.1 Experimental Setup

Hardware Synthesis and Aging Estimation The details of the experimental setup are presented in Fig. 6.15. The aging resilient architecture is implemented in VHDL. These circuits include the different aging balancing circuits (inverter, bit swap, and bit rotate) and other architectural components like MWT, MRT, AGU, etc. The architecture is synthesized using a 65 nm TSMC technology [22] (0.99 Volt, 25 °C junction temperature, FF corner) using Synopsys Design Compiler [23]. The gate-level simulations and functional verifications of the proposed architecture are performed using ModelSim [24], which is also used to generate the

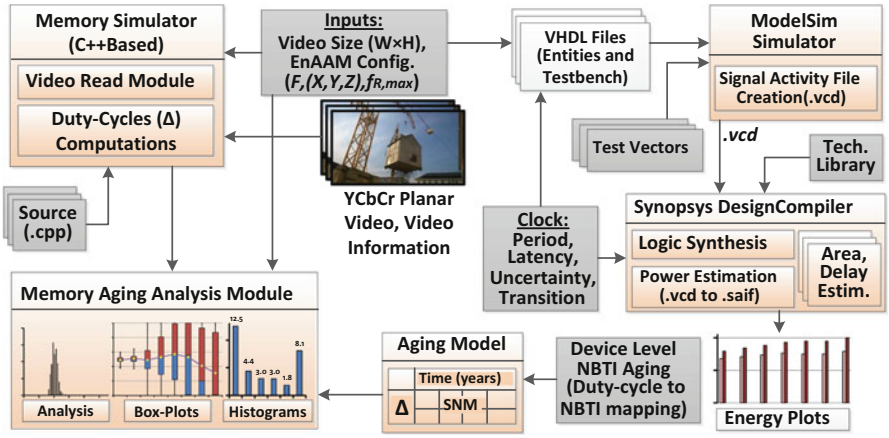


Fig. 6.15 Experimental setup for analyzing the impact of different components on aging of SRAM circuits

switching activity waveforms for power analysis. The memory simulator generates the duty cycle of all the 6 T SRAM cells for the current video input. Using this duty cycle information, one can estimate the aging rate by mapping the duty cycle to SNM degradation.

Plotting Aging Results For fast analysis and visualization of memory aging, we have developed a GUI-based tool (written in C#) which is made open-source [25]. This tool accepts user configurations like memory size, location of test data sets, total number of years the memory will be used, etc. Using this tool, memory analysis and aging impacts can be visualized with ease. Additionally, it can also perform basic image and video processing applications (like filtering, color conversion, etc.). This tool automatically generates stressmaps, box plots, and duty cycle histograms for different input videos (examples are shown in Sect. 3.3.3).

Test Video Sequences For these experiments, different test video sequences recommended by the Joint Collaborative Team on Video Coding (JCT-VC) [26] are used, which are available for download and testing [21, 27]. Some representative video sequences used in these experiments are described briefly in Table 6.4 along with their key attributes. These videos have diverse characteristics, and they can comprehensively represent various video capture scenarios.

6.3.2.2 Results and Comparison

In Fig. 6.16, the percentage duty cycle histograms are plotted for different MWTs with a single frame memory for different test video sequences. The explanation about these MWTs is given in Table 6.5. Except for the "base" and "controller" case, these histograms are generated with $f_R = 1$, i.e., every second frame is adapted. For the base case, there is no adaptation and the controller cases use the controlled invert switches.

The "Base" case (Fig. 6.16a) has a high distribution of SRAM cells with a biased duty cycle (i.e., $\delta \neq 0$; see Table 5.4), and we see bars that are not crowded near zero. Majority of these cells are responsible for storing the higher-order, low-activity bits and thus bear the largest amount of stress among the SRAM cells. For the invert MWT (Fig. 6.16g), almost all SRAM cells have the best possible duty cycle. Comparing with [29, 30, 31], the usage of bit invert MWT in the controlled invert switch architecture will also have the same aging impact. However, the aging balancing in [29, 30, 31] is achieved by employing additional hardware and architectural changes to SRAM cells. This limits the designer to only use customized SRAM memories with additional enhancements. Further, the leakage energy consumed and the area overhead by these SRAMs are much higher than the controller-based inversion technique presented in Chap. 5 because, in this case, each SRAM cell will have additional transistors associated with it.

The nibble swap MWT (Fig. 3.13c) does not perform as well as compared to the inverter and the rotator. Without the adaptive controller and Write AGU, and with only selected bits inverted ($N = 1$ and $N = 3$ in Fig. 6.16d, e), we notice that $N = 3$

Table 6.4 Video sequences and their attributes

Name	Basketball	Flowervase	Keiba	FourPeople	Johnny	ChinaSpeed	BQTerrace	Traffic	People
Attributes	832 × 480	832 × 480	832 × 480	1280 × 720	1280 × 720	1024 × 768	1920 × 1080	2560 × 1600	2560 × 1600
Resolution, motion, camera zooming/ panning, frames	Medium motion, no camera panning, a basketball drill	Luminance changes, camera zooming in, no motion	Large motion, camera panning, camera following race horses	Very low motion, no camera panning, four people around a table	Very low motion, no camera panning, a person talking to the camera	Large motion, no camera panning, car-racing video game	Large static region, camera panning	Large static region, no camera panning	Medium motion, no camera panning

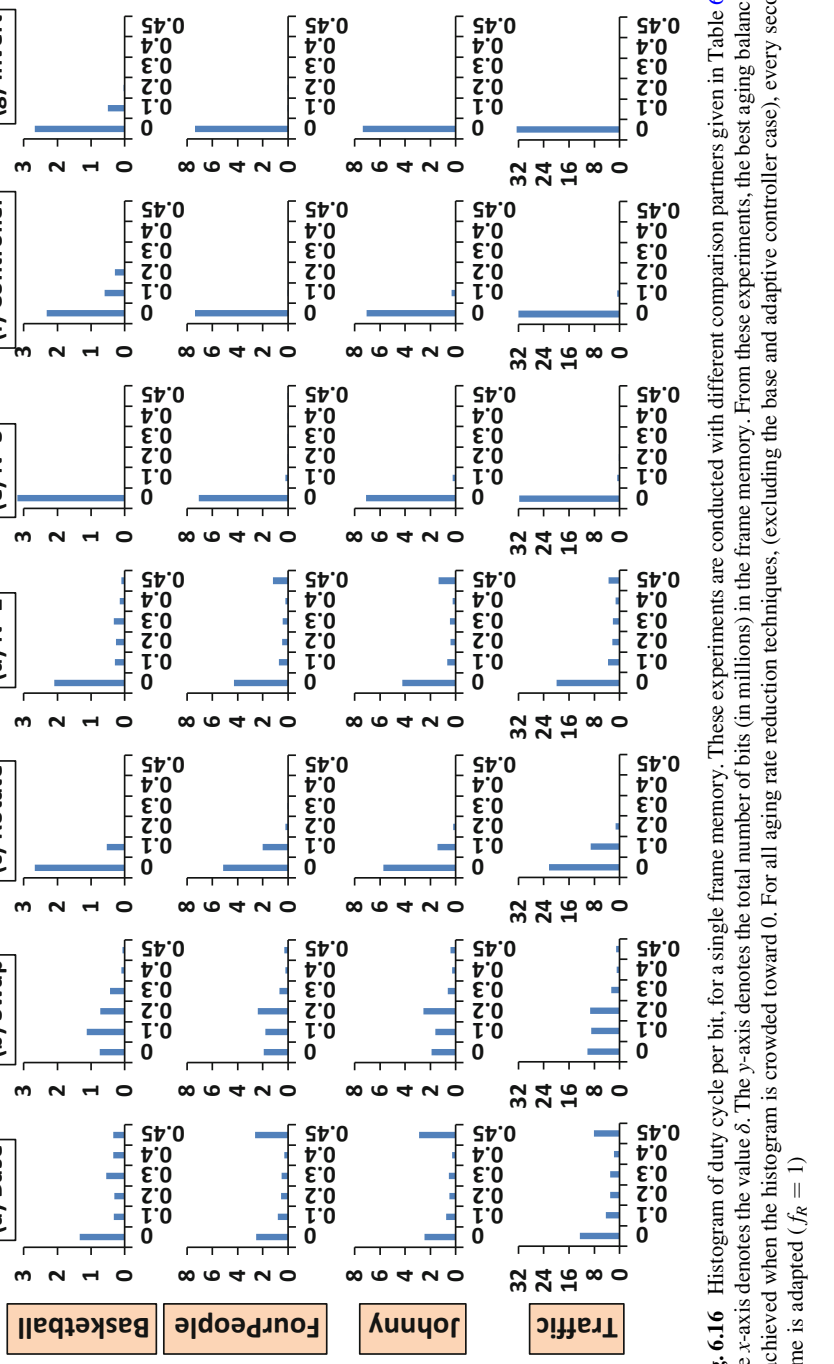


Fig. 6.16 Histogram of duty cycle per bit, for a single frame memory. These experiments are conducted with different comparison partners given in Table 6.5. The x-axis denotes the value δ. The y-axis denotes the total number of bits (in millions) in the frame memory. From these experiments, the best aging balancing is achieved when the histogram is crowded toward 0. For all aging rate reduction techniques, (excluding the base and adaptive controller case), every second frame is adapted ($f_R = 1$)

Table 6.5 Comparison partners for these experiments

MWT/MRT	Functional description
Base	No MWT or MRT or an AGU is used
Swap	Swap lower and upper nibbles of the complete frame [28]
Rotate	Rotate bits of every sample by 1 with every new frame
$N = 1$	Only inverter switch with bits 6–7 always *on*
$N = 3$	Inverter switches with bits 2–7 always *on*
Controller	Inverter switches controlled by the aging controller
Invert	Invert all bits of the complete frame [29, 30, 31]

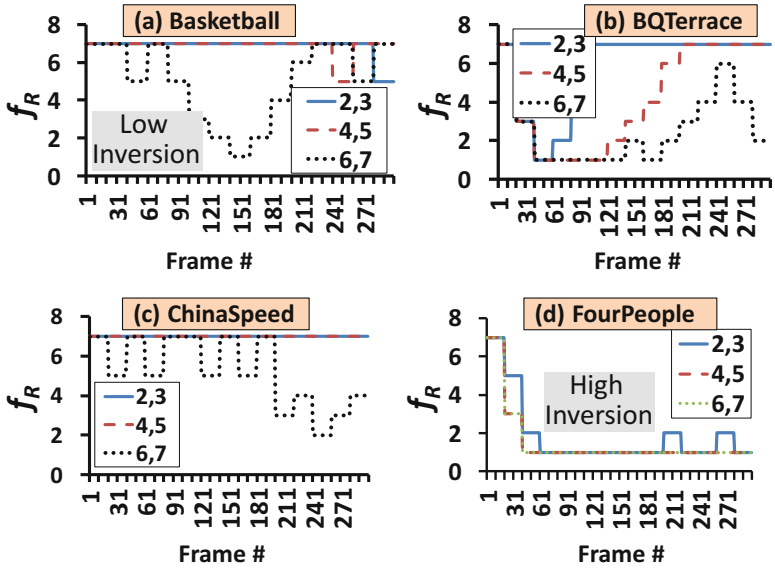
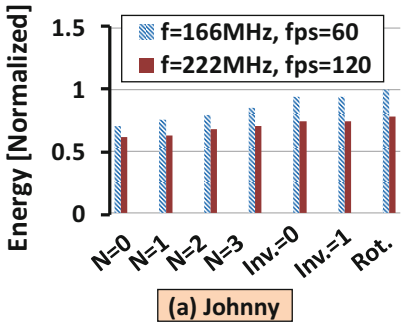
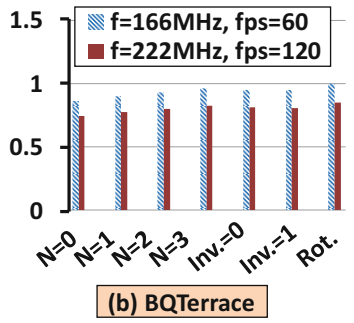


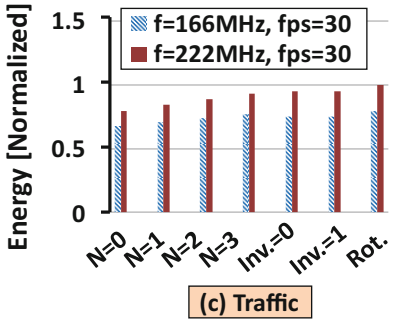
Fig. 6.17 Runtime adaptation of f_R by the adaptive aging controller. For these experiments, $n_i = 5$, $(\tau_1, \tau_2, \tau_3) = (0.75n_i, 0.50n_i, 0.25n_i)$, $F = 20$. Minimum $f_R = 1$, maximum $f_R = 8$. $s_L = (1\text{-}\varepsilon) \times s_{init}$ and $s_H = (1 + \varepsilon) \times s_{init}$ and ε for bits (3,5,7) = (1/64, 1/32, 1/8)

has almost the same impact on aging as the inverter case (i.e., inverting all the bits from 0 to 7). This is because bits 0 and 1 are self-balancing themselves while bits 2–7 are adapted with $N = 3$. For reference, see the box plot in Fig. 3.13j for bits 0 and 1. However, $N = 1$ only inverts bits 6 and 7, whereas from Fig. 3.13 g, h, we notice that bits 4 and 5 may also have highly biased duty cycles. If these bits are ignored, they can contribute to worsening SNM degradation. Still, $N = 1$ considerably balances the duty cycle, as compared to the swap case.

For experiments involving the adaptive controller (Fig. 6.16f), the parameters $(\tau_1, \tau_2, \tau_3) = (0.75n_i, 0.50n_i, 0.25n_i)$ are chosen, where $n_i = 5$ is the number of parts in which a bit plane is divided (see Fig. 5.26). The runtime variation of f_R by the adaptive controller is shown in Fig. 6.17. For majorly static sequences like







MWT/MRT	Area (in Cells)
Controller	516.23
Bit-Invert	520.91
Bit-Rotate	685.4

(d) Controller

Fig. 6.18 (**a**–**c**) Normalized energy consumption per frame of MWTs and MRTs at various frequency and FPS configurations, for different video sequences. In these figures, Inv. = 0 shows that the invert MWT/MRT is inactive while Inv. = 1 denotes active invert MWT/MRT. Rot. presents rotate MWT/MRT. (**d**) Total area (in cells) for different MWTs/MRTs

"FourPeople," more adaptation is required. Therefore, the adaptive controller tunes f_R of each inverter switch to the minimum possible f_R, in order to adaptively encounter the aging that such a sequence will induce. For high-activity sequences or sequences with camera panning (like "BQTerrace"), the controller can relax the adaptation rate. Therefore, f_R of each bit is increased, because it is not required to aggressively invert the frames. In addition, it is also noted that the aging impact of the adaptive controller with multiple frame memories is almost identical.

Figure 6.18 shows the energy consumption per frame and area of different MWTs, for different clock frequencies and FPS. This data is created by annotating the input to MWTs using ModelSim simulation. The annotated signals are then queried by the Synopsys Design Compiler to approximation average signal activity on each pin of the circuit. Afterwards, it generates the leakage and dynamic power based upon the signal activity. As noticed, the memory subsystem architecture presented here with no inverter switch active ($N = 0$) consumes the smallest amount of energy, whereas the bit-rotate MWT consumes the largest amount of energy and

area. From Fig. 6.16, we also notice that aging balancing achieved by the bit inverter and adaptive bit inversion can easily surpass the performance by bit-rotate MWT. Therefore, it is rational to use inverters in MWTs for aging resiliency instead of bit-swapping and bit-rotation logic. Further, when the technique described in [28] is applied to SRAM memories, additional overhead is encountered. For example, this technique requires testing for leading zeros and read/write of infrequently accessed memory addresses and additional information storage. On the contrary, AGU presented in this book does not require such tests because it adaptively generates addresses to cover the whole memory space in a circular fashion. This introduces activity in the otherwise low-activity cells. Compared to [32], the adaptive controller-based technique does not require additional reads and writes to the SRAM memory, which itself consumes high dynamic energy.

Moreover, depending upon the application scenario and the allowable energy budget, adaptive controller-based technique enables the application designer to select the best f_R and N configuration, suitable for their application. For example, using Fig. 6.18, a designer may achieve up to ~15% energy savings by turning off all the inverter switches at the cost of SRAM aging. Therefore, a trade-off between energy and SRAM aging can be established to select the best configuration of the application.

6.3.2.3 HCI-Induced Aging

In this section, the evaluation of the duty cycle (relevant for NBTI) and switching activity (relevant to HCI) are presented. Using different MWTs, Fig. 6.19 plots the duty cycle and toggling rate for two video sequences. As noticed, the baseline and swap MWTs incur the highest NBTI-induced aging but with small HCI-induced aging. The rotate MWT balances the duty cycle, but with enlarged maximum value as shown in the box plot: it also lifts the toggling rate of the most significant bits (as shown by the concentration around 0.5–0.6 and a high maximum value), and hence, the corresponding cells have a higher HCI-induced aging. The invert MWT encountering NBTI-induced aging considerably balances the duty cycle. However, it also increases the toggling rate due to aggressive switching of every bit, which results in a higher HCI-induced aging.

In contrast to the techniques mentioned above, the adaptive aging controller adjusts the spatial and temporal granularity of applying the aging balancing methods. Therefore, it provides improved distribution profiles for both duty cycles and switching activity across different bits.

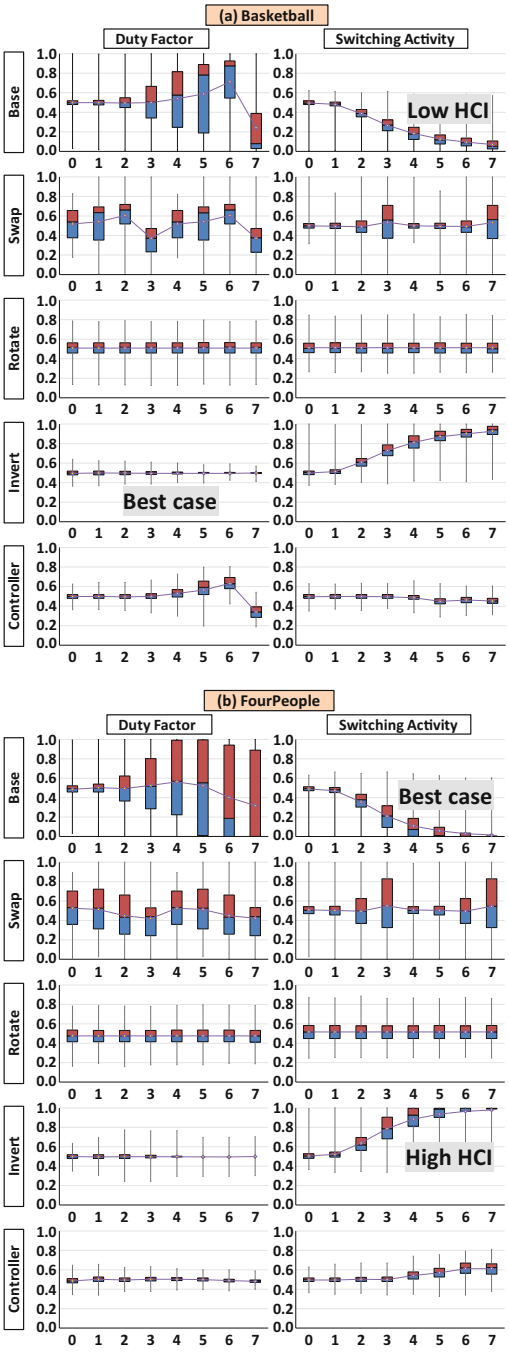


Fig. 6.19 Using different MWTs, duty cycle and toggling statistics for (**a**) "Basketball" and (**b**) "Four People" video sequence. The *x*-axis on all the graphs presents the bit planes of the memory. The *y*-axis on the duty factor plots shows the average duty cycle per bit cell. The y-axis for the toggle graphs shows the box plot of average toggling rate for each SRAM cell, i.e., the average number of times a write to a cell results in a bit flip. The duty cycle box plots should be crowded around 0.5, while the toggle box plots should be crowded toward zero for the best aging rate reduction.

References

1. Bienia, C. (2011). Benchmarking modern multiprocessors. Princeton University.
2. Fritts, J. MediaBench II. [Online]. Available: http://euler.slu.edu/~fritts/mediabench/. Accessed 6 Oct 2015.
3. Wang, Z., Liu, W., Xu, J., Li, B., Iyer, R., Illikkal, R., Wu, X., Mow, W. H., & Ye, W. (2014). A case study on the communication and computation behaviors of real applications in noc-based MPSoCs. In *Annual Symposium on VLSI*.
4. Guthaus, M. R., Ringenberg, J. S., Ernst, D., Austin, T. M., Mudge, T., & Brown, R. B. (2001). MiBench: A free, commercially representative embedded benchmark suite. In *International Workshop on Workload Characterization (WWC)*.
5. POSIX Threads for Windows – REFERENCE - Pthreads-w32. sourceware.org, [Online]. Available: https://sourceware.org/pthreads-win32/manual/. Accessed 6 Oct 2015.
6. HEVC reference software. Fraunhofer Institute, [Online]. Available: https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/. Accessed 29 Aug 2013.
7. HEVC x265 encoder. Google Code, [Online]. Available: https://code.google.com/p/x265. Accessed 29 Aug 2013.
8. 4×Six-Core AMD Opteron processor. [Online]. Available: http://www.amd.com/en-us/products/server/benchmarks/sap-sd-two-tier-four-socket. Accessed 08 Sept 2014.
9. Carlson, T., Heirman, W., & Eeckhout, L. (2011). Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation. In *SC*.
10. Li, S., Ahn, J. H., Strong, R., Brockman, J., Tullsen, D., & Jouppi, N. (2009). McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Microarchitecture*.
11. Ma, K., Li, X., Chen, M., & Wang, X. (2011). Scalable power control for many-core architectures running multi-threaded applications. In *Internation Symposium on Computer Architecture*.
12. Sharifi, A., Mishra, A., Srikantaiah, S., Kandemir, M., & Das, C. R. (2012). PEPON: Performance-aware hierarchical power budgeting for NoC based multicores. In *Parallel Architectures and Compilation Techniques*.
13. Bjontegaard, G. (2001). Calculation of average PSNR differences between RD-curves. VCEG Contribution VCEG-M33.
14. Rosas, C. Morajko, A. Jorba, J., & Cesar, E. (2011). Workload balancing methodology for data-intensive applications with divisible load. In *Symposium on Computer Architecture and High Performance Computing*.
15. Colin, A., Kandhalu, A., & Rajkumar, R. (2015). Energy-efficient allocation of real-time applications onto single-ISA heterogeneous multi-core processors. *Journal of Signal Processing Systems,* pp. 1–20.
16. Cesar, E., Moreno, A., Sorribes, J., & Luque, E. (2006). Modeling Master/Worker applications for automatic performance tuning. *Parallel Computing, 32*(7), 568–589.
17. Esmaeilzadeh, H., Blem, E., Amant, R., Sankaralingam, K., & Burger, D. (2011). Dark silicon and the end of multicore scaling. In *International Symposium on Computer Architecture*.
18. ITRS. (2011). International technology roadmap for semiconductors, 2010 update.
19. Khan, M. U. K., Shafique, M., & Henkel, J. (2014). Software architecture of high efficiency video coding for many-core systems with power-efficient workload balancing. In *Design, Automation and Test in Europe*.
20. Dong, X., Wu, X., Sun, G., Xie, Y., Li, H., & Chen, Y. (2008). Circuit and microarchitecture evaluation of 3D stacking magnetic RAM (MRAM) as a universal memory replacement. In *Design Automation Conference (DAC)*.
21. Bossen, F. (2012). Common test conditions. Joint Collaborative Team on Video Coding (JCT-VC) Doc. I1100.
22. Taiwan Semiconductor Manufacturing Company Limited. TSMC, [Online]. Available: http://www.tsmc.com/. Accessed 7 Oct 2015.

23. Design Compiler. Synopsys, [Online]. Available: http://www.synopsys.com/Tools/Implementation/RTLSynthesis/DesignCompiler/. Accessed 7 Oct 2015.
24. ModelSim – Leading Simulation and Debugging. Mentor Graphics, [Online]. Available: http://www.mentor.com/products/fpga/model/. Accessed 7 Oct 2015.
25. Khan, M. U. K., Shafique, M., & Henkel, J. CES Free Software – EnAAM, Chair for Embedded Systems (CES), KIT, [Online]. Available: ces.itec.kit.edu/EnAAM/. Accessed 5 Oct 2015.
26. Joint Collaborative Team on Video Coding (JCT-VC), ITU, [Online]. Available: http://www.itu.int/en/ITU-T/studygroups/2013-2016/16/Pages/video/jctvc.aspx. Accessed 7 Oct 2015.
27. Video Library and Tools – NSL. Network Systems Lab, [Online]. Available: https://cs-nsl-wiki.cs.surrey.sfu.ca/wiki/Video_Library_and_Tools. Accessed 7 Oct 2015.
28. Amrouch, H., Ebi, T., & Henkel, J. (2013). Stress balancing to mitigate NBTI Effects in register files. In *Dependable Systems and Networks (DSN)*.
29. Shin, J., Zyuban, V., Bose, P., & Pinkston, T. (2008). A proactive wearout recovery approach for exploiting microarchi-tectural redundancy to extend cache SRAM lifetime. In *International Symposium on Computer Architecture (ISCA)*.
30. Siddiqua, T., & Gurumurthi, S. (2010). Recovery boosting: A technique to enhance NBTI recovery in SRAM arrays. In *Annual Symposium on VLSI*.
31. Sil, A., Ghosh, S., Gogineni, N., & Bayoumi, M. (2008). A novel high write speed, low power, read-SNM-Free 6T SRAM cell. In *Midwest Symposium on Circuits and Systems*.
32. Wang, S., Jin, T., Zheng, C., & Duan, G. (2012). Low power aging-aware register file design by duty cycle balancing. In *Design, Automation and Test in Europe (DATE)*.

Chapter 7
Conclusion and Future Outlook

Targeting multimedia systems under high throughput, resource and power constraints, this book discusses efficient software-/application-level techniques and hardware-/architectural-level designs for the multimedia (specifically video) systems. Mainly, the aim of the techniques discussed in this book is to maximize the throughput-per-watt metric of the system while considering some modern design challenges and methodologies. The challenges addressed in this book include parallelization of multimedia applications on possibly heterogeneous systems, load balancing on many-core and customized nodes, resource (number of cores and power) budgeting, and efficient design of the multimedia system's memory architecture. In a broader perspective, these problems can collectively represent the power wall or dark silicon challenge for the next-generation video processing systems.

In this chapter, a summary of the content discussed here is given. At the end, we will carry out a brief prognosis of future extensions related to the complexity and power efficiency in relation to multimedia applications.

7.1 Software-Level Techniques

The techniques for parallelization of video systems presented in this book address the throughput demands of the video processing system and attributes of the underlying hardware while being power efficient and providing high output video quality. These techniques determine a proper *compute configuration*. Video frames are divided into tiles, and these tiles are packed and processed on the underlying compute nodes, depending upon the workload characteristics and properties of the

The authors would like to point out that this work was carried out when all the authors were in Department of Computer Science, Karlsruhe Institute of Technology, Karlsruhe, Germany.

M.U.K. Khan et al., *Energy Efficient Embedded Video Processing Systems*,
DOI 10.1007/978-3-319-61455-7_7

nodes. For this purpose, uniform and non-uniform tiling techniques are presented [1, 2]. Uniform tiling assigns a video tile to a core for processing (i.e., there is a one-to-one correspondence between nodes and tiles), while non-uniform tiling may assign multiple tiles to a single compute node. Both these techniques balance the workload on the nodes. Further, depending upon the workload of the associated tiles, the voltage-frequency levels of the nodes are set to sustain the throughput demands (FPS) at reduced power consumption while still providing high output video quality. The voltage-frequency levels are determined using a frequency estimation model. In case the frequency estimation model is inaccurate (or not derived), runtime adjustment/derivation of frequency model constants is carried out using a recursive least squares (RLS) filter.

Furthermore, these techniques also derive *application configuration* that will select the application's workload configuration (by tuning application parameters) while meeting a quality constraint set by the user. As a proof of concept, this book presents HEVC application configuration by selecting HEVC parameters and bargain computational complexity with the output video quality. Moreover, techniques to appropriately map these parameters reduce the degradation of output video quality. For example, the reduction in number of intra-angular predictions used for HEVC encoding presented here results in 18 to 44% on average timesaving against a comparison partner [3]. Further, the depth of HEVC block subdivision for RDO is reduced, and hence up to ~57% time savings are obtained, with negligible video quality loss (-0.048dB reduction in PSNR compared to −0.118dB by [4]). In short, employing the HEVC intra-angular and depth adaptation for application configuration on uniform tiling-based compute configuration results in ~42% power savings and ~19.2% power savings compared to [5]. Additionally, the non-uniform tiling results in ~7.8% more power reduction compared to the uniform tiling.

To exploit the advantages of heterogeneity, techniques for *workload distribution and balancing on heterogeneous systems* are also discussed in this book. The compute configuration that selects the compute nodes and their frequencies aims to increase the throughput-per-watt of the system. For this, the workload distribution technique considers the power and computational efficiency of the underlying compute nodes (soft-cores and hardware accelerators). Here, we also discuss three different efficiency indices for workload distribution, and an optimization problem is derived, which can be solved using Nelder-Mead heuristic. Compared to [6], which also derives an optimization problem and then proposes a heuristic, the technique discussed in this book can result in up to ~64% increase in the throughput-per-watt of the heterogeneous system. Note that the technique in [6] allocates subtasks/video tiles to the cores in bin-packing fashion, and the core that results in minimum power is allocated the next tile.

Further, considering multiple mixed multithreaded tasks/applications running on the same hardware platform (e.g., as is the case with video multicasting), the *resource budgeting* problem becomes more challenging. The resource budgeting techniques presented here allocate compute configurations (i.e., cores and frequency/power) in a hierarchical manner, from individual clusters down to the

individual cores. Each task gets a budget of the computing cores and total system power (TDP). This budgeting considers the compute characteristics of the tasks, the available frequency levels of the cores, and the complexity of the tasks along with their resource history. Once the number of cores and power allocated to a cluster is determined, the power is divided among the cores of the cluster and regulated at runtime. The number of cores and power budgeted to the clusters is also adapted. When the resource budgeting technique presented here is tested for HEVC multi-tasking, it results in up to ~18% additional throughput against a [7], for the same power utilization of the system.

7.2 Hardware-Level Techniques

The software-level techniques discussed in this book are applicable to multi-/many-core systems, possibly heterogeneous and with hardware accelerators. To increase the throughput and power efficiency of the video systems, the architectural enhancements discussed in this book target hardware accelerator design, workload offloading mechanisms to these hardware accelerators, and the memory subsystem of the system.

In this regard, efficient architectures to support *video I/O and communication* among computing nodes and the hardware accelerators are discussed. The efficient I/O encompasses video input pipeline (VIP), video frame read from and write to the external memory, and architectural enhancement to provide block-based video data to the video processing system, which can be a soft-core, coprocessor, or a hardware accelerator. The complete design is implemented and functionally verified on a FPGA (see Appendix C for details). For communication among nodes, a communication logic is proposed, which employs a register file-based custom interface. This interface exchanges metadata (e.g., memory pointers) and status signals ("start," "done," etc.) among the nodes. This interface is implemented on a multi-core Nios II-based FPGA system, which can encode the video frames in HEVC format (see Sect. B.4 for more details).

To utilize the high power efficiency of the hardware accelerators, an optimization problem is derived, and a Nelder-Mead-based heuristic is presented to allocate the *shared hardware accelerator* for processing subtasks of concurrently running threads/applications in a round-robin manner. This heuristic determines the frequencies of the compute cores, and the fraction of the subtasks of each application that must be offloaded to the hardware accelerator to fulfill its throughput requirement, while reducing the power consumption of the complete system. Additionally, the technique presented here let the soft-cores to be powered *off* once they offload their subtasks to the accelerator. This helps in reducing the temperature and hence addresses the issues related to dark silicon. Moreover, the implementation details of a functionally verified multicasting system based on H.264/AVC are also given. The hardware accelerator is shared among different video encoders in a round-robin fashion. For sharing purposes, custom hardware scheduler and reschedulers are

used. The functional verification of this design is carried out using an FPGA (see details in Appendix C).

Moreover, this book also discusses *architectures for efficient hardware accelerators*, specifically for video encoders (H.264/AVC and HEVC). For H.264/AVC, a low-latency intra-encoding loop is designed, which addresses some dependencies to allow for high throughput of the encoding loop. Firstly, an area-efficient design of the transform module of H.264/AVC including (I)DCT and (I)Q is presented for both AC and DC paths. This design integrates a HT lookahead buffer that calculates the HT input coefficients at the 4×4 reorder stage, instead of generating these coefficients by the DCT module in the AC path. Thus, this module effectively decouples the AC and DC paths. Both (I)HT and (I)DCT architectures are merged and folded to reduce area. The one-cycle latency due to folding the (I)HT/(I)DCT is exploited in fusing the Q/IQ stages together, thereby reducing the number of multipliers and barrel shifters. A hardware architecture of the H.264/AVC mode decision circuit is also given, along with an efficient technique to determine intra-mode that will be selected by the mode decision module with high probability. This technique uses a hardware-based edge extractor, which determines the precedence order of the prediction modes. On the other hand, other methods like open-loop (OL) algorithm [8] use the original pixels for estimating the mode. This way, in case the cycle budget of the encoding loop is reduced (due to increased resolution or I/O stalls), the number of modes tested can be reduced while scanning the precedence order in a sequence. The encoding-loop architecture presented in [9] utilizes 6.89 kilo gates per pixel and 63.5 mW per pixel, whereas the architecture explained in this book uses 6.65 kilo gates per pixel and 61.77 mW per pixel. Further, a hardware and software collaborative architecture is presented, which can use the software for control and hardware for computations. The hardware unit is composed of multiple sub-accelerators, and, therefore, it is termed as a distributed hardware module. Each sub-accelerator can be turned *off* independently and thus save power. This results in up to ~42% energy savings for HEVC intra-encoding.

As discussed, the hardware accelerators usually use on-chip scratchpad memories for fast computations. This book discusses the alternative solutions of using *hybrid scratchpad memories* for increased power efficiency. Specifically, MRAM-based NVMs in conjunction with SRAM-based VMs are presented here. Large MRAM buffers can be used for storing the video frames, while small SRAM buffer is used to (a) store the immediately fetched data from the external memory and (b) act as input and output buffers of the accelerators. The technique presented here reduces the power consumption related to external memory access and leakage by adaptively turning *on* the normally *off* MRAM memory sectors. An unsupervised learner (SOM) learns the turning *on*/*off* pattern. Compared to the widely utilized search window updating technique [10], the hybrid scratchpad-based methodology results in up to ~43% power savings.

Moreover, to have reliable operation of video processing system over long deployment durations, this book analyzes the aging profiles of SRAM-based video buffers. Based upon this analysis, power-efficient *SRAM anti-aging circuits* are discussed. Unlike other methodologies [11, 12, 13] for mitigating SRAM aging,

the techniques presented here modify video data on the fly using a write transducer, while the data is being written to the SRAM memory. Other techniques for aging rate reduction usually are (a) employing reading data from SRAM buffers, modifying that data and writing it back to the SRAM, or (b) designing custom SRAM cells, which results in high leakage power. Furthermore, when the application reads this modified data, the data is again transformed to its original state using read transducers. The microarchitecture of write and read transducers is designed such that the energy consumption is reduced. Specifically, certain bits of the video samples are adapted, while others are left unchanged. The aging controller takes this decision in spatial and temporal locality. Moreover, the starting address for writing every new video frame is changed, and therefore, the data of the 6T SRAM cells is self-adapted, even if no transducer is used.

7.3 Further Improvements

The technical evaluations of the techniques put forth by this book at software and hardware abstraction levels demonstrate that for high power efficiency of a video system, one must jointly consider the codesign space of the application and the hardware. Comparing with other state-of-the-art techniques, considerable amount of complexity and power and area improvements are achieved. However, there are some research frontiers, which are not explored by this book. These directions can also be considered as effective means for resource and power efficiency. Some of these directions can be orthogonally added to algorithm/architectures presented here, with minimal effort. In the following, a few of these possible future enhancements are outlined.

7.3.1 Approximate Computing

Approximate computing [14, 15, 16, 17, 18] exploits inherent resiliency of applications like audio/image/video processing to gain power savings. Since for these applications, multiple outcomes are acceptable, therefore, the mechanism that results in acceptable output quality with the best power efficiency is selected for processing the workload. These mechanisms can be incorporated at the software level or at hardware level. At the software level, techniques like "iteration skipping" (similar to the one presented in Sect. 4.3) can be used. However, the hardware-level functional approximation of basic circuits (e.g., adders) has gathered considerable interest over the last decade. The main idea is that by tolerating errors at the microarchitectural level, one can employ circuits with reduced accuracy (i.e., output is approximated) and thus save energy. Mainly, the focus of recent research about approximate computing has been to optimize small kernels with approximation and to implement them as hardware accelerators. Approximation

mechanisms are employed in basic arithmetic units like adders, multipliers, etc. For example, [19, 20] explored transistor-level addition approximations. The authors of [21] proposed a runtime accuracy-configurable adder (ACA), while [22] implements efficient but approximate multipliers.

These approximate circuits can be used to design power-efficient hardware architectures for the video systems. For example, the SAD unit (see Fig. 2.8) can be designed using approximate adders, the image/video processing filters can use approximate multipliers. Moreover, the hardware accelerator sharing technique presented in this book (Sect. 5.2) can additionally implant a relationship to account for the output quality (or approximation level), and the hardware accelerator itself can consist of multiple approximation units. Additionally, distributed hardware accelerators can employ approximated building modules and activate these modules at runtime, depending upon the application quality requirement. Similarly, the workload balancing and distribution techniques for heterogeneous systems can also utilize the approximation levels of these accelerators. This way, the power efficiency increases because approximate accelerators can process the workload even faster.

7.3.2 GPU-Based Acceleration

A GPU is a heterogeneous chip multiprocessor that can incorporate hundreds of parallel executing threads and requires a specialized code. GPUs are universally employed for image rendering. GPUs are also well suited for video output pipelines, like the ones used in videogames. They are also used in applications benefiting from large degree of parallelism, where a small compute kernel is employed for pixel-level computations [23, 24, 25, 26]. Another concept popular nowadays is general-purpose graphics processing unit (GPGPU) that can also perform non-specialized, mostly controlled execution, typically conducted by a CPU.

In many cases, a GPU can be considered as a completely separate compute node, and the workload balancing and distribution techniques (especially for heterogeneous nodes) of this book can be used. Specifically, the scientific challenge to address will be division of a task into subtasks, such that the subtasks are allocated to the GPU (or GPGPU) in a manner that increases the power efficiency of the system. Another central goal of workload distribution should be the power management of GPU, because GPU is one of the major power consumers. In addition, implementing the workload balancing on a heterogeneous system, employing multiple GPUs, FPGAs, and soft-cores, via OpenCL [27], is both a scientific and engineering challenge.

7.3.3 Reliability and Workload Management

The software and hardware layers of the multimedia system must be adapted in case they are used under soft- and/or hard-errors scenarios. Further, the power-efficient

techniques presented in this book must also adapt. For example, to process a given task, multiple compute nodes are used with dual or triple modular redundancy (DMR or TMR). This will increase the power consumption of the system [28], and, therefore, the resource budgeting techniques presented here must adapt. That is, the number of cores allocated to process a task and the power allocated to the task must consider the impact of reliability-aware processing. Likewise, the modules of the image/video processing application, which need reliability and protection, must be identified. This is because usually, image/video applications are inherently resilient to errors in some parts of the computation pipeline, while some other parts (e.g., entropy coder) are extremely sensitive to errors.

7.3.4 Generalization

In addition, applications other than multimedia (databases, radar, signal processing for wireless communications, biotechnology, etc.) can be tested and evaluated by the parallelization, workload balancing, compute and application configurations, and resource budgeting techniques presented here. Similarly, one can design and integrate hardware units for these types of applications using the methodologies presented in this book (proposed in Sects. 5.1.4 and 5.3.2). Note that the techniques outlined in this book can be easily extended to multidimensional video processing paradigms. For example, 3D-HEVC [29] and MVC [30] used for encoding 3D video streams, scalable video coding (SVC) [31], and multiple description coding (MDC) [32] involve multiple layers of video content. Therefore, parallelization and workload balancing techniques can be extended to incorporate the additional layers or dimensions of video (i.e., the view or the layer), because the additional dimension can be added as a separate dimension to the optimization problems.

Mostly, this book targets the application and hardware layers for resource budgeting and optimizations. However, kernel-level optimizations and efficient utilization of available resources may result in high system efficiency. Moreover, it will be challenging to extend these techniques for reliable, hard real-time systems involving strict guarantees of fulfilling the timing constraints.

References

1. Khan, M. U. K., Shafique, M., & Henkel, J. (2014). Software architecture of high efficiency video coding for many-core systems with power-efficient workload balancing. In *Design, Automation and Test in Europe*.
2. Shafique, M., Khan, M. U. K., & Henkel, J. (2014). Power efficient and workload balanced tiling for parallelized high efficiency video coding. In *International Conference on Image Processing*.

3. Jiang, W., Mal, H., & Chen, Y. (2012). Gradient based fast mode decision algorithm for intra prediction in HEVC. In *International Conference on Consumer Electronics, Communications and Networks*.
4. Sun, H., Zhou, D., & Goto, S. (2012). A low-complexity HEVC Intra prediction algorithm based on level and mode filtering,. In *International Conference on Multimedia and Expo (ICME)*.
5. Rosas, C. Morajko, A. Jorba, J., & Cesar, E. (2011). Workload balancing methodology for data-intensive applications with divisible load. In *Symposium on Computer Architecture and High Performance Computing*.
6. Colin, A., Kandhalu, A., & Rajkumar, R. (2015). Energy-efficient allocation of real-time applications onto single-ISA heterogeneous multi-core processors. *Journal of Signal Processing Systems,* pp. 1–20.
7. Ma, K., Li, X., Chen, M., & Wang, X. (2011). Scalable power control for many-core architectures running multi-threaded applications. In *Internation Symposium on Computer Architecture*.
8. Fonseca, T. A., Liu, Y., & Queiroz, R. L. D. (2007). Open-loop prediction in H.264 / AVC for high definition sequences. In *SBrT*.
9. Kuo, H. C., Wu, L. C., Huang, H. T., Hsu, S. T., & Lin, Y. L. (2011). A low-power high-performance H.264/AVC intra-frame encoder for 1080pHD video. *IEEE Transactions on Very Large Scale Integrated Systems (TVLSI), 19*(6), 925–938.
10. Chen, C., Huang, C., Chen, Y., & Chen, L. (2006). Level C+ data reuse scheme for motion estimation with corresponding coding orders. *IEEE Transactions on Circuits and Systems for Video Technology, 16*(4), 553–558.
11. Shin, J., Zyuban, V., Bose, P., & Pinkston, T. (2008). A proactive wearout recovery approach for exploiting microarchi-tectural redundancy to extend cache SRAM lifetime. In *International Symposium on Computer Architecture (ISCA)*.
12. Siddiqua, T., & Gurumurthi, S. (2010). Recovery boosting: A technique to enhance NBTI recovery in SRAM arrays. In *Annual Symposium on VLSI*.
13. Sil, A., Ghosh, S., Gogineni, N., & Bayoumi, M. (2008). A novel high write speed, low power, read-SNM-Free 6T SRAM cell. In *Midwest Symposium on Circuits and Systems*.
14. Shin, D., & Gupta, S. (2010). Approximate logic synthesis for error tolerant applications. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*.
15. Venkataramani, S., Sabne, A., Kozhikkottu, V., Roy, K., & Raghunathan, A. (2012). Salsa: Systematic logic synthesis of approximate circuits. In *Design Automation Conference (DAC)*.
16. Venkataramani, S., Roy, K., & Raghunathan, A. (2013). Substitute-and-simplify: A unified design paradigm for approximate and quality configurable circuits. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*.
17. Ranjan, A., Raha, A., Venkataramani, S., Roy, K., & Raghunathan, A. (2014). ASLAN: Synthesis of approximate sequential circuits. In *Design, Automation and Test in Europe Conference and Exhibition (DATE)*.
18. Chakrapani, L. N., Muntimadugu, K. K., Lingamneni, A., George, J., & Palem, K. V. (2008). Highly energy and performance efficient embedded computing through approximately correct arithmetic: A mathematical foundation and preliminary experimental validation. In *international conference on Compilers, architectures and synthesis for embedded systems*.
19. Gupta, V., Mohapatra, D., Park, S. P., Raghunathan, A., & Roy, K. (2011). IMPrecise adders for low-power approximate computing. In *International Symposium on Low Power Electronics and Design (ISLPED)*.
20. Gupta, V., Mohapatra, D., Raghunathan, A., & Roy, K. (2012). Low-power digital signal processing using approximate adders. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), 23*(1), 124–127.
21. Khang, A. B., & Kang, S. (2012). Accuracy-configurable adder for approximate arithmetic designs. In *Design Automation Conference (DAC)*.

22. Kulkarni, P., Gupta, P., & Ercegovac, M. (2011). Trading accuracy for power with an under-designed multiplier architecture. In *International Conference on VLSI design*.
23. Momcilovic, S., Ilic, A., Roma, N., & Sousa, L. (2014). Dynamic load balancing for real-time video encoding on heterogeneous CPU+GPU systems. *IEEE Transactions on Multimedia, 16* (1), 108–121.
24. Momcilovic, S., Roma, N., & Sousa, L. (2013). Exploiting task and data parallelism for advanced video coding on hybrid CPU + GPU platforms. *Journal of Real-Time Image Processing,* pp. 1–17.
25. Cuomo, S., Michele, P. D., & Piccialli, F. (2014). 3D data denoising via nonlocal means filter by using parallel GPU strategies. In *Computational and Mathematical Methods in Medicine*.
26. Mittal, S., & Vetter, J. S. (2015). A Survey of CPU-GPU Heterogeneous Computing Techniques. *ACM Computing Surveys, 47*(4), 1–35.
27. OpenCL – The open standard for parallel programming of heterogeneous systems. Khronos, [Online]. Available: https://www.khronos.org/opencl/. Accessed 12 Oct 2015.
28. Salehi, M., Tavana, M. K., Rehman, S., Florian Kriebel, M. S., Ejlali, A., & Henkel, J. (2015). DRVS: Power-efficient reliability management through dynamic redundancy and voltage scaling under variations. In *International Symposium on Low Power Electronics and Design (ISLPED)*.
29. Muller, K., Schwarz, H., Marpe, D., Bartnik, C., Bosse, S., Brust, H., Hinz, T., Lakshman, H., Merkle, P., Rhee, F., Tech, G., Winken, M., & Wiegand, T. (2013). 3D high-efficiency video coding for multi-view video and depth data. *EEE Transactions on Image Processing, 22*(9), 3366–3378.
30. Vetro, A., Wiegand, T., & Sullivan, G. (2011). Overview of the stereo and multiview video coding extensions of the H.264/MPEG-4 AVC standard. *Proceedings of the IEEE, 99*(4), 626–642.
31. Schwarz, H., Marpe, D., & Wiegand, T. (2007). Overview of the scalable video coding extension of the H.264/AVC standard. *IEEE Transactions on Circuits and Systems for Video Technology, 17*(9), 1103–1120.
32. Goyal, V. (2001). Multiple description coding: compression meets the network. *IEEE Signal Processing Magazine, 18*(5), 74–93.

Appendices

Appendix A: Pseudo-codes

A.1 Compute and Application Configuration

ParallelVideoProcessing ( ):
Input:
Available number of cores r_{tot}; Allowable frequency set of the core $\boldsymbol{f}_{set}$; Video frame dimensions $w \times h$; Frame rate (in frames per second) f_p;
Output:
Processed video;

```
1.  NewTiling ← true;
2.  while(AreFramesRemaining){
3.      if(λ frames processed And NT( )){
4.          NewTiling ← true;}                          //New compute configuration selection
5.      if(NewTiling){
6.          (k_tot, f_m, α_m) ← GenerateTiles( ); NewTiling ← false;}
7.      ∀k ∈ {1 to k_tot tiles}{                        //For all parallel tiles
8.          if(EpochFinished){
9.              α_k ← AppConfig(k, α_m);
10.             f_k ← AllocEpochFreq(k, α_k );}
11.         ChangeCoreFreq(k);
12.         t_k ← ProcessTile(k, α_k );                 //Video Tile processing
13.         if(NewEpochStart){
14.             FreqModelTuning(k, f_k, α_k );}
15. }
```

Algorithm 1 Parallel processing and workload management of a video application

The authors would like to point out that this work was carried out when all the authors were in Department of Computer Science, Karlsruhe Institute of Technology, Karlsruhe, Germany.

M.U.K. Khan et al., *Energy Efficient Embedded Video Processing Systems*,
DOI 10.1007/978-3-319-61455-7

A.2 Compute Configuration

GenerateVideoTiles ( ):

Input:

Available number of cores r_{tot}; Available frequency set $\boldsymbol{f}_{set}$; Minimum frequency of a core f_{min}; Maximum frequency of a core f_{max}; Image dimensions $w \times h$; Quantization Parameter QP; Frame rate (in frames per second) f_p; Workload matrix A;

Output:

Total tiles/threads/cores k_{tot}; Maximum frequency of each core $f_{k,m}$; Initial $\boldsymbol{\alpha}_{k,m}$ per tile;

1. AreCoreAvail ← true; $k'_{tot} \leftarrow 1$; //*Initial configuration*
2. while(AreCoreAvail){
3. $k_{tot} \leftarrow \text{TileMap}[k_{tot}]$; //*Actual number of tiles*
4. $\hat{c}_{k,\boldsymbol{\alpha}} \leftarrow g(\boldsymbol{\alpha}_{max}, k_{tot}, W \times H)$; //*Estimate number of cycles per block*
5. $\forall k, k \in \{0, \cdots, k_{tot}\}\{$ //*For each tile/thread/core*
6. $\boldsymbol{\alpha}_k \leftarrow \max(A)$; //*Start with maximum workload for the tile*
7. $\hat{f} \leftarrow n_{tot} \times \hat{c}_{ktot,\boldsymbol{\alpha}} \times f_p$; //*Estimate frequency of the core*
8. $f_{k,m} \leftarrow \text{Quantize}(\hat{f}, \mathbf{f}_{set})$; //*Supported frequency*
9. if $(f_{k,m} \geq f_{max})\{$ //*If the required frequency is too high*
10. while$(\boldsymbol{\alpha}_k \geq \min(A))\{$ //*Test all workload configurations*
11. $\hat{c}_{k,\boldsymbol{\alpha}} \leftarrow g(\boldsymbol{\alpha}_k, k_{tot}, W \times H)$;
12. $f_{k,m} \leftarrow \text{Quantize}(n_k \times \hat{c}_{k,\boldsymbol{\alpha}} \times f_p, \mathbf{f}_{set})$;
13. if $(f_{k,m} \leq f_{max})\{\boldsymbol{\alpha}_{k,m} \leftarrow \boldsymbol{\alpha}_k$; break;$\}$
14. $\boldsymbol{\alpha}_k \leftarrow \text{NextLowerWorkload}(\boldsymbol{\alpha}_k, A);\}\}\}$
15. if $(k_{tot} = r_{tot})\{$ //*All available cores are used*
16. $\forall k, k \in \{0, \cdots, k_{tot}\}\{$
17. if $(\boldsymbol{\alpha}_{k,m} < \min(A))\{\text{WarnExit}(\,);\}\}$ //*Workload still not supportable*
18. AreCoreAvail ← false;$\}$
19. elseif $(\boldsymbol{\alpha}_{k,m} = \max(A)\ \forall k, k \in \{0, \cdots, k_{tot}\})\{$
20. return;$\}$ //*Maximum workload supportable*
21. $k'_{tot} \leftarrow k_{tot} + 1;\}$ //*Increase cores and repeat*

Algorithm 2 Total subtasks/tiles/threads/cores, frequency, and workload initialization for video processing

AllocFrequencyOfCore ( ):
Input:
Recommended core frequency f_k; Allowable frequency set f_{set}; Frame rate (in frames per second) f_p; Epoch size in number of frames z;
Output:
Frequency for all cores in one epoch f;

1. $f_{k,h} \leftarrow$ GetNeighborHighFreq(f_k, f_{set});
2. $f_{k,l} \leftarrow$ GetNeighborLowFreq(f_k, f_{set});
3. for(i in 0 to z-1){
4. $\xi \leftarrow (f_p - i) \times f_{k,l} + i \times f_{k,h}$;
5. if($\xi \geq z \times f_k$){break;}
6. }
7. $\forall_{j \in \{0 \text{ to } i\}} f(j) \leftarrow f_{k,l}$; //*Allocate lower frequency*
8. $\forall_{j \in \{i+1 \text{ to } z-1\}} f(j) \leftarrow f_{k,h}$; //*Allocate higher frequency*

Algorithm 3 Epoch frequency allocation algorithm

A.3 PU Map (PUM) Generation for HEVC

PUMGenerator ( ):
Input:
Video frame block *lcu*; CTU size b_w, Block merge variance threshold v_{th};
Output:
PU map PUM;

1. $\forall$ **i** $\in$ **4×4** of **CTU** {
2. $[sb_i.x, sb_i.y] \leftarrow$ GetLocationWithinCTU(4×4);
3. sb_i.PUsize $\leftarrow 4$; //*Initially, all PUs are set to asize of 4×4*
4. $sb_i.m \leftarrow$ GetMean($CTU_{4\times4}$); $sb_i.v \leftarrow$ GetVariance($CTU_{4\times4}$);
5. PUM($sb_i.x, sb_i.y$) = sb_i.PUsize; }
6. $blk_{size} \leftarrow 4$;
7. **while(**$blk_{size} < b_w$**)** {
8. $\forall_{h \in \{\text{sb of } blk_{size} \times blk_{size}\}}$ { //*For all sub-blocks of the current size*
9. $\forall_{i \in \{4 \text{ Neighbors of } blk_{size} \times blk_{size}\}}$ { //*Four sub-blocks of the same size in the neighborhood*
10. IsMerge $\leftarrow$ true;
11. $V_i, m_i \leftarrow$ CombineVariance(V_i, m_i, sb_i);
12. **if(**$V_i > v_{th}$ **or** $sb_i.v > v_{th}$**)** IsMerge $\leftarrow$ false; break; }
13. **if(**IsMerge = true**)** {
14. $sb_h.m \leftarrow m_i$; $sb_h.v \leftarrow V_i$;
15. sb_h.PUsize $\leftarrow blk_{size} \times 2$; //*Merge the four sub-blocks to form a bigger block*
16. PUM($sb_h.x, sb_h.y$) = sb_h.PUsize; } }
17. $blk_{size} \leftarrow blk_{size} \times 2$; } //*Start with the next sub-block size*
18. **return (**PUM**);**

Algorithm 4 PU map (PUM) generation algorithm for HEVC

A.4 Workload Balancing on Heterogeneous Nodes

LoadBalancingOnHeterogeneousNodes ( ):

Input:

Task deadline $t_{i,max}$, Total number of subtasks per task n_i; Number of available nodes r_{tot}; Power vs. frequency profile of all nodes $p(f)$; Efficiency indices of all nodes ϕ; Cycles for a subtask for all the nodes $\boldsymbol{c}$;

Output:

Total number of required nodes k_{tot}; Frequency of all the nodes $\boldsymbol{f}$;

```
1.  g_φ ← SortNodesDesc(φ);                          //Sort nodes according to their efficiency φ
2.  k_tot ← 1; k ← 0;
3.  while(k_tot ≤ r_tot){
4.    ∀k∈{0 to k_tot -1 nodes}{
5.      f_k ← f_k,min;                               //Test with minimum. frequency
6.      n_i,k ← DistLoad(n_i, φ);}                   //Load distribution formula
7.    ThrptMet ← true;                               //Assume throughput is met
8.    ∀k∈{0 to k_tot -1 nodes}{
9.      t_i,k ← ProcessTimeOnNode(c_i,k, n_i,k, f_i,k); //Time spent on processing the load
10.     NodeThrptMet ← true;                         //Assume node's throughput requirement met
11.     if(t_i,k > t_i,max){                         //Throughput not met
12.       NodeThrptMet ← false;
13.       ∀f_i,j∈{f_k,min to f_k,max of node k}{      //Test increasing frequency
14.         t_i,j ← TimeOnNode(c_i,j, n_i,j, f_i,j); //Time spent on processing the load
15.         if(t_i,j > t_i,max){NodeThrptMet ← true; break;}}
16.       if(NodeThrptMet = false){ThrptMet ← false;}}
17.   }                                              //All k_tot nodes tested with every possible frequency
18.   if(ThrptMet = false){                          //Throughput not satisfied
19.     k_tot ← IncrementNode(g_φ);                  //Introduce an additional node
20.   else{break;}                                   //Configuration found, exit
21. }
```

Algorithm 5 Workload distribution and balancing algorithm on heterogeneous nodes

A.5 Resource Budgeting for Concurrent Applications

AdaptiveResourceAndPowerAllocation ( ):
Input:
Available number of cores k_{tot}; Number of applications/tasks/clusters a_{tot}; Available power (TDP) budget p_{tot}; Deadline per task $t_{i,max}$;
Output:
Resource and Power budgeting to all the applications.

```
1.  do{
2.    ∀i∈{0,···,atot−1} ki ← EstimateReqCores(ti,max );   //Estimate cores for every cluster
3.    ∀i∈{0,···,atot−1} pi ← AllocatePower( );            //Distribute power to all clusters
4.    if(IsConfigChanged){∀i∈{0,···,atot−1}{
5.      ∀j∈{0,···,ki −1} pi,j ← CorePowerAlloc( );}}      //Allocate power to every core
6.    while(1){
7.      ∀i∈{0,···,atot−1}{                                //For every cluster, adapt resources
8.        GetNewTask( );
9.        ∀j∈{0,···,ki −1} ProcessSubtask( );             //Process the subtask
10.       no,i ← TaskOffset( );                           //Determine the offset from the deadline
11.       xo,i ← UpdateAppOffset(no,i );
12.       FrequencyAdjust( );                             //Adjust frequencies of the cores
13.       if(IsEpochExpired) goto Line-16;                //Start new epoch
14.     }                                                 //Task finished
15.   }                                                   //All tasks within the corefinished
16. }while(AreTasksRemaining);
```

Algorithm 6 Resource distribution (cores and power) among concurrently running applications

FrequencyAdjust ( ):
Input:
Total number of cores in the cluster/application k_i; Per core offset $n_{o,i,j}$; Core frequency set f_{set}={$f \mid f \in$ allowable core frequencies}; Total allocated power for the application p_i;
Output:
Frequency of all the cores $f_{i,j}$

```
1.  n_{o,i,list} ← SortThreads_n_{o,i,j}( );          //Sort threads in descending order
2.  z_max ← 0; z_min ← k_i − 1;                       //Indices of the sorted list
3.  if(IsFrequencyAdjustmentAllowed){
4.    while(n_{o,i,list}(z_max) > 0 AND n_{o,i,list}(z_max) > n_{o,i,list}(z_min)){
5.      T_high ← GetThreadID(n_{o,i,list}, z_max);
6.      T_low ← GetThreadID(n_{o,i,list}, z_min);
7.      if(PowerExchange(f_set[T_low.f − 1], f_set[T_high.f + 1]) ≤ p_i){
8.        T_low.f ← f_set[T_low.f − 1];             //Decrease frequency of less critical core
9.        T_high.f ← f_set(T_high.f + 1);}          //Increase frequency of the critical core
10.     z_max ← z_max + 1; z_min ← z_min − 1;
11. }
```

Algorithm 7 Frequency adaptation among cores of a cluster after processing a task

A.6 Cost Function for Hardware Offloading

NelderMeadCostFunction ( ):
Input:
Time per epoch t_t; Attributes $n_{sec,h,k}$, $n_{sec,k}$, $c_{s,k}$, $c_{h,k}$, for all tasks; Frequency of the accelerator f_h; Minimum soft-core frequency $f_{k,min}$; Maximum soft-core frequency $f_{k,max}$;
Output:
Cost of the objective function for the given set of inputs v;

```
1.  v ← 0;
2.  ∀k∈{0 to k_tot -1 soft-cores}{
3.    t_{h,k} ← TimeSpentOnAcc(t_t, n_{sec,h,k}, c_{h,k}, f_h);
4.    t_{s,k} ← t_t − t_{h,k};
5.    f_k ← SoftCoreFreq(n_{sec,k}, n_{sec,h,k}, c_{s,k}, t_{s,k});}
6.  v ← v + exp(|t_t − ∑ t_{h,k}|);
7.  ∀k∈{0 to k_tot -1 soft-cores}{
8.    v ← v + |n_{sec,k} − n_{sec,h,k}|;            //Increase cost if offset increases
9.    if(n_{sec,h,k} < 0){v ← v + |n_{sec,h,k}|;}   //Penalize if out of bounds
10.   if(f_k > 0){v ← v + f_k;}
11.   if(f_k < f_{k,min} AND t_{s,k} > 0){v ← v + (f_{k,min} − f_k);}
12.   if(f_k > f_{k,max}){v ← v + (f_k − f_{k,max});}   //Penalize if out of bounds
13. }
```

Algorithm 8 Cost function used for Nelder-Mead heuristic for solving shared hardware allocation problem

A.7 Edge Detection for 16×16 MB

ExtractDominantIntraMode ( ):
Input:
16×16 MB X; MB edge size (16 pixels) s;
Output:
Dominant prediction mode for H.264/AVC m;

1. $rd_w, E_w \leftarrow$ BorderRunningDifference (X), $\forall\ w = a, b, c, d$
2. $Line_{out} \leftarrow$ GetDominantLine(rd_w, E_w)
3. **switch** $Line_{out}$ **do** *//Implicit 'break' after each case*
4. **case** $Line_0$
5. $m \leftarrow$ (DC, P, H, V) *//No edges found on any border*
6. **case** $Line_1$
7. **If** $E_a - E_b > s/2$ ➔ $m \leftarrow$ (DC, H, P, V)
8. **otherwise, if** $E_a > E_b$ ➔ $m \leftarrow$ (DC, P, H, V)
9. **otherwise, if** $E_b - E_a < s/2$ ➔ $m \leftarrow$ (DC, P, V, H)
10. **otherwise** $m \leftarrow$ (DC, V, P, H)
11. **case** $Line_2$
12. **If** $E_a - E_c > s/2$ ➔ $m \leftarrow$ (H, DC, P, V)
13. **otherwise, if** $E_a > E_c$ ➔ $m \leftarrow$ (DC, H, P, V)
14. **otherwise, if** $E_c - E_a < s/2$ ➔ $m \leftarrow$ (DC, P, H, V)
15. **otherwise** $m \leftarrow$ (DC, P, V, H)
16. **case** $Line_3$
17. **If** $E_b - E_d > s/2$ ➔ $m \leftarrow$ (DC, V, P, H)
18. **otherwise, if** $E_b > E_d$ ➔ $m \leftarrow$ (V, DC, P, H)
19. **otherwise, if** $E_d - E_b < s/4$ ➔ $m \leftarrow$ (DC, V, P, H)
20. **otherwise** $m \leftarrow$ (DC, P, H, V)
21. **case** $Line_4$
22. **If** $E_c - E_d > 3s/4$ ➔ $m \leftarrow$ (DC, V, P, H)
23. **otherwise, if** $E_c > E_d$ ➔ $m \leftarrow$ (DC, P, V, H)
24. **otherwise, if** $E_d - E_c < 3s/4$ ➔ $m \leftarrow$ (DC, P, V, H)
25. **otherwise** $m \leftarrow$ (DC, H, P, V)
26. **case** $Line_5$
27. $m \leftarrow$ (V, DC, P, H) *//Dominant Vertical mode*
28. **case** $Line_6$
29. $m \leftarrow$ (H, DC, P, V) *//Dominant Horizontal mode*
30.

Algorithm 9 Estimating most probable intra-prediction mode for H.264/AVC 16×16 MB

A.8 Motion Estimation with Hybrid Memory

HybridMemoryME( ):
Input:
Current video frame *F*; Reference frame search window *sw*; ME Engine memory access pattern ME_{patt};
Output:
NVM's power gate control register PG_{SW};

```
1.  ∀ i | i ∈ {CTU in F} {
2.    wait(IsReadBankReady);                    //Motion estimation is ready for CTU
3.    CU ← CTU_i;
4.    PerformME(CU);
5.  }
6.  Function PerformME(CU) {
7.    [β1 β2] ← EstimateNVMAddr(ME_patt, CU);
8.    PG ← TurnONMemSectors(β1, β2);            //Preemptively turn ON memory sectors
9.    TurnOFFExcessMemSectors(ME_patt, CU, PG);
10.   ∀ j | j ∈ {PU in CU} {
11.     ∀ n | n ∈ {Search number in ME_Patt} {
12.       [b_w, b_h] ← GetDimensions(PU_j);
13.       WaitForME( );                         //ME not ready yet, current PU must wait
14.       [x_ref, y_ref] ← GetReferenceAddr(n, ME_patt);
15.       if (β1 > x_ref or β2 < x_ref + b_w) {
16.         PG ← TurnONMemSectors(x_ref, x_ref + b_w); //Latency penalty
17.         UpdateEstimator(x_ref, CU); }
18.       DoMotionEstimation(n, ME_patt, x_ref, y_ref, PU); }
19.   }
20.   if (CU = SmallestCU) return;              //At 8×8 CU level, cannot divide further
21.   ∀ k | k ∈ {0,1,2,3} {                     //Split the CU in 4 equal sub-CUs
22.     CU_k.C_W ← CU.C_W / 2; CU_k.C_H ← CU.C_H / 2; CU_k ← CU;
23.     PerformME(CU_k); }                      //Recursively call ME again
24. }
```

Algorithm 10 Motion estimation process in HEVC video encoder using AMBER

Appendix B: ces265 HEVC Video Encoder

In context of high workload of video compression due to high resolution and FPS requirement, video encoding standards now allow for multithreaded (parallel) encoding possibilities, which can be exploited on multi- and many-core systems. In this book, we have envisioned a multithreaded video application and applied the concepts of compute and application configuration to this application. To demonstrate these concepts, one such application is the ces265 video encoder, an open-source HEVC software architecture for parallel video encoding. By simultaneously encoding video tiles on independent cores, time complexity and power savings are obtained, especially for large workload (like video encoding with high frame resolutions). The architecture of ces265 is flexible and allows for easy integration, runtime adaptation, and extensions.

B.1 Introduction and Motivation

New video standards, like high efficiency video coding (HEVC) [1] and VP9 [2], permit parallel encoding of video by breaking coding dependencies and create virtual boundaries around independent coding regions. Although current industry standards like H.264/AVC [3] and VP8 also allow for parallel encoding, their compression efficiency is not high compared to their successors (i.e., HEVC and VP9). On the other hand, the enhanced compression efficiency offered by HEVC and VP9 is accompanied by high complexity and application's workload. As a concrete example, HEVC takes about 40–70% more time as compared to H.264 [4] for processing the same video. Therefore, it is vital to parallelize the latest and next-generation video encoders if they must replace the current industry-standard video encoders.

In addition, recent trend of designing programmable processing chips is to have multiple low-frequency, low-power cores instead of a single, fast, and power-hungry core. This is because as already discussed, the dynamic power (p_{dyn}) of a core is related to its frequency (f) with the following relation:

$$p_{dyn} \propto c v_{dd}^2 f \tag{A.1}$$

In this equation, c is the capacitance and v_{dd} is the device voltage. Moreover, increasing the frequency of a core requires a linear increase in v_{dd}; therefore, the dynamic power is roughly proportional to the cube of the frequency. Further, for every chip, there is a power cap, which must not be violated, otherwise, we risk damaging the chip. Hence, a core's frequency cannot be arbitrarily increased or even, in some cases, independently assigned. This constraint is due to the power wall that prohibits increasing the frequency of the cores beyond a certain limit to sustain quality of service demand (i.e., frame rate). Hence, chip designers have

steadily moved towards having multiple compute cores, which has resulted in software designers to develop parallelization mechanisms, to exploit the potential of multiple cores.

In HEVC, tiles can be independently encoded (in parallel). However, as already pointed out, to satisfy the encoding frame rate demand, the software needs to distribute tiling workload by considering (a) the number of available cores and (b) the frequency of the cores. It is also possible that HEVC video encoding takes place on a heterogeneous multi- or many-core system. In such cases, some cores have lower resources or processing capacity (e.g., lower frequency, smaller caches, etc.) than the rest. Additionally, workload of video tiles may not be similar due to non-uniform tile structure or due to change in the video content at runtime. Thus, tiles may consume different amount of processing cycles, and the workload of cores is unbalanced, which may result in computational hotspots and decrease core utilization. The software architecture of HEVC under consideration, ces265 [5], addresses these challenges by providing adaptability and flexibility to HEVC video encoding, and provides integration and extension possibilities.

B.2 Technical Description of ces265

The software architecture of ces265 video encoder is outlined in Fig. B.1. There are numerous classes used in ces265. The Encoder's Top class is used to access YCbCr 4:2:0 planar raw video samples, construct tile structure for encoding video frames, and write encoder statistics. The main compression class is the Group of Pictures (GOP) Compressor class. A group of video frames is pushed to this class, and it outputs compressed bitstreams for each video frame to the Encoder's Top class. The GOP Compressor calls the methods of Slice Compressor class to compress video

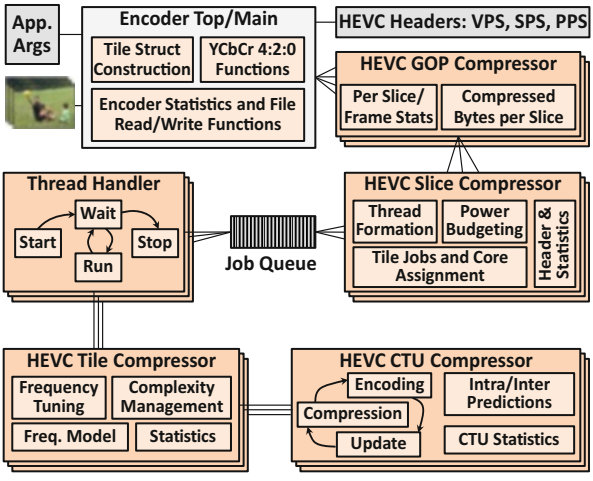


Fig. B.1 Software architecture of ces265 video encoder. The Top object can have multiple "GOP Compressor" objects. Each "GOP Compressor" object can instantiate multiple "Slice Compressor" objects. With each "Slice Compressor" object, a "Job Queue" object is attached, which can be accessed by multiple "Thread Handler" objects. Currently, ces265 only supports single GOP and Slice Compressor objects

frames. The Slice Compressor class calls methods of the Tile Compressor class, and each Tile Compressor class has an associated Coding Tree Unit (CTU) Compressor class. The CTU Compressor class is the class where the intra- and inter-encoding functions of HEVC are written and bitstream is generated.

With reference to the challenges mentioned above, some scientific and technical aspects of ces265 will now be briefly discussed.

Tile Formation in ces265 A user can assign a tile structure by three different methods:

- If the multi- or many-core system is homogeneous (i.e., all the computing cores have the same resources, e.g., frequency, cache size), the user can select a uniform tile structure. The user needs to provide ces265 with width and height on the frame in tiles.
- For the second option, the user can state the exact width and height of each tile in CTUs. This method is suitable for workload balancing on heterogeneous systems with dissimilar set of compute nodes or when it is required to reduce the number of compute cores [6].
- For the third option, the software determines the number and location of uniform tiles, depending upon the number of available/allotted cores and frequency of the cores [5]. These parameters (number of cores and frequency) must be provided to ces265 to utilize this option.

Multithreading in ces265 Multithreading is achieved by three entities: a processing thread, a job queue (or FIFO), and a manager to fill the job queue with encoding tasks. Manager fills the job queue from one side and the threads pull the jobs from the other side. To encode multiple tiles simultaneously in ces265, the Slice Compressor class generates a pool of parallel threads. In addition, it also creates a Job Queue object and attaches these threads to this queue. These threads (defined in Thread Handler class) scan the job queue wait for the arrival of jobs. The Slice Compressor class acts as a manager and creates tile compression jobs. A processing thread currently idle and waiting for a processing job pops a tile compression jobs from the queue. A video frame is deemed encoded when all the processing threads are idle and there are no more jobs left in the Job Queue.

Consider the case of uniform tiles encoded with a homogeneous many-core system. In this case, all threads will approximately take the same amount of time, and the number of threads created by ces265 will be equal to the number of tiles. In case of non-uniform tiling (with some tiles smaller than the others), the number of parallel threads created by ces265 can be lesser than the number of tiles. This is because a single thread can process multiple smaller tiles in the same time as a thread processing a larger tile. However, for both uniform and non-uniform tile processing, note that the software does not require any changes. Larger and smaller tile jobs enter the same workload queue and are treated by the threads all the same. An idle thread will start processing the tile as soon as there is a job available to process in the queue.

Note that the Job Queue and the Thread Handler classes are independent of ces265 encoding. These classes can be reused in ces265 for GOP or slice-based threading, or by other applications not limited to video, as they only need to know which function to call when processing and a structure containing input arguments to this function.

Power Management in ces265 As discussed in Sect. A.2.1, power consumption of a system is a major concern and will remain in the near future. The ces265 video encoder is designed with the pretext of application- or kernel-level power management techniques. If the platform allows the application to change the working frequency of the compute core, then the functionality of frequency tuning can be embedded into the software. By changing the frequency, an application can lessen the power consumption of the system by reducing the clock frequency or satisfy the quality of service demand (frame rate) by not running the core too slow. The ces265 allocates such frequency control functions in the Tile Compressor class, whereby just after the core starts processing the tile, the frequency of the core is modified to compress the corresponding tile.

B.3 Implementation and Uses of ces265

ces265 is written in C++, without using any architecture specific Intrinsics [7] or SIMD [8] instructions. The pthread APIs are used for multithreading. Using the same source, ces265's binary can be compiled on both Windows- and Unix-/Linux-based operating systems. The premises upon which foundation of ces265 is built are:

- Easily understandable and readable, for getting started with video encoding concepts and, specifically, HEVC
- Portable to multiple platforms, by using only standard C++ libraries and functions, and universally available development packages (e.g., pthreads)
- Easily extendible to include additional functionalities, such as parallelizing GOP and Slice compression
- By having a small memory footprint, portable to small systems with limited fast memories
- Introducing runtime tuning, by presenting the user multiple encoding options and configuration knobs
- Extensive analysis of encoder's workload and parameters, by dumping encoder's statistics at multiple levels of hierarchy (CTU, tile, slice, and GOP-level statistics)

The ces265 is open-source software and is available for download at http://ces.itec.kit.edu/ces265/ under GNU General Public license (http://www.gnu.org/licenses/). To test multithreading and power management (by frequency scaling) of ces265 on a many-core system, Sniper x86 many-core simulator [9] is used, as

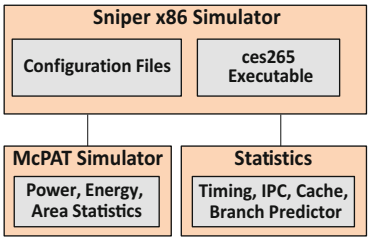


Fig. B.2 Sniper x86 many-core simulator setup to run ces265

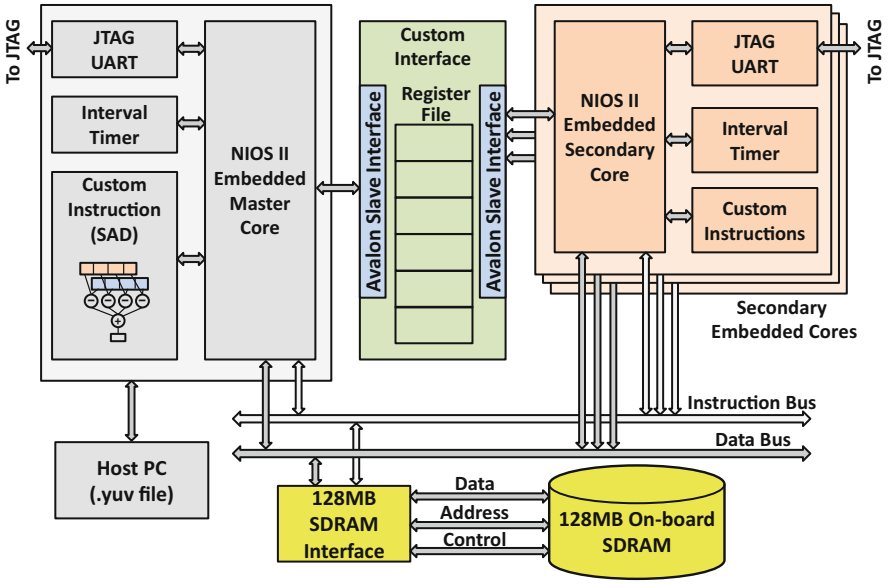


Fig. B.3 Multi-core architecture of ces265 on an Altera's FPGA

shown in Fig. B.2. Sniper simulator allows for application-level frequency control, which is exploited by ces265 while simulation.

B.4 Implementation of Multi-core ces265 on FPGA

To test video encoding and its associated challenges on embedded platforms, ces265 video encoder is also ported to an Altera's FPGA-based soft-core system as shown in Fig. B.3, which shows the multi-core hardware architecture. For this architecture, the soft-cores called Nios-II (provided by Altera) are used to process HEVC encoding. Each core processes a single tile. The so-called Master core also prepares the metadata (e.g., memory addresses) for other cores, called secondary cores.

The multi-core ces265 implementation on FPGA includes efficient arbitration and sharing of peripherals. In this system, all the cores share:

- SDRAM controller for storing their software and processed data
- A custom peripheral for efficient communication among the cores
- System ID for successful debugging of the software in multi-core environment

Sharing the peripherals is another design and performance challenge for this multi-core system. There is no guarantee that a core will be assigned a peripheral in a specific order. Similarly, sharing external memory via the SDRAM controller is also a challenge of a multi-core system. Since the memory contains the program and data for each core, it is important to use separate area for code execution of each core. It should be noted that cores sharing the same program memory space would result in erroneous execution as one core might overwrite other cores' memory. Therefore, each core has its own unique .text, .rodata, .rwdata, .heap, and .stack sections. Usually these five default linker sections are defined as:

- .text covers the executable code.
- .rodata contains the read-only data used for execution of the software.
- .rwdata is used for storage of read and write variables, along with pointers that have been used in the software.
- .heap is used for storing dynamic memory (using "malloc" in C or "new" in C++).
- .stack stores function call parameters and other temporary data.

Another addition to the multi-core systems is the implementation of Custom Instructions (CI) or in-core hardware accelerators. These accelerators can be called by user-defined macros that are designed and implemented in hardware. With the help of in-core accelerators, functions that might take a large number of clock cycles to execute can be replaced with CIs, which usually take lesser number of cycles. This will further decrease the total computation time of the ces265 encoder. In this implementation of ces265, the SAD function is implemented as a CI.

Further, note that this architecture is not limited to ces265. Any parallelizable application can be implemented on custom hardware like FPGAs, using the above methodology.

Working The Master core governs the processing flow. It has information about the addresses of the data frames in the external memory, the tile structure for secondary cores to properly work, and the details of the custom interface used for communication.

Depending upon the number of secondary cores, the master core constructs the exact dimension of the video tiles as well as the start and end pixel locations of these tiles. We can employ uniform or non-uniform tiling techniques for tiling. The master core also allocates memory for bitstream handlers and other associated data structures (which will be used by the secondary cores). Afterwards, it starts writing the tile addresses and memory pointers to the custom interface (a set of registers for each secondary core). Afterwards, the master core sets a "go" signal, which starts the tile

processing. In this architecture, the master core also processes tiles. Once it finishes processing its tile, it starts collecting status signals of all other secondary cores. The secondary cores fill the appropriate registers within the custom interface to notify the master core about their status. If all cores finish their workload, the master core initiates the processing of the next frame.

B.5. Future Directions

ces265 can be extended by including vector processing instructions (or SIMD instructions). Parallelizing the GOP and slice compression engines is a logical step towards hierarchical resource and power management of ces265. As pointed out earlier, this can be achieved with minimal effort, because the Job Queue and Thread Handlers can be seamlessly replicated and inserted before GOP Compression and Slice Compression classes. In addition, ces265 can be used for real-time 3D-HEVC (for efficient compression of 3D videos). Moreover, the frequency control functions (currently embedded into tile compressors) can be used in the CTU Compressor class for fine-grained frequency tuning of the processing core.

Appendix C: FPGA-Based H.264/AVC Prototype

To functionally verify the hardware accelerators for H.264/AVC and its multicasting solution, the H.264/AVC encoder is prototyped on an Intel® FPGA (formally known as Altera FPGA). This prototype includes:

- Parallel processing of up to four parallel full-HD (1920×1080) video frames at 25 fps.
- Capturing raw video samples from real-world analog camera inputs and converting them into YCbCr 4:2:0 planar format
- Writing and reading video frame samples to/from the external DDR3 memory of the FPGA development board
- Processing the video samples on the FPGA and compressing them to H.264/AVC standard-compliant bitstream
- Displaying the reconstructed video via DVI for quick debugging

Some of the design challenges for implementing the prototype consist of:

- Debugging the hardware board
- Incorporation of cameras on the daughter board with the FPGA board
- Converting analog camera video streams into digital YCbCr 4:2:0 planar format
- Appropriately tuning the timing parameters for interfacing the DDR3 memory with the FPGA
- Setting the clock frequencies of the DDR3 and the FPGA (video encoder)
- Integrating Gigabit Ethernet with the video encoder

- Setting up the DVI output for reconstructed video

The following provides brief details about the working principles of this prototype.

C.1. Simulation and Design Workflow

The simulation and design workflow of the H.264/AVC video encoder is shown in Fig. C.1. The H.264/AVC intra-encoder is developed using an in-house C-based H.264/AVC intra-encoder, ModelSim, and MATLAB co-simulation framework. Quartus logic device design software together with Synopsys Design Compiler is used for design implementation and analysis. H.264/AVC and other encoding configurations are fed to the C-based H.264/AVC encoder, which generates compressed video bitstreams and test vectors for use in VHDL simulation with ModelSim. These test vectors are also used to visualize the impact of different configuration settings on the hardware accelerator via MATLAB's tools. The VHDL files (for H.264/AVC entities and associated test benches), test vectors, and the Executable and Linkable (elf) file of the embedded Nios-II CPU [10] are used for RTL-level simulations and debugging via ModelSim. These VHDL files are also used in the Quartus software to generate the bitstream of the architecture to be burned on the FPGA. The Synopsys Design Compiler generates area and power logs for the H.264/AVC VHDL entities using a 65nm TSMC library.

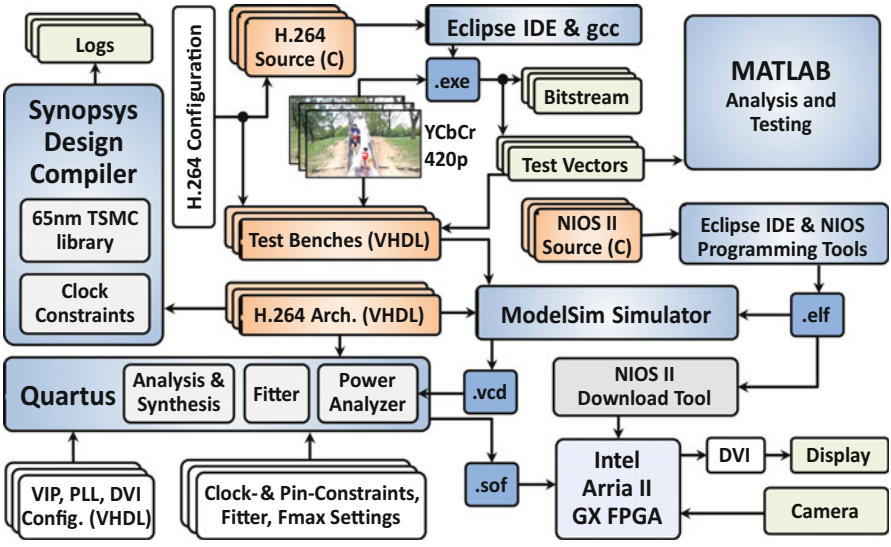


Fig. C.1 Design methodology and simulation setup of the H.264/AVC video encoder

C.2. FPGA Prototype

For functional verification, the H.264/AVC encoder and the multicast video gather and display are separately prototyped on a real-world FPGA. These architectures are implemented on Altera's DK DEV 2AGX260N FPGA Development Kit, with a cost-effective Arria II GX EP2AGX260 FPGA. The architecture for the H.264/AVC intra-encoder is shown in Fig. C.2. The development kit has a 64-bit-wide DDR2 dual inline memory and a 16-bit-wide single chip DDR3. For functional verification, the video data is streamed from a SD camera to the encoder, and the output bitstream (generated by CAVLC [11]) is passed as AVB packets [12] via Gigabit Ethernet interface. The whole encoder uses a single clock domain of 150 MHz whereas the DDR3 is clocked at 300 MHz. NIOS II embedded soft-core is used for frame-by-frame control and for future extensions of the prototype. Nios-II also configures video input, Ethernet, and DVI modules at startup. Note that the video output path is not required in the actual implementation, however, it is used to view the reconstructed video frames. Similarly, there is an optional mixer component only used to stack the frames together and generate background.

As a service to the research community, the VHDL of the proposed H.264/AVC encoder is open-sourced [13]. The reason being writing and testing custom hardware for a large and complex project (like H.264/AVC) is a highly time-consuming process. Therefore, with this release, the research community will have a head start in developing video compression architectures, and it will help to implement and test novel architectures in an efficient manner.

C.3. H.264/AVC Prototype Evaluation

On the FPGA prototype, the total cycles taken by the transform module is 56. The prediction (X') generation takes 17 cycles for every perdiction, including the P prediction mode. Compared to other architectures, the different attributes of this architecture are tabulated in Table C.1. Other architectures shown here do not consider multicasting. Therefore, we have written the resolution supported by a single encoder presented here. In contrast to the other techniques, this encoder can be adjusted to balance the workload and encoding methods. Moreover, the area footprint is reduced by: (a) realizing only one DCT folded butterfly [18], (b) reusing the same hardware for quantization and inverse quantization, and (a) reusing the same structure for SAD computations of all the four modes. Further, compared to [19], this architecture only uses one prediction generation unit (instead of two parallel units) at 150 MHz instead of 310 MHz. However, this architecture can also be extended in a similar fashion and is more flexible because the parameters d_s and θ are configurable, as discussed in Chap. 5. An algorithm implemented in Nios-II can regulate these variables, even at runtime. Area usage and maximum frequency of important modules of the encoding loop implementation are given in Table C.2. Note that M9K embedded SRAM block memories are used as FIFOs to

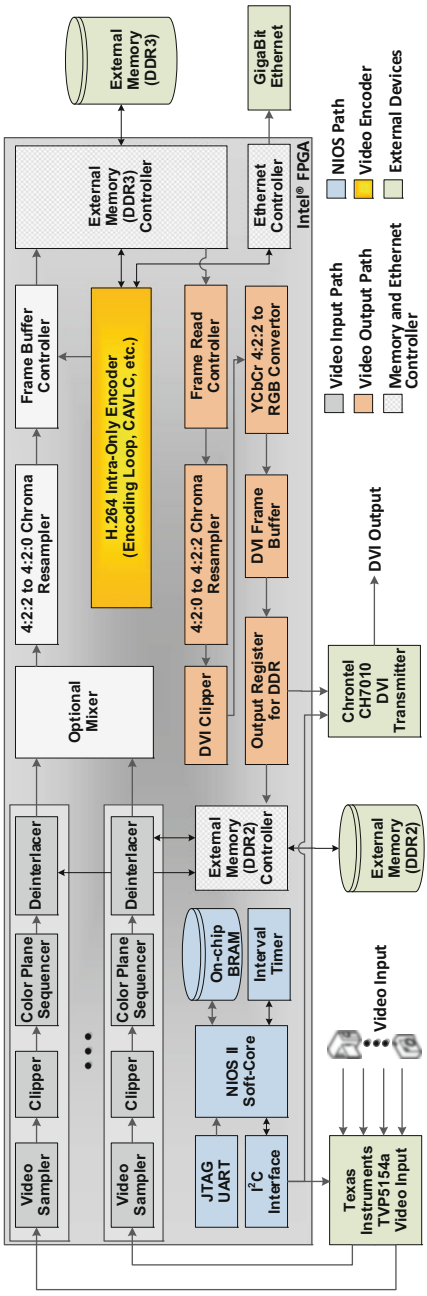


Fig. C.2 Architecture on Arria II GX FPGA with the H.264/AVC encoder, video input and output paths, and external devices. For complete system implementation and verification of H.264 Intra-only encoder, only one video-input is utilized

Table C.1 Comparison of H.264/AVC encoder peresented here with other architectures

	[14]	[15]	[16]	[17]	Presented
MHz	114	54	150	140	150
Size	1920×1080 4:2:0	720×480 4:2:0	1920×1080 4:2:0	1920×1080 4:2:0	4068×2288 4:2:0
FPS	30	31	61	30	25
Mode	I4MB, I16MB Parallel	I4MB, I16MB Parallel	I4MB, I16MB Parallel	I4MB, I16MB Serial	I16MB Serial
Design	130 nm TSMC	250 nm TSMC	180 nm TSMC	130 nm TSMC	40 nm Altera FPGA
Area	265.3K Gates	89K Gates	201.8K Gates	94.7K Gates	11K ALUS, 8K Regs, 562 Kbit BRAM
Entropy Coding	CABAC	CAVLC	NI	CAVLC	CAVLC
Multicast Support	No	No	No	No	Yes
Cam. input	No	No	No	No	Yes
Tf Folding	No	No	No	No	Yes
Tf Merging	Yes	Yes	Yes	Yes	Yes
Q Merging	Yes	No	No	No	Yes

Table C.2 Synthesis results for the encoding loop presented in this book. The total area (last row) exceeds the sum of the components, because of the further glue logic among components (e.g., FIFOs)

Module	MHz	ALUTs	Regs	Memory [Kbit]
4×4 RO+HT	321.34	438	1604	0
Merged Tf	167.87	7901	3958	5.34
Edge Detector	385.8	283	525	0
Mode Pred.	171.14	1426	747	256
Reconstruct	475.74	460	969	0
Total	–	10,583	8088	562

Table C.3 Synthesis with TSMC 65 nm at 310 MHz, Synopsys DC. Total gates denote the logic equivalent of a two-input NAND gate

Module	4×4 RO+HT	Luma Tf	Choma Tf	Edge Detector	Reconstruct	Scheduler (n_v=4)
Gates	5.83K	106	217.2	4.38	1.14	1.56
SRAM	0	5.34K	10.69K	0	0	0

Table C.4 Resolution supported in 16:9 aspect ratio by the H.264/AVC encoder presented here for 25 FPS, operating at 150MHz

	$\theta = 1$	$\theta = 2$	$\theta = 3$	$\theta = 4$
***ds*= 1**	4894×2752	4564×2568	4294×2416	4068×2288
***ds*= 2**	4894×2752	4564×2568	4416×2484	4280×2408

connect the encoding stages, and the total area of the encoding loop for this architecture also includes these FIFOs.

Tf transform, *NI* not implemented

Table C.3 shows the results for synthesizing the VHDL of the proposed encoder on a TSMC 65 nm technology [20], using a target frequency of 310 MHz. Synthesis is carried out using Synopsys Design Compiler [21], with medium effort mapping and automatic hold time violation correction. The total number of gates actually presents the two-input NAND gate equivalent of the 65 nm TSMC technology.

One can determine the total number of MBs n_{frm} of a video frame that can be processed by the encoder, for a given cycle budget per MB c in the loop clocked at f_k MHz with a target FPS of f_p. For this purpose, n_{frm} can be calculated as shown in Eq. (A.2). Using n_{frm}, the maximum image dimensions for given aspect ratio can be determined. In Table C.4, we show the maximum sustainable frame dimensions at 16:9 for different encoder configurations. Here, $f_k = 150$ MHz at 25 FPS. As seen, the encoder presented in this book can process the maximum dimensions of 4068×2288 while supporting all 16×16 modes at 25 FPS with no row down-sampling.

$$n_{frm} = \frac{f_k}{f_p \times c} \tag{A.2}$$

References

1. Sullivan, G. J., Ohm, J., Han, W., & Wiegand, T. (2012). Overview of the high efficiency video coding. *IEEE Transactions on Circuits and Systems for Video Technology, 22*(12), 1649–1668.
2. The WebM Project | VP9 Video Codec Summary. WebM, [Online]. Available: http://www.webmproject.org/vp9/. Accessed 03 Oct 2015.
3. Ostermann, J., Bormans, J., List, P., Marpe, D., Narroschke, M., Pereira, F., Stockhammer, T., & Wedi, T. (2004). Video coding with H.264/AVC: Tools, performance, and complexity. *IEEE Circuits and Systems Magazine, 4*(1), 7–28.
4. Pourazad, B. M. T., Doutre, C., Azimi, M., & Nasiopoulos, P., (2012). HEVC: The new gold standard for video compression: How does HEVC compare with H.264/AVC?. *IEEE Cosnumer Electornics Magazine,* pp. 36–46.
5. Khan, M. U. K., Shafique, M., & Henkel, J. (2014). Software architecture of high efficiency video coding for many-core systems with power-efficient workload balancing. In *Design, Automation and Test in Europe*.
6. Shafique, M., Khan, M. U. K., & Henkel, J. (2014). Power efficient and workload balanced tiling for parallelized high efficiency video coding. In *International Conference on Image Processing*.
7. Compiler Intrinsics. Microsoft. [Online]. Available: http://msdn.microsoft.com/en-us/library/26td21ds.aspx. Accessed 03 Oct 2015.
8. Siewart, S. (2009). Using Intel® streaming SIMD extension and Intel® integrated performance primitives to accelerate algorithms. White paper, Intel.
9. Kim, W., Gupta, M., Wei, G.-Y., & Brooks, D. (2008). System level analysis of fast, per-core DVFS using on-chip switching regulators. In *International Symposium on High Performance Computer Architecture (HPCA)*.
10. Altera. Nios II Process Reference Handbook. [Online]. Available: https://www.altera.com/literature/hb/nios2/n2cpu_nii5v1.pdf. Accessed 08 June 2015.
11. Advanced video coding for generic audiovisual services. ITU-T Rec. H.264 and ISO/IEC 14496–10:2005 (E) (MPEG-4 AVC), (2005).
12. IEEE 802.1 AV Bridging Task Group. IEEE, 20 March 2013. [Online]. Available: http://www.ieee802.org/1/pages/avbridges.html. Accessed 04 Oct 2015.
13. Khan, M. U. K. H.264 VHDL, Chair for Embedded Systems (CES), KIT, June 2015. [Online]. Available: http://ces.itec.kit.edu/h264hdl/. Accessed 04 Oct 2015.
14. Kuo, H. C., Wu, L. C., Huang, H. T., Hsu, S. T., & Lin, Y. L. (2011). A low-power high-performance H.264/AVC intra-frame encoder for 1080pHD video. *IEEE Transactions on Very Large Scale Integrated Systems (TVLSI), 19*(6), 925–938.
15. Huang, Y.-W., Hsieh, B.-Y., Chen, T.-C., & Chen, L.-G. (2005). Analysis, fast algorithm, and VLSI architec-ture design for H.264/AVC intra frame coder. *IEEE Transactions on Circuits and Systems for Video Technology, 15*(3), 378–401.
16. Diniz, C., Zatt, B., Thiele, C., Susin, A., Bampi, S., Sampaio, F., Palomino, D., & Agostini, L. (2011). A high throughput H.264/AVC intra-frame encod-ing loop architecture for HD1080p. In *International Symposium on Circuits and Systems*.
17. Lin, Y.-K., Ku, C.-W., Li, D.-W., & Chang, T.-S. (2009). A 140-MHz 94 K Gates HD1080p 30-Frames/s intra-only profile H.264 encoder. *IEEE Transactions on Circuits and Systems for Video Technology, 19*(3), 432–436.
18. Malvar, H., Hallapuro, A., Karczewicz, M., & Kerofsky, L. (2003). Low-complexity transform and quantization in H.264/AVC. *IEEE Transactions on Circuits and Systems for Video Technology, 13*(7), 598–603.
19. He, G., Zhou, D., Zhou, J., & Goto, S. (2010). Intra prediction architecture for H.264/AVC QFHD encoder. In *Picture Coding Symposium*.
20. Taiwan Semiconductor Manufacturing Company Limited. TSMC, [Online]. Available: http://www.tsmc.com/. Accessed 7 Oct 2015.
21. Design Compiler. Synopsys, [Online]. Available: http://www.synopsys.com/Tools/Implementation/RTLSynthesis/DesignCompiler/. Accessed 7 Oct 2015.

Index

A
Accelerator, 5, 12, 14, 16, 18, 19, 42, 45, 53, 54, 56, 67, 70, 71, 112, 131, 132, 135, 138, 153, 157, 159, 203, 204
  allocation, 132–141
  arbiter, 134
  clustered accelerator, 53
  custom instruction (CI), 69, 131
  decoupled accelerator, 54
  distributed, 153–157
  in-core accelerator, 53, 69
  loosely coupled accelerator, 54, 68, 131
  off-load, 12, 17, 19, 40, 133, 134, 136
  tightly coupled accelerator, 53
Address generation units (AUGs), 27, 43, 70, 130, 159, 164, 167, 172, 191
  Write AGU, 167, 172, 192
AMD, 4
  Opteron, 4, 69, 177
Android, 4
Application configuration, 91, 92, 97, 104–112, 177–182, 202, 207
  configuration matrix, 100, 105, 176, 182
Application-specific instruction-set processor (ASIP), 53
Application-specific integrated circuits (ASIC), 5, 6, 15, 53
Approximate computing, 205–206
Aspect ratio, 97
ATM machines, 2
Audio, 2, 170
Automotive, 43

B
Barrel shifter, 55, 147, 148, 204
Batch processing, 6
Battery, 6, 40
Bitrate, 7, 106, 154, 179, 181
Bitstream, 31, 38
Box plot, 86, 192

C
C#, 20, 192
C++, 20, 176
Camera, 6, 8, 19, 26, 129, 143, 159, 167, 172, 196
Capacitance, 41
Ces265, 20, 108, 176, 186
Cisco Visual Network Index, 3
Color space, 26
  chrominance, 27, 129, 130, 140
  luminance, 27, 84, 129, 130, 140, 149, 150
  RGB, 27, 128
  YCbCr, 27, 128, 129
Communication, 1, 2, 18, 37, 69, 90, 127, 128, 131, 132, 172
  bandwidth, 7, 38, 40, 102, 128, 130
  bridge controller, 132
  encoding loop, 152
  ethernet, 130, 144
  FIFO, 27, 129, 140, 146, 158, 164
  I/O, 17, 18, 67, 82, 127, 129, 139, 143, 144, 203
  internet, 2

M.U.K. Khan et al., *Energy Efficient Embedded Video Processing Systems*,
DOI 10.1007/978-3-319-61455-7

Communication (*cont.*)
multicast, 12, 19, 21, 38, 117, 128, 139–141, 144, 185, 203
packet detection, 130
parity, 39
TCP/UDP, 6
transmission energy, 2
turbo coding, 39
wireless, 6, 28
Compression, 2, 6–8
CODEC, 6
Daala, 7
distributed video coding (DVC), 8, 38–40
encoding loop, 139, 142–153, 204
hybrid DVC (HDVC), 8, 38–40
lossy compression, 31
predictive video coding, 7
Slepian-Wolf decoder, 39
Compute configuration, 91, 94–104, 116, 176, 201
Computer-aided design (CAD), 5
Computing, 2
Context switches, 71, 90, 139
Control, 49, 188
Cosmic, 175

D
3D, 8, 128, 207
Deinterlacer, 129
Denoising, 26, 47
Digital signal processor (DSP), 15
Discrete cosine transformation (DCT), 79, 143, 144, 146, 185, 204
AC coefficient, 143
butterfly, 146
DC coefficient, 143, 144, 148
Doping, 9
Downsample, 150

E
Education, 1
Electric field, 9
Encryption, 26
Entertainment, 1, 2, 38
gaming, 26
screen, 2
televisions, 2
YouTube, 4, 6, 10
Entropy, 31, 140
context-adaptive binary arithmetic coding (CABAC), 141, 143
context adaptive variable length coder (CAVLC), 140, 141, 146
entropy coder (EC), 140, 147
entropy coding (EC), 143, 145
Epoch, 102, 103, 106, 107, 118, 121, 122, 124, 134, 136, 180, 186. *See also* Group of pictures (GOP)
Ethernet, 19

F
Feedback, 16, 102, 106, 117
Field-programmable gate arrays (FPGA), 5, 15, 19, 70, 128, 147, 156, 203
Nios-II, 20, 53, 69, 131, 203
Floating-point, 170
Frame drop, 28
Frequency, 4, 12, 18, 31, 42, 47, 50, 68, 70, 74, 79, 91, 93, 97, 99, 100, 102, 104, 113, 116, 118, 120, 121, 123, 133, 135–137, 156
clock-domain crossing, 129, 140
cycle, 72, 79, 99, 102, 114, 115
self-regulate, 92, 100–103

G
Google
google file system, 10
VP8, 7, 36
VP9, 7, 36
Gradient, 47, 78, 79, 107
edge extraction, 151
Sobel edge filter, 107
Graphics processing unit (GPU), 5, 15, 26, 37, 46, 56, 67, 113, 206
Group of pictures (GOP), 36, 39, 93
GUI, 20, 192

H
H.264/AVC, 7, 9, 18, 25, 29, 32–36, 38, 46, 48, 50, 89, 139, 184, 203
macroblock (MB), 32, 89, 140–142, 144, 149
reorder (RO), 144, 204
Hadamard transform (HT), 143–146, 149, 204
HT lookahead, 144, 145, 149, 204
Hardware replication, 139
Heterogeneous, 14, 15, 18, 46, 49, 69, 76, 89, 112–116, 128, 131, 132, 184, 185, 202, 206

efficiency index, 113, 115, 184 (*see also* Homogeneous)
master node, 131
secondary node, 131
thermal design power (TDP), 10
Heuristic, 19, 103, 203
bin-packing heuristic, 97, 182, 184
High efficiency video coding (HEVC), 7, 9, 18, 25, 29, 32–36, 38, 46, 47, 73, 89, 156, 181, 202
coding tree unit (CTU), 32, 36, 48, 76, 89, 108, 123, 137, 155, 156, 158, 159, 162, 163, 182
coding unit (CU), 32, 76, 156, 162, 163
H.265, 7
prediction unit (PU), 32, 76, 105, 109–112, 155
PU map (PUM), 110, 155
PU map above (PUMA), 110, 155
rate-distortion (RD), 77, 153, 202
transform unit (TU), 76
wavefront parallel processing (WPP), 74
x265, 177
Histogram, 74, 82, 83, 108, 155, 171, 192, 194
Homogeneous, 45, 89. *See also* Heterogeneous

I
IBM, 4
Instruction-level parallelism (ILP), 44
Instruction set architecture (ISA), 56
Intel, 4, 108
Core-i7, 4
Intel Atom E600C, 5
Larrabee, 4
Polaris, 4
single-chip cloud computer, 4
Xeon Phi, 4
Interlace, 144, 148, 152
International Telecommunication Union (ITU), 7
Internet of things (IoT), 40
Interpolation, 26, 36, 181
Inter-prediction, 29, 34–36, 48. *See also* Intra-prediction
Intra-prediction, 33, 34, 105, 107, 155. *See also* Inter-prediction
closed loop, 153
open-loop, 153, 204
seed intra angular mode, 108
seed pixel, 108
ISO-Moving Picture Expert Group (ISO-MPEG), 7

J
Job queue, 18
Joint Collaborative Team on Video Coding (JCT-VC), 7, 32, 190

L
Latency, 18, 20, 47, 51, 52, 72, 82, 83, 140–153, 157, 158, 160, 163, 164, 204
Leakage, 9
Linear pulse code modulated (LPCM), 170
Linux, 69
Lookup table, 95

M
Many-core system, 6, 38, 67, 69, 72, 74, 92, 94, 118, 127, 133, 134, 203
soft-cores, 128, 131, 133, 138, 139, 203
Mapping, 16, 45, 92, 184
McPAT, 79, 137, 177
MediaBench, 175
Memory, 26, 42, 67, 188–197
adaptive energy management for on-chip hybrid video memories (AMBER), 157, 158, 160, 163, 175, 189–191
bit line, 43
cache, 28, 42, 51, 55, 102
controller, 72, 102, 130
DDR3, 128, 130, 139
DRAM, 6, 52, 82, 159
external memory, 6, 26, 29, 35, 42, 46, 53, 80, 83, 128–130, 137, 139, 157, 160
frame memory, 27
hybrid memory, 52, 53, 69, 82, 83, 157–164, 189–191, 204
magnetoresistive random-access memory (MRAM), 19, 52, 67, 72, 82, 189, 204
nonvolatile memory (NVM), 6, 13, 19, 52, 67, 70, 80, 82, 157, 189
on-chip memory, 6, 26, 29, 51, 80, 82
phase-change RAM (PRAM), 52
read master, 140, 159
register, 55, 72
ring buffer, 130, 167
scratchpad, 6, 15, 17, 134, 138
search window, 35, 36, 48, 81, 82, 159–161, 163, 190, 191, 204
sector, 159, 161, 162, 204
SRAM, 6, 10, 13, 43, 67, 71, 80, 82, 83, 85, 134, 138, 157, 159, 160, 163–173, 189, 191–198, 204

Memory (*cont.*)
triple-buffering, 130
word line, 43
write master, 130, 159
Memory read transducers (MRT), 20, 85, 166, 191
Memory wall, 44
Memory write transducer (MWT), 20, 85, 164, 166, 169, 173, 191, 205
inverter switch, 166, 168, 169, 173
MiBench, 175
Mobile, 6, 39. *See also* Communication
ModelSim, 156, 191
Moore's law, 41
Motion estimation (ME), 32, 51, 52, 79, 81, 82, 107, 158
EPZS, 35
level C+, 51
prefetching, 35, 190
read factor, 35, 128, 160
reference frame, 81, 128, 158–160
search window, 35, 48, 82, 159, 161, 163, 190, 204
super-resolution, 36, 47
TZ, 35, 81, 157
Multimedia, 4, 45, 201
Multiplexer (MUX), 141, 145–147, 166
Multiplier, 147, 148, 204
Multithreaded, 12, 14, 18, 20, 42, 49, 116–124, 175, 185, 202
OpenCL, 206
OpenMP, 70, 176
pthread, 176
Multiview video coding (MVC), 36, 207

N
Nanoscale, 1
Navigation, 1
Network on chips (NoC), 13
NMOS, 44
Nonlocal means filter (NLMF), 47, 75

O
Open-source, 48, 176, 192. *See also* Ces265
Operating system (OS), 70, 76, 119
Linux, 176, 186
Windows, 176
Optimization problem, 91, 113, 119, 136, 203
cost, 137
goal programming problem, 94, 116
Nelder-Mead technique, 136, 202
nonlinear, 136
penalty function, 137
Oracle, 4

P
Parallelization, 12, 13, 16, 17, 21, 36–38, 45–47, 70, 74, 89–94, 175–185, 201, 207
Parsec, 175
Peak signal-to-noise ratio (PSNR), 29, 36, 154, 202
Bjøntegaard delta bitrate (BD-Rate), 29, 98, 108, 111, 179, 182
Bjøntegaard delta peak signal-to-noise ratio (BD-PSNR), 29, 98, 108, 111, 179, 182
PMOS, 43, 83
Portability, 14, 17, 72, 176
Power, 71, 114–116, 118, 120, 121
dynamic frequency scaling (DFS), 42
dynamic power, 41, 52, 70, 74, 81, 133, 160, 166, 169, 190
dynamic power management (DPM), 46, 49, 56
dynamic voltage frequency scaling (DVFS), 42, 48, 69, 70, 94, 118, 182
leakage power, 157, 158, 160, 161, 163, 190
static power, 41
thermal design power (TDP), 42, 71, 116, 119, 122, 124, 187, 203
Power budgeting, 12, 14, 18, 46, 49, 57, 89, 116, 118–123, 187
Power wall, 1, 9, 10, 41, 73, 116, 132, 186
clock gating, 155, 156, 166
coolant, 41
dark silicon, 10, 15, 16, 21, 41, 48–50, 122, 132, 187, 201 (*see also* Reliability)
Dennard's scaling, 9
dynamic thermal management (DTM), 48, 50
power density, 9
power gating, 68, 155, 157, 161, 163
TDP, 14
task migration, 56
temperature, 10, 41, 48, 56, 133, 203 (*see also* Reliability)
thermal design power (TDP), 12, 15, 16, 42

Q
Quality of service (QoS), 2, 5, 7. *See also* Service level agreement
Quantization, 31, 80, 103, 120, 143, 147, 185
  quantization parameter (QP), 105, 147, 154, 156, 179

R
Rate-distortion (RD), 32, 47, 77
Real-time, 6, 9, 27, 139
Reconstructed frame, 31, 82, 143
Region of interest (ROI), 76
Reliability, 1
  aging, 10, 12, 14, 17, 20, 21, 42–44, 54, 55, 83–86, 164–173, 191–197, 204
    controller, 164, 168, 195
    map, 171
  bias temperature instability (BTI), 10
  bit-flips, 10
  bit rotation, 85, 172, 196
  data adaptation rate, 168, 169, 172, 195
  delay margin, 43
  duty cycle, 43, 55, 68, 83, 166, 170, 171, 173, 192, 194, 198
  field-programmable gate arrays (FPGA), 6
  hot carrier injection (HCI), 44, 197
  negative-bias temperature instability (NBTI), 10, 12, 20, 42–44, 167, 197
  nibble swap, 85, 172, 192
  static noise margin (SNM), 10, 83, 171, 192
  stressmap, 84, 192
  triple modular redundancy (TMR), 207
Resample, 128, 129
Rescheduler, 140, 141, 203. *See also* Scheduler
Residue, 31, 38, 143, 144, 150, 151
Resolution, 7
  4K UHD, 55, 80, 141
  8K UHD, 32
  1080p, 51
  CIF, 51
  clip, clipper, 128, 129
  full-HD, 1, 128, 138, 142, 164
  QCIF, 7, 51
Resource budgeting, 45, 71, 117–120, 185–188, 202. *See also* Power budgeting
  cluster, 117–122, 124, 186
  misprediction, 120
  offset, 120, 122
Robotics, 1
  navigation, 2
  robots, 2, 39
Running difference, 108, 151

S
Scalable video coding (SCV), 207
Scheduler, 140, 141, 203. *See also* Rescheduler
Scheduling, 37, 53, 54, 132–141, 147, 148
  raster scan, 140, 142
  round-robin, 69, 139, 140, 167, 203
Security, 1, 2, 38
Self-organizing map (SOM), 20, 162, 204
Service level agreement (SLA), 2, 3
Shift register, 27, 130, 166
Slice, 36, 47. *See also* Tile
Sniper simulator, 79, 137, 177
Spatial correlation, 5
Spatial neighborhood, 28, 33. *See also* Spatial correlation
Streaming application, 6, 39, 123
Sum of absolute differences (SAD), 31, 35, 149, 150, 152, 206
Switching activity, 41, 197
Synchronization, 90, 131
Synopsys Design Compiler, 191
System identification, 72, 101
  estimation model error, 101
  model coefficients, 92, 101, 103, 177, 181
  recursive least squares (RLS), 93, 97, 102, 105, 177, 181, 202
  regression, 72, 102, 114, 123, 182

T
Temporal correlation, 5, 170
Temporal neighborhood, 28, 34. *See also* Temporal correlation
Texture, 77, 83, 107, 112, 137, 150, 156
Threshold, 106, 110, 150, 151, 163, 169, 188
Throughput-per-watt, 3, 5, 11, 14, 15, 21, 28, 40, 46, 56, 69, 89, 107, 115, 184, 185, 201
  FPS-per-watt, 179, 180
Tile, 37, 74, 94, 132, 179, 202. *See also* Slice
  collocated tile, 37, 75
  master tile, 97
  non-uniform tiling, 96–99, 182–184, 202
  retiling, 104, 181
  secondary tile, 97
  uniform tiling, 94–96, 118, 177–182, 202

Tilera, 4
Time delay, 41
Time to market, 5
Toggling, 129, 169, 197
Transform, 31

V
Variance, 79, 98, 106, 109, 110, 137, 155, 156
  Chan's formula, 111
  Knuth's formula, 106 (*see also* Texture)
VHDL, 191
Video input pipeline (VIP), 27, 129, 139, 203
Video quality, 28, 32, 40, 51, 74, 89, 90, 95, 104, 106, 108, 179
VLIW, 15
Voltage, 41

W
Web browser, 4
  Chrome, 4
WebRTC, 2
Workload balancing, 15, 16, 20, 37, 38, 45, 46, 68, 90–94, 112–116, 131, 175–185, 201
  configuration, 94, 100, 106
  distribution, 90, 112–114, 116, 131, 202, 206
  variation, 45, 71, 75, 76, 91, 102, 104, 177