QUALITY-AWARE PERFORMANCE ANALYSIS FOR MULTIMEDIA MPSoC PLATFORMS

DEEPAK GANGADHARAN

(B.Tech, University of Kerala, India)

A THESIS SUBMITTED FOR

THE DEGREE OF

DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

NATIONAL UNIVERSITY OF SINGAPORE

2012

Declaration

I hereby declare that this thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis. This thesis has also not been submitted for any degree in any university previously.

Deepak Gangadharan

16/11/2012

Acknowledgments

The PhD years have shaped my thoughts about life and therefore I am glad that I took the decision to pursue graduate studies. Professionally, the PhD journey has been one of the most challenging and rewarding journeys of my life. Hence, there are several people I would like to thank for helping me in this journey.

I would firstly like to thank my first supervisor Prof.Samarjit Chakraborty for introducing me to the interesting area of System level Performance Analysis. Although he left NUS 1.5 years into my PhD program, he constantly supported me by giving timely advice on my research directions. I also thank him for hosting me at TU Munich where some very important works of this thesis were developed. Secondly, I would like to thank Prof.Roger Zimmermann for agreeing to supervise me when Prof.Samarjit left. They also were generous enough to give me complete freedom in etching out the research direction.

I am grateful to my PhD thesis committee members Prof.Tulika Mitra, Prof.Wong Weng Fai and Prof.Nalini Venkatasubramaniam for providing their valuable inputs to improve the thesis. I thank the School of Computing at NUS for supporting me throughout the program. This journey would not have been possible but for the collaboration with some wonderful colleagues. I therefore thank Linh, Haiyang and Balaji for helping me in the publications that we jointly published.

I would equate the journey of PhD to a roller coaster ride with its ups and downs. The support from friends and family members cannot be overlooked during such times. I was fortunate enough to have a good set of friends in Vintu, Suresh, Senthil, Vinitha and

Vijith whenever I needed to relax my mind. Similarly I had some good friends at NUS (Ankit, Unmesh, Ramkumar, Swaroop, Balaji, Kathy, Vamsi, Malai, Ransi and Mahesh) with whom I have spent enjoyable moments.

I finally dedicate this thesis to my parents (Mr. G.Gangadharan and Mrs.Sreedevi Gangadharan) and my sister (Ramya) for having supported me when I decided to take a plunge into graduate studies. I am indebted to my parents for allowing me to follow my own career path though it meant that I would stay away from them for a long period of time.

Contents

List of Figures

List of Tables

Abstract

State-of-the-art embedded devices (e.g. mobile devices) run multiple applications on multiprocessor system-on-chip (MPSoC) platforms. MPSoC platforms are becoming popular due to the increasing number and complexity of target applications. Among the target applications that the embedded devices run, video players are extensively used by the end user and contribute to a large fraction of the workload. They are used to play both stored and live videos which are decoded on the MPSoC platform. Decoders are resource intensive applications requiring large buffer sizes, processor bandwidth and thermal management techniques to adhere to thermal constraints. These are the primary factors that determine the cost of the target embedded device. In order to analyze these crucial system resources early in the design cycle, various system level performance analysis techniques are employed. Although we focus on video decoding in this thesis, the techniques developed are general and can be applied to all applications that employ frame-based processing (e.g. games that are made up of graphics frames).

Although there is a large body of work that discusses system level performance analysis techniques for multimedia applications mapped to a MPSoC platform in various design contexts, most of these were not quality loss-aware techniques (quality losses have earlier been considered only in the case of power management). These techniques compute the platform resource requirements that enable maximum output video quality. However, multimedia applications can tolerate some data loss without significant deterioration in the output video quality. This property has not been considered in performance analysis techniques before, i.e., quality loss-aware performance analysis techniques have not been studied before. In our work, we present simulation-based and analytical performance analysis techniques to determine the system resources in a quality-aware manner. The quality-resource trade-off has been shown to be important in saving vital resources for insignificant loss in quality. These works are briefly described below.

1. In the first work, we study the impact of video frame drops in buffer-constrained MPSoC platforms. In this work, we propose a formal framework to evaluate the buffer size vs. video quality trade-offs, which in turn will help a system designer to perform quality driven buffer sizing. In particular, we mathematically characterize the maximum numbers of frame drops for various buffer sizes and evaluate how they affect the worst-case PSNR value of the decoded video.
   However, the limitation in the formal framework does not allow a priority scheme to drop frames. Therefore, we study the impact of a novel prioritized frame dropping

scheme in buffer-constrained MPSoC platforms. The frame dropping scheme is crucial here to drop frames appropriately such that the required buffer size is reduced and target quality requirement is satisfied. Towards this, we propose a simple prioritized frame dropping mechanism which reduces the required buffer space more than existing frame dropping policies.

2. A Picture-in-Picture (PiP) like application where two videos are played simultaneously, is efficiently handled in televisions and personal computers by providing maximum quality of service to the multiple streams . However, it is a difficult task in devices with resource constraints. Therefore, we propose a network calculus based formal framework to help schedule multiple video streams in a PiP application in the presence of buffer contraints. We obtain considerable reductions in the processor cycle requirement for multimedia processing by trading with quality.

3. In order to satisfy thermal constraints while running power hungry applications like video players, dynamic thermal management (DTM) techniques are employed. Most of the earlier work in reducing peak temperature for multimedia applications relied on dynamic voltage and frequency scaling (DVFS) and dynamic power management (DPM) methods while taking care that maximum video quality is achieved. However, no prior work has exploited frame drops to lower the temperature under fixed quality constraints. Given the quality constraint, we propose a DPM framework that utilizes frame drops to dynamically insert low idle times in order to adhere to given peak temperature constraint.

In addition to the quality-aware performance analysis techniques mentioned earlier, we also have done some work in the direction of model-based fast performance analysis for multimedia MPSoC platforms. Here, we present techniques to reduce the simulation time for simulation-based performance analysis techniques for multimedia MPSoC platforms by using application workload models and performance models.

In this thesis, we add another dimension to the design stage of system level performance analysis by using the application quality loss information to perform quality loss-aware resource dimensioning. We develop quality-aware analytical and simulation based performance analysis techniques in order to dimension the critical resources.

List of Publications

Related to Thesis

1. Published

   - Deepak Gangadharan, Samarjit Chakraborty and Roger Zimmermann, "'Quality-Aware Media Scheduling on MPSoC Platforms"', Accepted in Design Automation and Test in Europe (DATE), 2013.
   - Deepak Gangadharan, Ma Haiyang, Samarjit Chakraborty and Roger Zimmermann, "'Video Quality-Driven Buffer Dimensioning in MPSoC Platforms via Prioritized Frame Drops"', 29th IEEE International Conference on Computer Design (ICCD), October 2011.
   - Deepak Gangadharan, Linh T. X. Phan, Samarjit Chakraborty, Roger Zimmermann and Insup Lee, "'Video Quality Driven Buffer Sizing via Frame Drops"', 17th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), August 2011.
   - Deepak Gangadharan, Samarjit Chakraborty and Roger Zimmermann, "'Fast Hybrid Simulation for Accurate Decoded Video Quality Assessment on MPSoC Platforms with Resource Constraints"', 16th Asia and South Pacific Design Automation Conference (ASP-DAC), January 2011.
   - Deepak Gangadharan, Samarjit Chakraborty and Roger Zimmermann, "'Fast

Model-Based Test Case Classification for Performance Analysis of Multimedia MPSoC Platforms"', International Conference on Hardware-Software Codesign and System Synthesis (CODES+ISSS), October 2009.

2. In Preparation

   - Deepak Gangadharan, Samarjit Chakraborty and Roger Zimmermann, "'Quality-aware Multimedia Processing on Thermally-Constrained MPSoC Platforms"', 2012 (Journal).

Chapter 3 is based on published article 1. Chapter 2 is based on the published articles 2 and 3. Chapter 5 is based on published articles 4 and 5.

The article in preparation constitutes Chapter 4.

Other Publications (Not part of the thesis)

- Haiyang Ma, Deepak Gangadharan, Nalini Venkatasubramanian and Roger Zimmermann. Energy-aware complexity adaptation for mobile video calls. *ACM Multimedia*, November 2011.
- Balaji Raman, Guillaume Quintin, Wei Tsang Ooi, Deepak Gangadharan, Jerome Milan and Samarjit Chakraborty. On Buffering with Stochastic Guarantees in Resource-Constrained Media Players. *International Conference on Hardware-Software Codesign and System Synthesis (CODES+ISSS)*, October 2011.

Chapter 1

Introduction

System-level performance analysis of MPSoC platforms is becoming an increasingly non trivial task with increase in complexity of these platforms. The increasing complexity is due to the large and varied set of applications mapped onto the MPSoC platforms. In order to support these applications, these platforms need to provide adequate resources, which are diverse in nature. The host of non functional dependencies introduced by processor and bus scheduling also need to be considered in performance analysis [4]. The non functional dependencies arise due to the nature of interactions among the various components in the architecture. These dependencies often are the main reasons for the contradicting performance demands of the target MPSoC platform. Here, the performance analysis task has to predict the important system parameters namely end-to-end delays and buffer requirements in the initial stage of the design cycle.

As portable embedded systems are increasingly incorporating MPSoC platforms, a sound system-level performance analysis is very important in the design cycle of these embedded systems. The existence of orthogonal product demands are the very reason for the requirement of a robust performance analysis process. Although the portable devices need to be designed with adequate resources to support many applications, the main goal is to reduce the overall cost of the system. The choice of hardware resource configurations and thermal considerations are the primary factors that affect cost of such a system. In order to reduce cost, if we cut down on these resources or do not provide sophisticated cooling solutions, the performance of the system is adversely affected. On the other hand, higher performance targets also results in increased cost of the system. Therefore, in order

to reduce cost, it is sometimes necessary to design the system such that the performance degrades gracefully, i.e., the deterioration in performance of the system is not perceptible.

Multimedia applications are a suitable choice to explore the tradeoff between resource requirements (and hence cost) and performance (we look at objective quality here). Therefore, this thesis deals with performance analysis for multimedia MPSoC platforms, which is briefly discussed in the next section. Although we present performance analysis for multimedia MPSoC platforms (specifically running video decoders employed in video players) without considering the presence of other applications, similar techniques can be extended to analyze the performance of multimedia applications in the presence of other non multimedia applications. In the next section, we discuss the multimedia MPSoC platforms, in particular, the variability of the tasks and the workload experienced by them and how it affects the design.

1.1 Multimedia MPSoC Platforms

In portable embedded systems, the MPSoC platforms primarily process multimedia content in video players and other similar applications. Such applications require considerable amount of computing resources (multiple processors interconnected in various topologies) and on-chip buffer resources. Video conferencing is another important application that is envisaged to be used extensively on mobile phones. Here, video encoding task needs to be executed on the MPSoC platform, which is a more resource intensive task in comparison to video decoding. Moreover, with the continuing evolution of video encoding/decoding standards, programmable platforms are playing an important role in readily incorporating additions in functionality. On the other hand, dedicated hardware platforms require unacceptably long design times for the same.

Viper SoC architecture [5] and Eclipse architecture template from Philips [6] are examples of MPSoC platforms that provide generic and programmable frameworks to process the wide variety of multimedia applications. They have been conceptualized to enable the system designers to rapidly design media processing devices like set-top boxes, high definition television etc. The complexity of designing these platforms arises from the large variation in the workload experienced by them for different input video clips. There is a considerable difference between the average to worst-case

workloads experienced here. Therefore, if the platform is designed for the worst-case scenario, the determination of resource requirements results in overestimates for majority of other multimedia inputs (e.g., video clips), which makes the design of multimedia MPSoC platforms a non trivial task. In the case of portable devices with MPSoC platforms running multimedia applications, it is very essential to take the large variation in input workload into consideration in order to derive appropriate system resources enabling low cost.

Before getting into the performance analysis techniques for specific system parameters, we first present a broad classification of the existing methodologies in system level performance analysis of MPSoC platforms. Here, we address the pros and cons of various MPSoC performance analysis techniques.

1.2 Classification of MPSoC Performance Analysis Techniques

There has been a large body of work dealing with system level performance analysis methodologies for MPSoC platforms in order to derive the critical system resources. The various methodologies that exist in literature are:

1. Simulation based methods.
2. Formal methods.
3. Semi-formal methods.

Simulation-based system-level performance analysis is a more widely adopted methodology for multimedia MPSoC platforms, mainly SystemC based full system simulation or trace-based simulation ([7], [8]). In the context of a video processing application such as an MPEG-2 decoder, these simulations take a library of test video clips as input. When simulated with this library, the MPSoC platform is considered to be appropriately designed if it behaves in accordance to all the performance constraints. It is analogous to the common software functional testing methodology [9]. However, unlike in the software testing scenario, simulation of MPEG-2 decoder application with the library of video clips is very expensive with respect to time. As mentioned in an earlier work [10], it may take tens of hours for the simulation of only a few minutes of video in a decoding application.

Therefore, the performance analysis time for such architectures steeply increases with the input library size. Further, manual identification of uncorrelated test inputs so as to expose the MPSoC architecture to all possible corner cases is a tedious exercise.

Hence, researchers resorted to a more systematic methodology for MPSoC performance analysis. Here, they have studied formal techniques ([11], [12]), in which various system components are modeled mathematically and worst case bounds of performance characteristics are found according to the model. This methodology eliminates the need for time consuming simulations altogether, but it has its own overheads in representing an entire system using a mathematical model. Moreover, formal analysis methods for multimedia MPSoC architectures do not generally take the inherent correlations among the workloads. It is also highly likely that some specifications of the MPSoC system are missed out in the models developed using this approach. Most importantly, the worst case bounds obtained for performance characteristics are very pessimistic. This does not lead to a very resource efficient MPSoC architecture.

There are some performance analysis methods in the literature which use a combination of both simulation and analytical methods. These come under the semi-formal methods. These methods try to use the good aspects of the two methods described above. Certain system components are simulated (especially which are hard to model) and the rest are analyzed using analytical models (to reduce the simulation time). However, this adds the burden of employing interfaces among two components being analyzed using different approaches. Less pessimistic results are also obtained using such methods when compared to complete formal performance analysis methods [13].

1.2.1 Simulation-based Performance Analysis

This method mainly involves performing extensive SystemC based full system simulation or trace-based simulation ([7], [8]) in order to estimate the performance metrics. A major difficulty in conventional simulation-based approach is the difficulty in generating an exhaustive set of test inputs that exposes the MPSoC architecture to all possible corner cases. This is made more non-trivial with the complex interactions among the various system components that occur under the influence of specific test inputs.

Wild et al. [14] propose an approach where the system resource functionalities are captured as

sequence of trace primitives. During simulation runtime, these are merged with the system architecture as transactions. SystemC is used as the modelling language.

Gao et al. [15] present a framework for hybrid simulation which shows a significant speed up when compared to conventional detailed simulation. It also provides more accurate performance estimation results for components like simple RISCs (Reduced Instruction Set Computers) to DSPs (Digital Signal Processors) and VLIW (Very Large Instruction Word) machines. The Processing Elements (PEs) are considered to be one of the above mentioned components and thus can be modelled. They claim a speed improvement of $3\times$ to $5\times$ for a multiprocessor simulation with low errors in performance estimates.

1.2.2 Formal Methods for MPSoCs

As discussed in Section 1.2, formal methods are used to find the best and worst case values of the performance parameters. The formal approach based system performance analysis domain works along two problem domains [11] namely task performance analysis in the form of process execution time analysis and resource sharing analysis, also known as schedulability analysis. However, we do not go into its details as it is outside the scope of this report.

In contrast to simulation-based approach, which considers each event individually, the formal analysis methods abstract each event to event streams and use some simple characteristics of these event streams to obtain the worst and best case performance parameter bounds [11]. However, this does not help in the global performance analysis of the system due to the complex nature of event streams. Hence, a mathematical framework called real-time calculus (RTC) ([16], [17]) was proposed in order to generalize the event model with upper- and lower-bound arrival curves. A technique called timed automata was used to model real time events with any level of detail but it leads to prohibitively large number of states [12].

Most of the work in formal methods for performance analysis of MPSoC architectures have not considered the workload correlations that exist. This gives very pessimistic results. Hence some work ([18], [19]) has been performed to develop a model to characterize and capture the existing workload correlations. These have been developed in conjunction with RTC, but give more tighter bounds on performance results (like processing delay of some event by a task mapped to a processor)

than given by RTC. They use workload correlation curves (WCC) which are formulated using the RTC framework in order to characterize the workload correlations. The detailed definitions can be found in [18].

Similarly Jersak et al. [4] have proved that, in the context of MPEG-2 video stream processing, using *system contexts* can improve the bounds obtained by performance analysis. This involves correlations between successive computation or communication. They also describe intra event stream and inter event stream contexts which can individually lead to tighter analysis bounds, although both these system contexts affect different parameters. Finally it has also been shown that a combination of these two system contexts can improve the performance analysis bounds further.

A modular performance analysis (MPA) method has been used ([20], [21]) to evaluate an in-car radio navigation system. The main idea of MPA is to provide a performance model that abstracts the functionality of a system with RTC into a performance model. As more information of the system (about the available computation and communication resources and other details) is available, it gives a more tighter bound on the performance parameters when compared to the RTC only based performance analysis.

1.2.3 Model-based Performance Analysis

Application specific models like scenarios have been lately used for an efficient performance analysis of the target platform. These approaches may use the good aspects of both the performance analysis approaches discussed earlier. Gheorghita et al. [22] propose the usage of *application scenarios* so as to speed up the design implementation and obtain more accurate estimates of the resource requirements. In contrast to use case scenarios, which provide the functional and timing behaviours, the application scenarios capture the internal details of the application in terms of the resource requirements necessary to meet the constraints. They further discuss the detection and classification of these application scenarios depending on the resources. Going forward, they also touch upon how the application level information can be used for scenario exploitation. This gives us an idea that it can be adapted into the multimedia MPSoC platform performance verification where the data dependent metrics are used to classify the video clips.

Raghavan et al. [23] discuss a model-based performance estimation in the context of a mobile de-

vice. They use modular and reusable component job models derived from simulation of hardware system models. The performance characteristics are analyzed by simulating the platform for various use cases. Those use cases that cause more demand of system resources are considered to be performance critical. An important aspect of this model-based performance estimation is that they lie in between the less accurate analytical models and detailed simulation-based approaches. Only few use cases are executed on a system level simulator while multiple parallel use cases are analyzed on a use case simulator (which takes in a use case model and generates performance metrics in lesser time). In this model, the resource usage function could be a table with inputs and corresponding outputs, a regression model or a single program giving an output for each input. The model-based performance estimation is also quite relevant in the hardware domain where parameters like interconnect power consumption are modelled.

In this thesis, we specifically look at low cost resource dimensioning for multimedia MPSoC platforms. In order to design low cost multimedia MPSoC platforms, certain application features of the multimedia data are exploited. The resulting resource dimensioning frameworks are developed using RTC tools. Moreover, the RTC performance analysis framework has been adapted to facilitate the design of low cost multimedia MPSoC platforms. Further, on conducting an extensive literature review on the state-of-the-art performance analysis methods, we realized that the problems experienced in the methods described earlier can be solved to a large extent by taking the approach of *model-based performance analysis*. To the best of our knowledge, very little work has been done in this area, especially for multimedia processing on MPSoC platforms. Hence, it is envisaged that efficient analytical models of the resources on an MPSoC platform can be derived based on the application test data. The test inputs can then be categorized into various well defined clusters based on the similarities that they exhibit within the framework of the resource models developed. Once the test inputs are clustered, representative inputs can be chosen from each cluster in order to perform system simulation. This also gives tighter bounds on the performance parameters along with reduction in simulation times (as the number of test inputs have now been reduced). Hence, this requires the need for a classification method of the multimedia streams which in turn need various resource models based on the complexity of the MPSoC architecture.

Before the contributions of this thesis are mentioned, it is essential to understand the state-of-the-art

in resource dimensioning methodologies, which will help emphasize the contributions discussed later. Therefore, we present the existing work on estimation of three vital resources of a MPSoC platform namely - buffer, processor cycles and thermal capacity (in terms of peak temperature).

1.3 Resource Dimensioning

Resource dimensioning for multimedia applications has been widely researched in the domain of multimedia over networks. Here the multimedia data is streamed from the server to the client over the network. This is implemented using various architectures ([24], [25], [26]) involving the server and the client. One of the key client parameters that many researchers have studied is the playout buffer or the jitter buffer size ([27], [28], [29]). The playout buffer size is interlinked with the minimum playout delay and the corresponding loss in quality [27]. Therefore, a trade-off has been explored between playout delay and buffer size ([29], [30]). However, given a buffer size and due to the variable nature of the incoming multimedia stream, adaptive playout techniques ([31]) have been studied in order to maintain an acceptable level of quality. Playout buffer sizing is all the more important in the wireless scenario where mobile devices exist with acute resource constraints [27].

Reduction of buffer size by buffer sharing ([32]) has been studied for streaming applications where multimedia data from different sources need to be streamed in a synchronous manner. In this context, the multiple buffers used for the multiple incoming streams are shared in order to reduce the overall buffer size. As in multimedia over networks, buffer sizing is a critical task for MPSoC platforms running multimedia applications. Here, there have been numerous efforts to minimize buffer with contradicting target objectives such as maximum throughput ([33]). Other efforts in buffer sizing for multimedia MPSoCs with an objective to maximize quality is discussed in Section 2.1. Although, there have been multiple efforts in buffer sizing, there are not many works that handle this problem by trading buffer size with a quantified quality loss (This is discussed in detail in the next section).

Processor time in terms of the number of cycles is another vital resource that is integral to the desired functioning of the multimedia MPSoC platform especially due to the intensive computations required for certain multimedia tasks. Processor scheduling algorithm is therefore an important

decision to efficiently handle multiple tasks. These algorithms are designed with various design objectives in mind. In [34], scheduling algorithms are discussed to minimize the buffer requirements for multimedia applications. The authors propose a static priority based scheduling algorithm which is shown to demonstrate smaller buffer requirements than the other existing scheduling algorithms. Jason et. al. [35] discuss an integrated scheduling framework to handle both real-time and conventional applications including multimedia with adequate fairness. Hence, in overloaded scenario the real-time tasks are also degraded gracefully.

Pawan et. al. [36] propose a hierarchical scheduler such that CPU bandwidth is allocated to the various application classes which in turn is partitioned among the sub classes. Wanghong et. al. [37] present a scheduler that accomodates the objective of energy efficiency while scheduling multimedia tasks on mobile devices by integrating dynamic voltage scaling along with soft real-time scheduling policy. There is rarely any scheduling algorithm that tries to allocate processor resources such that quality degradations are bounded and measurable. In this thesis, we do not present a scheduling policy, but derive mathematical bounds for the processor cycle requirements to process multimedia streams in a quality-aware manner.

Lately energy efficiency and thermal issues have become important design aspects in embedded systems. It is all the more important for mobile devices with limited energy budgets and low cost cooling solutions. As multimedia applications are one of the dominating loads in such devices, it becomes imperative to design mobile devices to efficiently process these applications in an energy/thermal-aware manner. In [38], the authors present a frame data computation aware dynamic voltage scaling (DVS) technique in order to decode both stored and real-time video clips with minimum deadline misses. Another work on DVS for MPEG decoding [39] tries to optimize DVS using two techniques : (1) minimizing delay and drop rate, and (2) using predicted decoding times. Yeo et. al. [40] propose a hybrid dynamic thermal management (DTM) scheme to increase the quality while reducing the peak temperature considerably in comparison to the existing methods. Here, the authors model the application thermal characteristics as a probability distribution of cycle requirements for decoding each frame. Many such techniques exist in literature that use DVS or DTM techniques to reduce energy or peak temperature, but most methods do not exploit quantified data losses to design energy/thermal aware systems as multimedia streams are tolerant to restricted

frame losses.

In the next section, we present a quality-aware approach to resource dimensioning, where we use an objective quality metric to drop data in order to obtain resource savings. Although there are existing works in literature that look at trading off system parameters with application quality by performing cross layer adaptations (at application, middleware, OS, network and hardware level) ([41]), this thesis delves into the mathematical frameworks to analyze trade-offs in the specific context of a multimedia MPSoC platform.

1.4 Resource Dimensioning: A Quality-Aware Approach

In MPEG-2/MPEG-4 video streams, there are typically three types of frames, namely, *I frame* (Intra coded), *P frame* (Predicted) and *B frame* (Bidirectionally predicted). I frames are intra coded frames and are not dependent on other frames in the video stream for decoding. Decoding a P frame requires the previous I or P frame as the reference frame. Finally, decoding a B frame requires two reference frames, namely, a forward reference frame (I/P frame) and a backward reference frame (I/P frame). It is clear from this organization of frames that B frame drops result in lesser amount of quality degradation in comparison to the I and P frame drops. In this thesis, we use this property to trade-off quality in a bounded manner with the various resources like buffer size, processor cycles and thermal capacity required. Although multimedia literature ([42], [43]) advocate the permissible number of frame drops within a window of displayed frames that result in tolerable loss, quantitative quality measures are not given. Therefore, we use an objective quality measure to instantaneously quantify the quality obtained in our frameworks.

Traditionally, video quality has been measured using both objective and subjective metrics. The

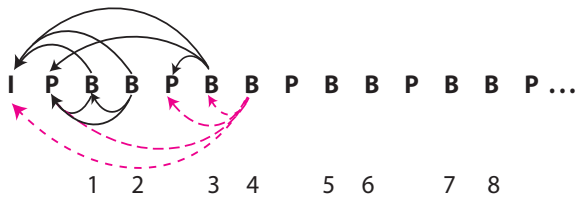


Figure 1.1: GOP decoding order with possible replacements for B frames if dropped.

subjective metrics like mean opinion score (MOS) are suitable to adequately capture the quality in accordance to the viewer perception [44]. However, it is not possible to get an instantaneous measurement of video quality using subjective metrics because it requires human subjects to view the video content and rate them based on certain factors. Moreover these measurements have to be conducted based on certain evaluation conditions ([45]) as given by [46] and [47]. On the other hand, traditional objective quality metrics like mean squared error (MSE) and peak signal to noise ratio (PSNR) are instantaneously obtained, but they are not a very accurate estimate of the user video perception. There are other more accurate objective quality evaluation metrics, but due to the simplicity in obtaining MSE and PSNR, we use them in our mathematical frameworks for resource dimensioning. Further, as the videos entering the target system do not have any reference to evaluate the quality, we use a no reference method whereby the quality deterioration is measured by substituting the dropped frame slots with concealment frames. We now discuss how the objective quality metrics are computed.

The maximum deviation among the dropped frames and the possible concealment frames (shown in Fig. 1.1) are computed in terms of MSE given by

$$MSE_{avg} = \frac{(MSE\_r + MSE\_g + MSE\_b)}{(3 \times W \times H)} \tag{1.1}$$

where $MSE\_r/g/b = \sum_{n=0}^{N_{drop}-1} (MSE\_r/g/b)_n$. $MSE\_r/g/b)_n$ is the deviation for red/green/blue pixels due to a dropped frame. The MSE for red pixel is given by

$$(MSE\_r)_n = \sum_{w=0}^{W-1} \sum_{h=0}^{H-1} (r_d(h,w,n) - r_c(h,w,n))^2 \tag{1.2}$$

where r_d is the red pixel intensity of the dropped frame and r_c is the red pixel intensity of the concealment frame (immediately preceding frame that was successfully processed). h, w and n are the height, width and frame drop number indices. Similar explanations hold true for MSE_g and MSE_b. W and H are the horizontal and vertical resolution of each frame in the video. N_{drop} is the number of frames dropped in the sequence. Finally, the PSNR value of a video sequence with frame drops is expressed as

$$psnr = 10 \times log_{10} \frac{(255 \times 255 \times N_{tot})}{(MSE_{avg})} \tag{1.3}$$

where N_{tot} is the total number of frames in the video sequence.

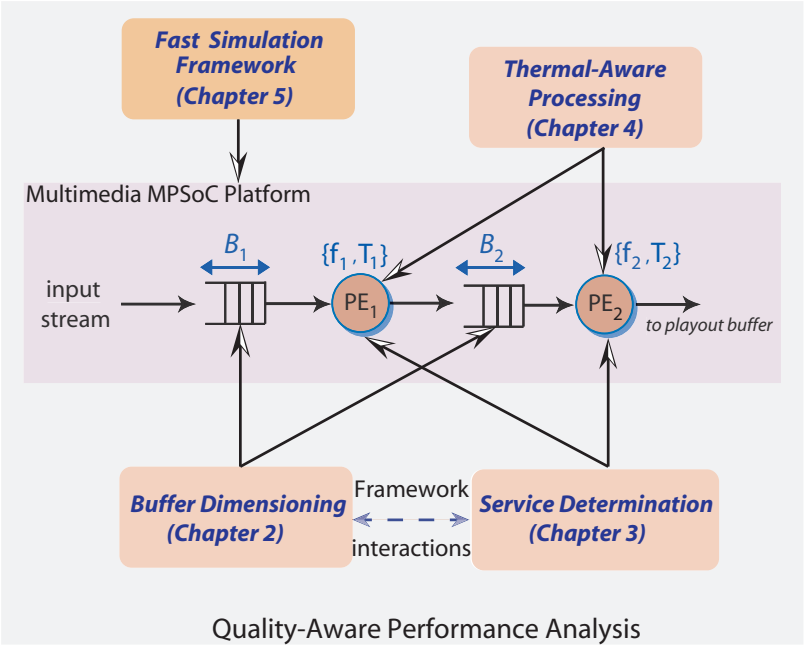


Figure 1.2: Quality-Aware Performance Analysis Framework.

1.5 Thesis Contributions

How do the individual frameworks glue together under a global system level performance analysis perspective: This thesis introduces novel analytical and simulation frameworks to do quality-aware performance analysis in order to determine the resource requirements in a quality-driven manner. To the best of our knowledge, this is the first work that uses an objective quality metric as part of the performance analysis frameworks to dimension resources while allowing some quantified quality loss. All individual performance analysis frameworks proposed in this thesis form building blocks of an integrated larger performance analysis framework as shown in Fig. 1.2. Although the different performance analysis frameworks for specific resource dimensioning discussed in this thesis consider the other resources to be constant, it is envisaged that a global performance analysis framework can be built where the proposed blocks (now considering only a single resource for performance analysis - discussed in Chapters 2, 3 and 4) in this thesis interact (shown by dashed blue line at the bottom of Fig. 1.2) to give an optimized set of resources for quality objective function or some multi-objective function including video quality as one objective. Although we show

the interaction between buffer dimensioning and service determination frameworks only in Fig. 1.2, similar interactions could also exist between either of the two frameworks with thermal-aware processing framework. The individual performance analysis techniques shown in Fig. 1.2 are also helped by the fast simulation techniques proposed in this thesis. These simulation techniques are used to either rapidly find the representative test clips, which would further speed up the analytical or simulation based performance analysis techniques to analyze the required system resources or to rapidly obtain the trace data that will be used by the proposed performance analysis techniques. The detailed contributions represented by the blocks are discussed in corresponding chapters.

1.5.1 Quality-Driven Buffer Dimensioning (Chapter 2)

In the first work, we study the influence of buffer sizing on worst case quality deterioration using a formal framework. There are two interlinked parts constituting our framework. For a given video clip, we perform the following operations.

1. Firstly, we derive the maximum number of frame drops (in any frame interval) for any given buffer size using a Network Calculus ([48]) based mathematical framework.

2. Secondly, we propose a novel method to compute worst case quality values for video clips. This is further used in conjunction with the maximum number of frame drops derived in the first part to find the worst case quality values for various buffer sizes.

A system designer does buffer sizing for an extensive library (covering all possible scenarios) of video clips, whereby sufficient buffer size is chosen so that a quality constraint is satisfied by all the clips in the library. Our framework can be used in this context. The information obtained from buffer size vs. quality trade-off curves for each clip can be used to determine the optimal buffer size for the entire library. In Section 2.2.1, we give an overview of our analytical framework.
In the second work on buffer dimensioning, we use a novel motion vector based frame dropping mechanism to decrease the required buffer size for a prespecified quality constraint. This motion vector based frame dropping is also compared with other existing frame dropping policies to show its effectiveness. Subsequently, a fast iterative strategy is proposed to derive the reduced buffer size for a target quality.

1.5.2 Quality-Driven Service Determination (Chapter 3)

In this chapter, we propose a formal framework to derive the processor cycle requirements for an incoming video stream in the presence of buffer constraints such that the video display quality satisfies the required target quality constraint. This framework will be very helpful to design schedulers for PiP (Picture in Picture) applications as they involve multiple incoming streams simultaneously that share processors in the platform. Therefore, a system designer would be able to use the framework to infer whether the multiple streams can be scheduled. Experiments were conducted using multiple video streams and it was verified that the processor cycle requirements derived using the framework actually satisfied the target quality constraints of the individual video streams.

1.5.3 Quality and Thermal-Aware Multimedia Processing (Chapter 4)

This is the first framework that combines an application level technique (namely frame drops) with dynamic thermal management (DTM) policy to process multimedia streams (video frames in this context) satisfying both quality as well as thermal constraints. It is a combined offline and online method where some stream information generated offline is used to optimize the idle time introduction online. The framework consists of two stages.

1. The first stage generates the frame drop pattern that satisfies a prespecified quality constraint. The quality constraint used in our work is the worst-case PSNR for a given interval of frames. This is an offline process and the frame drop pattern generated here is passed onto the next process which is online. The drop pattern is generated for each clip.

2. Once the quality driven frame drop pattern is derived, it is used to compute the idle times required such that the peak temperature never exceeds the threshold value. The additional idle times obtained due to frame drops reduces the idle times introduced. We prove this both theoretically and experimentally. Moreover, we also use a history based approach to optimize the idle times introduced. This is an online process.

We are able to get significant reductions in idle times and end-to-end delay for a small reduction in quality using our approach. For a 2 dB reduction in quality, we were able to reduce the PE_1 delay by approximately 2.5173 sec for a benchmark video with a $T_{max} = 80^{\circ}C$ setting.

1.5.4 Fast Simulation Frameworks for Multimedia MPSoC platforms (Chapter 5)

In our first work, we present a fast model-based test case classification methodology in order to classify video clips in a library to a fixed number of representative sets. A single video clip from each representative set can then be used to run system level simulations. This considerably reduces the number of simulations. However, in our work, we attempt to eliminate the simulation time for the representative clips also by using workload models for the multimedia tasks. The three major contributions of our first work are

1. A fast estimation of various Variability Characterization Curves (VCCs) of the video clips due to the use of bitstream analysis (avoids full decoding) for workload estimation.
2. A fine grained approach in choosing the VCCs (for classification) relevant to each stage in the architecture.
3. A new model for IDCT workload.

In the second work, we introduce a hybrid simulation based performance analysis framework to study resource trade-offs in the presence of data losses (or frame drops in our case). We use accurate workload models for some tasks and simulate the other tasks thereby reducing the simulation time required. Moreover, we are able to compute accurate quality losses (if frame drops are present) for various resource combinations.

1.6 Mathematical Background

In this Section, we briefly introduce the mathematical background, which forms the basis of performance analysis techniques presented in this thesis. We use the Network Calculus based RTC framework to analyze the performance of multimedia MPSoC platforms. The RTC framework properly captures the incoming multimedia data bursts and service provided for the incoming data to analyze the performance of multimedia MPSoC platforms. RTC defines certain interval based quantities called arrival curves and service curves in order to capture the variability in the incoming data and service. We now define these quantities based on a system model as shown in Fig. 1.3.

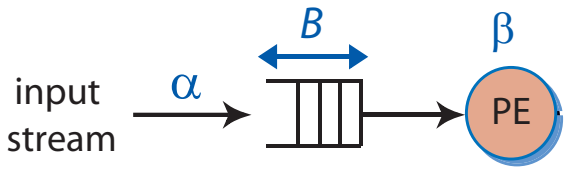


Figure 1.3: System Model for a processing component

Definition 1 ***(Arrival Curve).*** *For a video clip, let $a(t)$ denote the number of frames that arrive in time interval $[0,t)$. Then, the video clip is said to be bounded by the arrival curve $\alpha = [\alpha^u, \alpha^l]$ iff for all arrival patterns $a(t)$:*

$$\alpha^l(\Delta) \leq a(t+\Delta) - a(t) \leq \alpha^u(\Delta) \tag{1.4}$$

for all $\Delta \geq 0$. In other words, $\alpha^u(\Delta)$ and $\alpha^l(\Delta)$ give the maximum and minimum number of frames that can arrive over any interval of length Δ across the length of the video clip.

Definition 2 ***(Service Curve).*** *Let $c(t)$ denote the number of frames processed by a task mapped onto a processor in time interval $[0,t)$. Then, the service curve $\beta = [\beta^u, \beta^l]$ is a service curve of the processor iff for all service patterns $c(t)$:*

$$\beta^l(\Delta) \leq c(t+\Delta) - c(t) \leq \beta^u(\Delta) \tag{1.5}$$

for all $\Delta \geq 0$. In other words, $\beta^u(\Delta)$ and $\beta^l(\Delta)$ denote the upper and lower bounds on the number of frames processed over any interval of time Δ across the length of the clip.

Although RTC defines the above quantities in intervals of time, we have used frame intervals in order to perform some of the analysis in this thesis. Therefore we define frame interval as

Definition 3 ***(Frame Interval).*** *For a given video clip, a frame interval F is defined as a window of any F consecutive frames.*

This thesis also uses some elementary operations from Network Calculus. These operations are introduced further. For two functions f and g belonging to the set of monotonic functions

The (min,+) convolution $\otimes$ and deconvolution $\oslash$ operators are defined as:

$$\big(f \otimes g\big)(t) = \inf\big\{f(s)+g(t-s) \mid 0 \le s \le t\big\},$$
$$\big(f \oslash g\big)(t) = \sup\big\{f(t+u)-g(u) \mid u \ge 0\big\}.$$

Similarly, the (max,+) convolution $\overline{\otimes}$ and deconvolution $\overline{\oslash}$ operators are defined as:

$$\big(f \overline{\otimes} g\big)(t) = \sup\big\{f(s)+g(t-s) \mid 0 \le s \le t\big\},$$
$$\big(f \overline{\oslash} g\big)(t) = \inf\big\{f(t+u)-g(u) \mid u \ge 0\big\}.$$

1.7 Summary

First, we discussed the state-of-the-art performance analysis techniques for MPSoC platforms. The thesis was then motivated highlighting the aspect that application quality loss-aware performance analysis adds another dimension to the current performance analysis techniques. We then presented the overall framework of the thesis briefly describing the various proposed performance analysis techniques that take the application quality loss into consideration. The main contributions of the thesis were also mentioned in this chapter.

Overall Structure of the Thesis: Two quality-driven buffer dimensioning methods will be discussed in detail in Chapter 2. Then, we will present a quality-driven service determination technique for multiple multimedia streams on MPSoC platforms in Chapter 3. In Chapter 4, a thermal and quality-aware method for multimedia processing is developed in order to reduce the idle times inserted to satisfy the peak temperature constraints. All the previously mentioned performance analysis techniques are further helped by the use of fast simulation techniques, which will be described in detail in Chapter 5. Finally, we present our conclusions and discuss the possible future works in Chapter 6.

Chapter 2

Quality-Driven Buffer Dimensioning

Video decoders require significant amount of on-chip buffer resources in order to store the incoming/partially processed frames. A large on-chip buffer size increases the cost of the device running the video decoder. This is because large on-chip buffers are one of the major reasons for increase in the chip area ([49], [50]) and the power consumed ([51], [52]). Lowering power consumption is becoming increasingly important, especially in mobile devices, where extended battery life is one of the main design targets. Therefore, accurate buffer dimensioning in multimedia MPSoC platforms has attracted lot of research attention. All prior works in buffer sizing ([53], [54]) discounted the idea of frame losses in favor of maximum output quality. There have also been works on frame dropping policies ([2], [3]) to maximize output quality in the presence of scarce buffer resources. However, there has been no work on quality driven buffer dimensioning using efficient frame dropping strategies such that the required buffer size is reduced while satisfying a target output quality. This work can be appropriately used for multimedia decoders running on MPSoC platforms as these decoders can tolerate some quality loss without significant deterioration in video perception.

Contributions: In this chapter, two quality-driven buffer dimensioning methods are presented for multimedia MPSoC platforms. The first one is an analytical framework to derive the worst-case quality vs buffer size trade-offs via frame drops. Here, the oldest frame is dropped whenever the buffer is full. It is a non-trivial task to develop analytical frameworks to analyze the quality vs buffer size trade-offs using prioritized frame drops. Therefore, the second method discussed is a simulation based strategy for quality-driven buffer dimensioning using a prioritized frame dropping

strategy

2.1 Related Work

On-chip buffers take up a lot of chip silicon area. This is evident from [49], in which experiments clearly show the enormous amounts of silicon area increase due to the increase in FIFO size in the router. In [50], this same concern is demonstrated in the context of on-chip network design for multimedia applications. However, the authors do not drop any incoming packet from the buffer thereby giving importance to maximum application quality. A buffer sizing algorithm has been discussed in the context of networks on chip [55], where the authors are concerned about the reduction of buffers in network interfaces. There are various objective functions that are considered while choosing the appropriate buffer size. A buffer allocation strategy is proposed in [49] in order to increase the overall performance in the context of a networks-on-chip router design. In [56], an appropriate buffer size is chosen that gives the best power/performance figure.

Buffer dimensioning is an important aspect of designing media players. In the past, there has been lot of work in this area where several design factors have been taken into consideration while choosing the appropriate buffer size. Most of this work concentrated on studying the playout buffer vs. quality of service (QoS) tradeoffs. In [57], the authors discussed an optimal allocation of playout buffer size such that the playout delay is minimized for a given probability of underflow or a given QoS. Similarly, in [58], the buffer vs. QoS tradeoff is studied for multimedia streaming in a wireless scenario using a dynamic programming framework. A combined optimal transmission bandwidth and optimal buffer capacity is considered to support video-on-demand services [59]. Here, playout buffer overflow and underflow are not tolerated. There are also some other prior works which have not tolerated any loss as a result of buffer overflow and underflow ([60], [53], [61], [54]). However, none of these works have considered the tradeoff between buffer and video quality by allowing some buffer overflows (i.e., with constrained buffer). Here, video quality is not the end-to-end QoS, but the distortion in the received frames.

There are various frame dropping strategies that have been discussed in literature that try to maximize the video quality ([2], [3]). Invariably, all these strategies use a prioritization scheme to drop

the frames in a quality aware manner such that the quality deterioration is minimized. In [2], frame size is used to prioritize the frames before dropping. In this approach, frames with larger size are dropped later and frames with smaller size are dropped first. A distortion matrix is introduced in [3] to compute the priority of frame dropping based on the distortion that frame suffers if lost. As we drop only the B frames here, we consider the drop oldest policy during a buffer overflow. Similar schemes like *Drop Newest*, *Drop Random* and *Drop All* are also discussed in [62].

2.2 A Mathematical Framework for Video Quality Driven Buffer Sizing via Frame Drops

In this work, we propose a formal framework to explore the buffer size vs. video quality trade-offs, which can help a system designer to perform quality driven buffer sizing. Although these trade-offs can be explored using system simulations, simulation-based techniques are time consuming. The concepts discussed here, however, can be applied in the context of network- on-chip architectures where buffer size can be traded off against some quality parameter by dropping the less important data. In general, it is applicable to all such scenarios where losing some low priority data helps in saving buffer resources while still maintaining a good content quality. Therefore, it is important to recognize the least important data in the target application. As our framework bounds the quality degradation, the video quality does not deteriorate too much. In MPEG-2/MPEG-4 decoder applications mapped onto MPSoC platforms, B frame drops can be used to trade-off quality for buffer size. This selective dropping of frames requires a special scheme to differentiate among frames.

In our approach, a simple dual buffer management scheme is used in order to drop only the less significant frames (B frames). This scheme is shown in Fig. 2.1. The incoming multimedia stream is split into two distinct streams: the first consists of the less significant frames (B frames) and the second consists of the more significant ones (I/P frames). These two streams are fed to two distinct buffers. This partitioning will be explained in detail in Section 2.2.2. The processing element (PE) needs to be given a side information conveying the order in which the frames are to be processed (shown as the dotted line from the splitter to the PE in Fig. 2.1). In the setup shown in Fig. 2.1, drops occur only for B frames and the size of the associated buffer can be traded off with video quality.

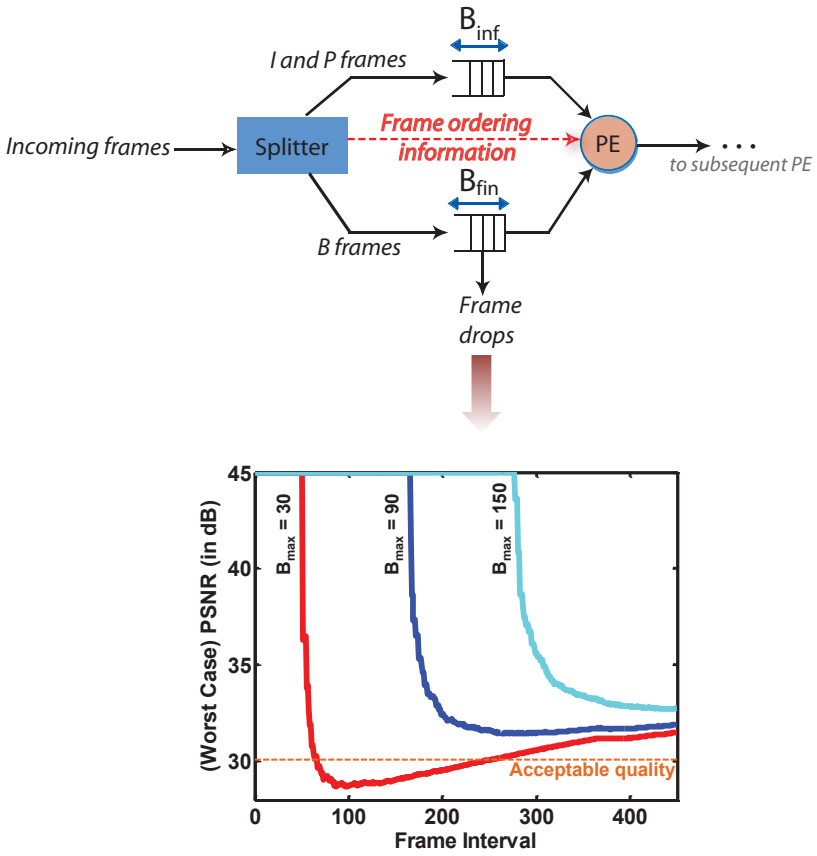


Figure 2.1: Dual buffer management scheme with drops in less significant frames and buffer size vs. video quality trade-off results for a benchmark MPEG-2 video *susi*_080 ([1]).

This trade-off (shown in Fig. 2.1) is obtained using a well known video benchmark *susi*_080 ([1]). In multimedia literature ([63]), 30 dB is considered to be an acceptable output video quality (shown as the horizontal line in the trade-off graph in Fig. 2.1). From Fig. 2.1, it can be observed that we give quality variations for three different buffer sizes over frame intervals.

The worst case quality value for a frame interval F is the minimum quality obtained over any F consecutive frames across the clip. From Fig. 2.1, it can be observed that if a maximum buffer size (B_{max}) of 30 frames is chosen, then the quality values (in dB) fall below the threshold value of 30 dB for certain frame intervals from 80 to 260. If the target quality constraint is to satisfy the 30 dB value for all frame intervals, then $B_{max} = 30$ frames will not be sufficient. However, if the target quality constraint is that the threshold value of 30 dB should be satisfied for any frame interval greater than 300, then $B_{max} = 30$ frames will be a good choice as the buffer size. We denote buffer

sizes in number of frames further because video frames consist of variable number of bits. However, we give an estimate of the minimum buffer savings in megabits (Mbs).

2.2.1 Buffer Sizing Framework

This section presents an overview of our mathematical framework to study the influence of frame drops on the PSNR of the decoded video under buffer constraints. We use the arrival curves and service curves from the Network Calculus to model the data streams and the service given by the resources, respectively, as they can model any arbitrary stream arrival pattern and any arbitrary resource service pattern. In addition, they can easily capture the data size variability and the processing variability exhibited in the multimedia setting we consider here. Before describing our framework, we introduce the underlying MPSoC platform.

Platform Description: In this work, we find the buffer size vs. worst case quality trade-off for a video clip on a buffer constrained MPSoC architecture as shown in Fig. 2.2. The terms explained in the problem definition are marked appropriately alongside the architecture. The architecture consists of two PEs, PE_1 and PE_2, each with its own offered service curves shown above them. Each PE is mapped with a set of tasks from the target decoder application. The PEs also each have a buffer in front of them, shown as B_1 and B_2, with maximum capacity of B_{1max} and B_{2max} (quantified in number of frames), respectively. As the buffer sizes are not always adequate, frame drops may occur, which are characterized as $\alpha^u_{drop1}(\Delta)$ and $\alpha^u_{drop2}(\Delta)$. $\alpha^u_{drop1}(\Delta)$ and $\alpha^u_{drop2}(\Delta)$ give the upper bounds on the number of frames dropped in any time interval of length Δ, where $\Delta \geq 0$. Although only a single buffer is shown in front of each PE, each buffer internally has two parts - one part where some of the least significant contents (B frames) are dropped and the second part where adequate buffer size is provided and the significant contents (I/P frames) are not dropped. The frame drops occur in the droppable buffer section and its drop bounds are derived by our framework. Before getting into the details of our framework, we first define some terminology.

Problem Definition: *Given the arrival curve* $[\alpha^u, \alpha^l]$ *of the video clip that is to be decoded on a decoder application mapped onto a MPSoC platform, the service curve* $[\beta^u, \beta^l]$*, we analytically explore the trade-off between buffer resource* B_{max} *(measured in number of frames) and the worst*

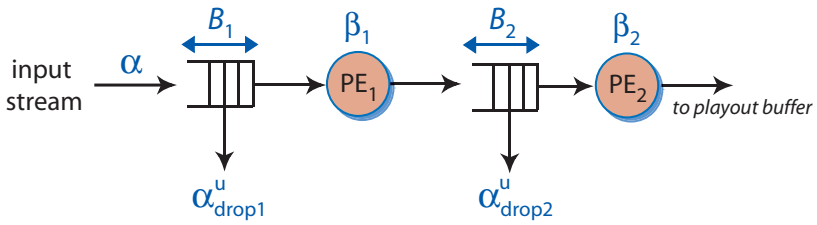


Figure 2.2: MPSoC setup with buffer constraints and frame drops

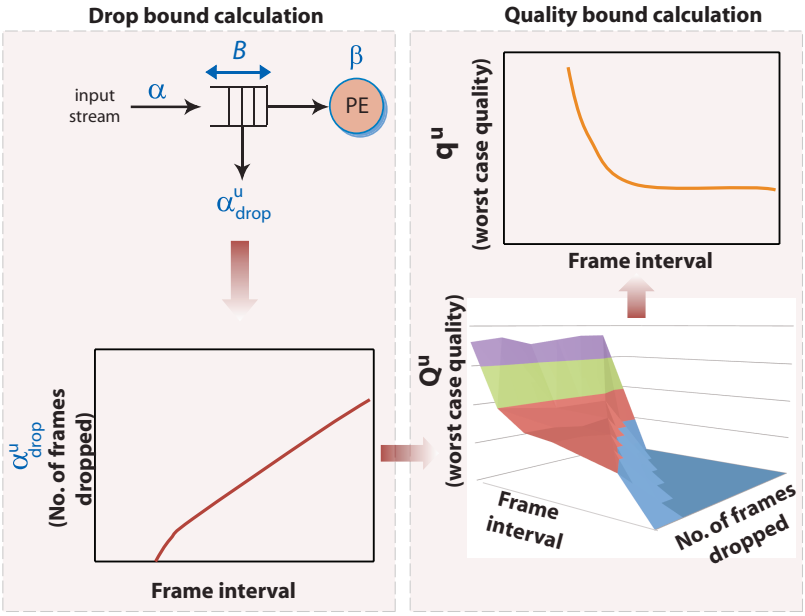


Figure 2.3: Overview of the Analytical Framework

case quality (quantified in terms of PSNR) of the decoded video.

Once this trade-off is explored for all the clips in the library, the system designer can appropriately choose the minimal buffer resource required to satisfy an acceptable quality constraint. The overall analytical framework consists of two stages as shown in Fig. 2.3, namely the *Drop bound calculation* stage and the *Quality bound calculation* stage. These two stages are described briefly next.

Drop bound calculation: The first stage formally derives the worst case frame drop bound α^u_{drop} for the droppable part of the buffer, with size B_{max}. This analysis is based on concepts from network

calculus. Specifically, it computes the bounds on the number of frames that are processed in an incoming stream when the arrival curves $[\alpha^u, \alpha^l]$, service curves $[\beta^u, \beta^l]$ and buffer size B_{max} for a single PE are given. Our computation is based on the idea of a virtual processor controlling the admission of frames into the buffer such that the buffer effectively acts as one with no drops, i.e., once an appropriate number of frames are dropped from the stream, the finite and constrained buffer will never overflow, thereby emulating an infinite buffer. We also compute the bounds on the service offered by the virtual processor to the incoming stream. This can be used to compute the worst case bound on the number of frame drops in any interval of time. However, we convert the time interval based computation of frame drop bounds into frame interval based bounds $\alpha^u_{dropF}(F)$, where F is the frame interval window and $1 \leq F \leq F_{total}$. Here, $\alpha^u_{dropF}(F)$ is the upper bound on the number of frames dropped in a window of F consecutive frames and F_{total} is the total number of frames in the clip. The detailed formulation will be shown in Section 2.2.3.

The useful feature of this stage is that it allows the analysis of multiple PEs in pipeline with buffer constraints to be done compositionally. In other words, one can compute the bounds on the arrival curve to the next stage. The computed arrival curve can then be used to derive the frame drop bounds in the next stage. These frame drop bounds computed at various stages (with constrained buffer resources) can be finally summed up to obtain the overall bound on the frame drops.

Quality Bound Calculation: Once the frame drop bounds are known, we compute a frame interval based worst-case bound on quality in terms of PSNR. Towards this, a parameter called the worst-case quality surface, denoted by Q^u, is constructed for each video clip. Q^u is defined as below.

Definition 4 ***Worst-case quality surface (Q^u).*** *For any frame interval F, the worst-case quality surface $Q^u(f,F)$, for all $0 \leq f \leq F$, is the worst-case quality of the video if f frames are dropped in any window of F consecutive frames.*

All dropped frames are replaced by immediately preceding and successfully processed frames called *concealment frames*. The amount of quality loss depends on the MSE between the dropped and concealment frames. The resultant quality is measured in terms of PSNR, which in turn depends on the MSE between the dropped and concealment frames. We find all possible concealment frames for

a dropped video and analyze which concealment frame results in maximum error or worst quality degradation.

B_{max} vs. quality trade-off: The final goal of the framework is to explore the trade-off between the maximum buffer capacity B_{max} and the quality for each video clip in the library. Once this trade-off is available for all the clips in the library, the system designer can take a well-informed decision on the appropriate buffer size. In order to derive this trade-off, we use the frame drop bound α^u_{dropF} and map it into the worst case quality surface $Q^u(f,F)$ where f is replaced by the value α^u_{dropF}. Therefore, the quality bound calculation is a mapping from a three dimensional (3D) space to a two dimensional (2D) space shown as

$$q^u(F) = Q^u(\alpha^u_{drop}(F), F) \tag{2.1}$$

where $q^u(F)$ is the worst-case quality bound for the video clip. This mapping is shown in Fig. 2.3, where the frame drop bounds are shown at the bottom left hand side and the worst-case quality space is shown on the bottom right hand side. The final worst-case quality bound for a video clip is shown in the top right hand side of Fig. 2.3.

2.2.2 Partitioning arrival and service curves

In this work, we study the effect of frame drops in the context of a video clip being processed by the associated decoder application. As we are more interested in studying the effect of frame drops on quality degradation, we intend to analyze the drop of those frames that least affect the quality degradation. It has been observed in MPEG-2 or MPEG-4 decoders that B frames are generally the least significant when compared to I and P frames as the loss of B frames results in least quality degradation when compared to I and P frames. Moreover, many video clips are encoded with a IPBBPBBP... frame pattern, where a large percentage of B frames exist. Therefore, we analyze the effect of only the B-frame drops. If there are videos encoded without B frames, then P frames can be dropped. In this case, the framework will remain the same. Consequently, the system model for the platform architecture consists of two kinds of buffers in front of each PE depending on whether B frame drops are allowed or not. This is shown in Fig. 2.4. If B frame drop is allowed, then we

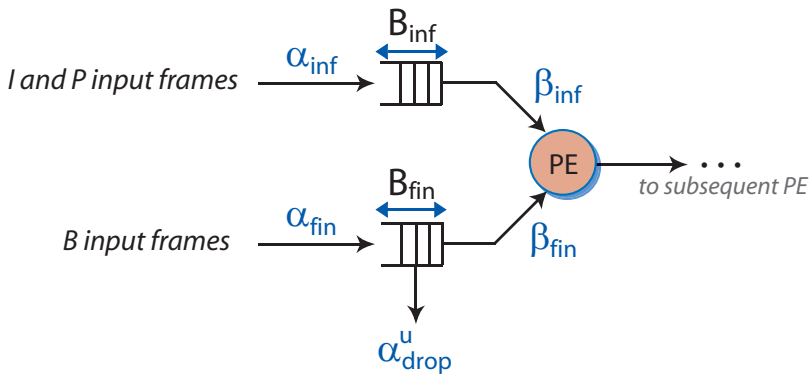


Figure 2.4: System model with infinite and finite buffer for a single PE

have a finite buffer called the B frame buffer (B_{fin})) and another finite buffer called the IP frame buffer (B_{inf})) that does not have any drop. The buffer size required for an IP frame buffer can be computed using conventional Network Calculus technique ([48]).

The existence of two buffers makes it necessary to partition the arrival curves and service offered to the two sets of frames. As illustrated in Fig. 2.4, the original arrival curves of the input stream are partitioned into $\alpha_{inf} = [\alpha^u_{inf}, \alpha^l_{inf}]$ and $\alpha_{fin} = [\alpha^u_{fin}, \alpha^l_{fin}]$, which correspond to the arrival curves of the I and P frames together and of the B frames, respectively. Similarly, the service curves offered by the PE are partitioned into $\beta_{inf} = [\beta^u_{inf}, \beta^l_{inf}]$ and $\beta_{fin} = [\beta^u_{fin}, \beta^l_{fin}]$, which correspond to the service curves offered to the I and P frames and to the B frames, respectively. As the I and P frames share the same buffer B_{inf} with no frame drops, their buffer size can be computed directly from α_{inf} and β_{inf} using the technique in [48]. On the other hand, the B frames can be dropped; their drop bound (α^u_{drop}) can be computed using α_{fin}, β_{fin} and B_{fin}.

The algorithm to compute the partitioned arrival curve for B frames is shown as Algorithm 1. The arrival curves for I and P frames can also be computed in the same manner. However, due to the existence of partitioned arrival curves and two buffers now, the PE needs to be given information about what is the order in which the frames are processed. This is generally the order in which the frames are encoded and sent out in a video stream.

In Algorithm 1, we compute the arrival curves $[\alpha^u_{fin}, \alpha^l_{fin}]$ for the B frames. Lines 4-12 compute the arrival times of each B frame (denoted by b_arr) in the video clip. F_{total} and B_CNT are the total

Algorithm 1 Computing partitioned arrival curve for B frame

Input: $frsize(B\_CNT)$ - Size of each frame in bits;
Output: $[\alpha^u_{fin}, \alpha^l_{fin}]$

1: $b\_arr(cnt) \leftarrow 0$ for all $0 \leq cnt \leq B\_CNT$, $ip\_arr \leftarrow 0$;
2: $btime\_max(k) \leftarrow 0$, $btime\_min(k) \leftarrow 0$ for all $1 \leq k \leq B\_CNT$
3: —Computing the arrival time of each B frame—
4: **for** $i = 1$ to F_{total} **do**
5: **if** $Bframe$ **then**
6: $b\_arr(cnt) = frsize(i)/RATE + ip\_arr$
7: $ip\_arr = 0$
8: $cnt = cnt + 1$
9: **else**
10: $ip\_arr = ip\_arr + frsize(i)/RATE$
11: **end if**
12: **end for**
13: —Find max and min arrival times for k consecutive B frames—
14: $btime\_max(k) = \max_{\forall i}\{\sum_{j=1}^{k} b\_arr(j+i)\}, 0 \leq i \leq B\_CNT - k$
15: $btime\_min(k) = \min_{\forall i}\{\sum_{j=1}^{k} b\_arr(j+i)\}, 0 \leq i \leq B\_CNT - k$
16: —Find upper and lower arrival curves for B frames—
17: $\alpha^u_{fin}(t) = \begin{cases} \max\{k-1\} & : btime\_min(k) < t \\ \min\{k\} & : btime\_min(k) \geq t \end{cases}$
18: $\alpha^l_{fin}(t) = \begin{cases} \max\{k-1\} & : btime\_max(k) < t \\ \min\{k\} & : btime\_max(k) \geq t \end{cases}$

number of frames and B frames, respectively, in the video clip. The input bit rate of the video clip is denoted by $RATE$. We then find the maximum and minimum arrival times for k consecutive B frames. This is shown in lines 14-15. Finally, the arrival curves are computed as in lines 17-18. The upper bound on the B frame arrival curve is obtained from the minimum arrival time required for k consecutive frames such that they satisfy the condition in line 17. Similarly, the lower bound of B frame arrival curve is determined by the maximum arrival time required for k consecutive frames.

The service curves $[\beta^u_{fin}, \beta^l_{fin}]$ for B frames are also computed as the arrival curves have been computed. The only difference here is that instead of the arrival times of B frames, we compute the time required for the execution of the tasks mapped on the PE for each B frame, i.e., b_arr is changed to execution time. Execution time also depends upon the frequency allocated to the PE. Subsequently, we compute the maximum and minimum execution time required for k consecutive B frames. Fi-

nally, we compute the service curves in a similar manner as we did for arrival curves. The arrival and service curves used in the following sections are the partitioned arrival and service curves for B frames presented here.

2.2.3 Bounds on dropped frames

In this section, we present a method for computing bounds on the number of frames that are dropped due to an overflow at a buffer. We first present the modeling idea and the basic concepts, and then present the details of how drop bounds can be obtained.

A single buffer case. Consider an input stream that is processed by a single processing element (PE). Suppose the input buffer that stores the incoming frames of the stream before being processed by the PE, has a finite capacity of B frames.If the buffer is full when a frame arrives, the oldest frame at the head of the buffer will be dropped and the newly arrived frame will be enqueued at the end of the buffer. We are interested in the maximum bounds on the frames that can be dropped over any interval of a given length. The system architecture is shown in the top part of Figure 2.5. In the figure, $a_1(t)$ denotes the input arrival pattern of the frame, i.e., $a_1(t)$ gives the number of frames that arrive over the time interval $(0,t]$. Similarly, $a_3(t)$ gives the number of output frames corresponding to $a_1(t)$, respectively, over the interval $(0,t]$.

To model the buffer refresh at the input buffer, we use a virtual processor P_v that serves as an admission controller, as shown in the bottom part of Figure 2.5. The virtual processor P_v splits the input stream $a_1(t)$ into two disjoint streams: the former, $a_2(t)$, contains the frames that will go through the system, and the latter, $a_2'(t)$, contains the frames that will be dropped, such that there are no overflows at the buffer.

Based on this transformed system, we give the relationship between $a_1(t)$ and $a_2(t)$, and the bounds on $a_3(t)$, stated by Lemma 2.2.1 and 2.2.2.

In what follows, g^* denotes the sub-additive closure of g, defined by $g^* = \min\{g^n \mid n \geq 0\}$, where $g^0(0) = 0$ and $g^0(t) = +\infty$ for all $t > 0$, and $g^{n+1} = g^n \otimes g$ for all $n \in \mathbb{N}$, $n \geq 0$. Further, $\mathcal{I}$ denotes the linear idempotent operator, i.e.,

$$\mathcal{I}_{a_1}(x)(t) = \inf_{0 \leq s \leq t} \{x(s) + a_1(t) - a_1(s)\}.$$

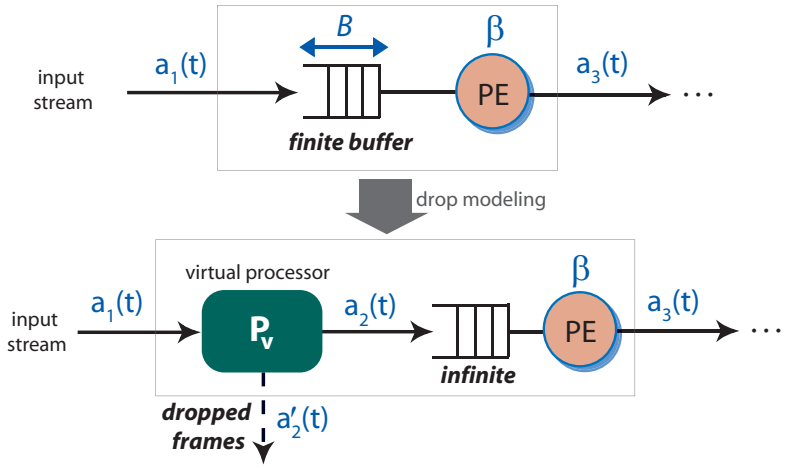


Figure 2.5: Modeling systems with drop due to buffer overflow.

Lemma 2.2.1 *Suppose f is the mapping from $a_2(t)$ to $a_3(t)$, i.e., $a_3 = f(a_2)$. Then, $a_2 = \big(\mathcal{I}_{a_1} \circ (f+B)\big)^*(a_1)$.*

Proof Since none of the items in a_2 is overwritten, for all $t \geq 0$, $b(t) = a_2(t) - a_3(t) \leq B$, or $a_2 \leq a_3 + B$. Let f be the function that maps the input a_2 to the output a_3, assuming f is monotonic. Then $a_3 + B = f(a_2) + B = (f+B)(a_2)$.

Further, the number of items that pass the admission test at P_v (i.e., not overwritten) over any time interval $(s,t]$ is no more than the number of original items that enter the system over the same interval. In other words,

$$\forall t \geq 0,\ \forall 0 \leq s \leq t:\ a_2(t) - a_2(s) \leq a_1(t) - a_1(s).$$

Recall that $\mathcal{I}_{a_1}(a_2)(t) = \inf\{a_2(s) + a_1(t) - a_1(s) \mid 0 \leq s \leq t\}$. Then, $a_2 \leq \mathcal{I}_{a_1}(a_2)$. Hence,

$$a_2 \leq \min\{a_1,\ \mathcal{I}_{a_1}(a_2),\ (f+B)(a_2)\} \tag{2.2}$$

$$\Leftrightarrow\ a_2 \leq a_1 \oplus \big(\mathcal{I}_{a_1} \oplus (f+B)\big)(a_2). \tag{2.3}$$

Hence, the input function of the items that actually go through the system is the maximum solution for Eq. (2.3).

From Theorem 4.3.1 in [48], the inequality $h \leq g \oplus f(h)$ has one unique maximal solution, given by

$h = f^*(g)$. Apply this theorem into Eq. (2.3), we obtain

$$a_2 = \big(\mathcal{I}_{a_1} \oplus (f + B)\big)^*(a_1).$$

This proves the lemma.

The next lemma further gives the bounds on $a_3(t)$ based on the relationship established in Lemma 2.2.1.

Lemma 2.2.2 *Consider the system in Figure 2.5. Denote α as the arrival curves of the input stream, β as the service curves of the PE, and B as the size of the buffer. The output stream of the system is bounded by the arrival curves $\alpha' = (\alpha^{u'}, \alpha^{l'})$, defined by*

$$\alpha^{u'} = \min\big\{ \big(\alpha^u \otimes \beta^u_{\mathsf{eff}}\big) \oslash \beta^l_{\mathsf{eff}},\ \beta^u_{\mathsf{eff}} \big\},$$
$$\alpha^{l'} = \min\big\{ \big(\alpha^l \oslash \beta^u_{\mathsf{eff}}\big) \otimes \beta^l_{\mathsf{eff}},\ \beta^l_{\mathsf{eff}} \big\}.$$

where

$$\beta^u_{\mathsf{eff}} = \big(\alpha^u \otimes \beta^u + B\big)^* \otimes \alpha^u \otimes \beta^u$$
$$\beta^l_{\mathsf{eff}} = \big(\alpha^l \otimes \beta^l + B\big)^* \otimes \alpha^u \otimes \beta^l.$$

Proof Let $\beta^u_v = \alpha^u \otimes (\alpha^u \otimes \beta^u + B)^*$ and $\beta^l_v = \alpha^u \otimes (\alpha^l \otimes \beta^l + B)^*$. We will prove that $a_1 \otimes \beta^l_v \leq a_2 \leq a_1 \otimes \beta^u_v$.

Indeed, from Lemma 2.2.1, we have $a_2 = \big(\mathcal{I}_{a_1} \oplus (f + B)\big)^*(a_1)$. This implies

$$a_2 = \big(\mathcal{I}_{a_1} \circ (f + B)\big)^* \circ \mathcal{I}_{a_1}(a_1).$$

Since β^l is the lower service curve of the PE and $a_3 = f(a_2)$, we have $f(a_2) = a_3 \geq a_2 \otimes \beta^l$, which can be rewritten as $f \geq \mathcal{C}_{\beta^l}$, or $f + B \geq \mathcal{C}_{\beta^l} + B$. Similarly, α^l is the lower arrival curve of A_1 implies that $a_1(t) - a_1(s) \geq \alpha^l(t-s)$. Thus, $\mathcal{I}_{a_1}(a_2) \geq \alpha^l \otimes a_2$, or $\mathcal{I}_{a_1} \geq \mathcal{C}_{\alpha^l}$. Hence, $a_2 \geq \big(\mathcal{C}_{\alpha^l} \oplus (\mathcal{C}_{\beta^l} + B)\big)^*(a_1)$, which imply

$$a_2 \geq \big(\alpha^l \otimes \mathcal{C}_{\beta^l} + B\big)^* \otimes a_1$$

$$\Leftrightarrow\ a_2 \geq \big(\alpha^l \otimes \mathcal{C}_{\beta^l} + B\big)^* \otimes \alpha^u \otimes a_1$$

because $f \otimes g \leq \min\{f, g\}$ or finally $a_2 \geq \beta_v^l \otimes a_1$.

By similar argument, we have $f + B \leq \mathcal{C}_{\beta^u} + B$ and $\mathcal{I}_{a_1} \leq \mathcal{C}_{\alpha^u}$. Thus,

$$a_2 \leq \left(\mathcal{C}_{\alpha^u} \circ (\mathcal{C}_{\beta^u} + B)\right)^* \circ \mathcal{C}_{\alpha^u}(a_1).$$

which can be rewritten as

$$a_2 \leq (\alpha^u \otimes \beta^u + B)^* \otimes \alpha^u \otimes a_1.$$

In other words, $a_2 \leq \beta_v^u \otimes a_1$. Hence,

$$a_1 \otimes \beta_v^l \leq a_2 \leq a_1 \otimes \beta_v^u$$

Combine the above with the fact that $a_2 \otimes \beta^l \leq a_3 \leq a_2 \otimes \beta^u$, we obtain

$$a_1 \otimes \beta_v^l \otimes \beta^l \leq a_3 \leq a_1 \otimes \beta_v^u \otimes \beta^u$$

or

$$a_1 \otimes \beta_{\mathsf{eff}}^l \leq a_3 \leq a_1 \otimes \beta_{\mathsf{eff}}^u.$$

In other words, $\beta_{\mathsf{eff}} = (\beta_{\mathsf{eff}}^u, \beta_{\mathsf{eff}}^l)$ is a valid pair of upper and lower service curves that effectively transform the input a_1 to the output a_3. The output arrival curves that bound a_3 can therefore computed using standard Network Calculus techniques from α and β_{eff}, which are given by $\alpha' = (\alpha^{u'}, \alpha^{l'})$. This proves the lemma.

Based on the above results, Lemma 2.2.3 gives the bounds on the dropped input frames.

Lemma 2.2.3 *Suppose $\alpha = (\alpha^u, \alpha^l)$ are the arrival curves of an input stream, $\beta = (\beta^u, \beta^l)$ are the service curves of the PE, and B is the size of the input buffer. Then, the number of input frames that can be dropped over any interval of length $\Delta \geq 0$ is upper bounded by $\alpha_{drop}^u(\Delta)$, defined by*

$$\alpha_{drop}^u = (\alpha^u - \beta_v^l) \overline{\otimes} 0$$

where $\beta_v^l \stackrel{\text{def}}{=} (\alpha^l \otimes \beta^l + B)^* \otimes \alpha^u$.

Proof Suppose $a_1(t)$ is an input arrival pattern modeled by α, and $a_2(t)$ and $a_3(t)$ are defined as in Figure 2.5. Denote $g = (f+B)(a_1)$. From Lemma 2.2.1,

$$\begin{aligned} a_2(t) &= \big(\big(\mathcal{I}_{a_1} \circ (f+B)\big)^*(a_1)\big)(t) \\ &= \inf_{n\in\mathbb{N}} \inf_{0\le u_{2n}\le\cdots\le u_1\le t} \big\{a_1(t) - a_1(u_1) + g(u_1) - a_1(u_3) \\ &\quad + g(u_3) - \cdots - a_1(u_{2n-1}) + g(u_{2n-1})\big\}. \end{aligned}$$

Hence, the number of frames that will be processed by the PE in the interval $(0,t]$, denoted by $A_t(\Delta)$ is given by:

$$\begin{aligned} A_t(\Delta) &\stackrel{\text{def}}{=} a_2(t+\Delta) - a_2(t) \ge a_1(t+\Delta) - a_1(t) \\ &\quad + \inf_{n\in\mathbb{N}} \inf_{t\le u_{2n}\le\cdots\le u_1\le t+\Delta} \big\{-a_1(u_1) + g(u_1) \\ &\quad - a_1(u_3) + g(u_3) - \cdots - a_1(u_{2n-1}) + g(u_{2n-1})\big\}. \end{aligned}$$

On the other hand, since β^l is the lower service curve of the PE and $a_3 = f(a_2)$, we have $f(a_2) = a_3 \ge a_2 \otimes \beta^l$, which can be rewritten as $f \ge \mathcal{C}_{\beta^l}$. Thus,

$$g = (f+B)(a_1) \ge (\mathcal{C}_{\beta^l} + B)(a_1) = \beta^l \otimes a_1 + B.$$

As a result,

$$
\begin{aligned}
A_t(\Delta) &= a_1(t+\Delta) - a_1(t) + \inf_{n\in\mathbb{N}} \inf_{t<u_{2n}\le\cdots\le u_2\le u_1\le t+\Delta} \\
&\qquad \sum_{i=1}^{n}\Big\{-a_1(u_{2i-1}) + \beta^l(u_{2i-1}-u_{2i}) + B + a_1(u_{2i})\Big\} \\
&= a_1(t+\Delta) - a_1(t) + \inf_{n\in\mathbb{N}} \inf_{t<u_{2n}\le\cdots\le u_1\le u_0=t+\Delta} \\
&\qquad \sum_{i=1}^{n}\Big\{-a_1(u_{2i-2}) + a_1(u_{2i}) + a_1(u_{2i-2}) \\
&\qquad - a_1(u_{2i-1}) + \beta^l(u_{2i-1}-u_{2i}) + B\Big\} \\
&= \inf_{n\in\mathbb{N}} \inf_{t<u_{2n}\le\cdots\le u_1\le u_0=t+\Delta} \\
&\qquad \sum_{i=1}^{n}\Big\{a_1(u_{2i-2}) - a_1(u_{2i-1}) + \beta^l(u_{2i-1}-u_{2i}) + B\Big\} \\
&\ge \inf_{n\in\mathbb{N}} \inf_{t<u_{2n}\le\cdots\le u_1\le u_0=t+\Delta} \sum_{i=1}^{n}\Big\{\alpha^l(u_{2i-2}-u_{2i-1}) \\
&\qquad + \beta^l(u_{2i-1}-u_{2i}) + B\Big\} \\
&\ge \inf_{n\in\mathbb{N}} \inf_{t<u_{2n}\le\cdots\le u_1\le u_0\le t+\Delta} \sum_{i=1}^{n}\Big\{(\alpha^l\otimes\beta^l + B)(u_{2i-2}-u_{2i})\Big\} \\
&\ge (\alpha^l\otimes\beta^l + B)^*(t+\Delta-t) \\
&= (\alpha^l\otimes\beta^l + B)^*(\Delta).
\end{aligned}
$$

Let $\beta_v^l = (\alpha^l\otimes\beta^l + B)^*\otimes\alpha^u$, then $A_t \ge \beta_v^l$ for all $t\ge 0$. In addition, $a_1(t+\Delta)-a_1(t)\le\alpha^u(\Delta)$ for all $t\ge 0$ and $\Delta\ge 0$. Hence, the number of dropped frames in the interval $(t,t+\Delta]$ satisfies

$$
\begin{aligned}
L_t(\Delta) &\stackrel{\text{def}}{=} a_1(t+\Delta) - a_1(t) - A_t(\Delta) \le \alpha^u(\Delta) - \beta_v^l(\Delta) \\
&\le \big((\alpha^u - \beta_v^l)\,\overline{\otimes}\,0\big)(\Delta) = \alpha_{drop}^u(\Delta).
\end{aligned}
$$

In other words, the number of dropped frames over any interval of length Δ is upper bounded by $\alpha_{drop}^u(\Delta)$.

Lemma 2.2.4 *Define α, β, B and α_{drop} as in Lemma 2.2.3. Denote $\delta^u(k) = \min\{\Delta\ge 0 \mid \alpha^l(\Delta)\ge k\}$ and $\delta^l(k) = \min\{\Delta\ge 0 \mid \alpha^u(\Delta)\ge k\}$ for all $k\in\mathbb{N}$. Then, for any given non-negative integer k, the*

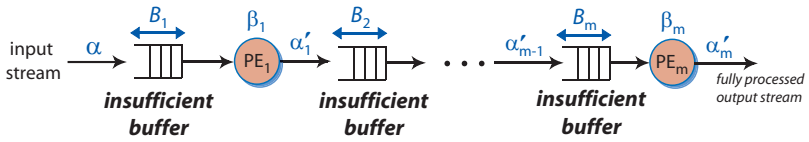


Figure 2.6: A sequence of PEs with insufficient buffers.

number of frames that can be dropped over any k consecutive input frames is upper bounded by $\alpha^u_{dropF}(k)$*, where*

$$\alpha^u_{dropF}(k) \stackrel{\text{def}}{=} \min\{k, (\alpha^u_{drop} \circ \delta^u)(k)\}, \tag{2.4}$$

Proof For any given integer $k \geq 0$, the maximum time required for k consecutive input frames to arrive is

$$\min\{\Delta \geq 0 \mid \alpha^l(\Delta) \geq k\} \stackrel{\text{def}}{=} \delta^u(k).$$

From Lemma 2.2.3, the number of frames that can be dropped over any interval of length $\delta^u(k)$ is at most $\alpha^u_{drop}(\delta^u(k))$, i.e., at most $\left(\alpha^u_{drop} \circ \delta^u\right)(k)$. Thus, the number of frames that can be dropped over every k consecutive input frames is at most $\left(\alpha^u_{drop} \circ \delta^u\right)(k)$. Since there can be no more than k frames dropped over every k consecutive input frames, the number of frames that can be dropped over every k consecutive input frames is at most

$$\min\{k, \left(\alpha^u_{drop} \circ \delta^u\right)(k)\} \stackrel{\text{def}}{=} \alpha^u_{dropF}(k).$$

This proves the lemma.

Multiple buffers case. Consider a system consisting of m PEs (as shown in Fig. 2.6). The input stream that is processed by a sequence of m PEs, $PE_1, \ldots, PE_m$, where the input buffer at PE_i has a finite capacity of B_i (frames). The arrival curves of the input stream and the service curves of PE_i are denoted by α_1 and β_i, respectively, as illustrated in Figure 2.6. Given such architecture, we would like to compute the maximum bounds on the total number of frames that are dropped within the system.

Since the frames that are dropped at the PEs are disjoint, the number of frames that are dropped in the system is the total number of frames that are dropped at each PE. The maximum number of the frames that are dropped at PE_1 over any interval of a given length Δ, denoted by N^1_Δ, is derived using

Lemma 2.2.4. The maximum number of frames N_Δ^i that are dropped over any interval of length Δ at each subsequent PE_i for all $2 \leq i \leq m$, can be computed in a compositional manner: first, compute the output arrival curves α'_{i-1} after being processed by PE_{i-1} by applying Lemma 2.2.2; then, compute the drop bounds N_Δ^i at PE_i using Lemma 2.2.4, with α'_{i-1} as the input arrival curves, β_i as the service curves and B_i as the input buffer size. We repeat this process until we reach the last PE. The maximum number of input frames that are dropped within the system over any interval of length Δ is then the summation of all the computed drop bounds, which is given by $N_\Delta^1 + \cdots + N_\Delta^m$.

2.2.4 Worst-case bound on Quality

In the previous section, we presented how the bounds are computed for dropped frames. In this section, we use this bound to compute the worst-case quality in terms of PSNR. In order to find the worst-case quality for a video clip, we need to construct a worst-case quality 3-D space as shown in Fig.2.3. This is a surface that maps the frame interval based drop bound from the previous section to a frame interval based quality bound. Let us denote this mapping function as Q^u and the frame interval based quality bound as q^u. Then the mapping can be depicted as $Q^u : \alpha^u_{dropF} \rightarrow q^u$. However, in order to perform this mapping, the worst-case quality surfaced Q^u needs to be constructed. We construct this surface by taking consecutive frame intervals as windows. For each frame interval in the entire video, we find the maximum noise error experienced if any number of frames upto the frame window size is lost. This quantity is architecture independent and depends only on the nature of the clip. In our case, we slide the frame interval window from $1 \rightarrow N_{tot}$. Within each frame interval window F, we find the worst-case PSNR value or the highest MSE value from Equation.1.1 for every value f, such that $0 \leq f \leq F$. Here f is the number of frames that were dropped in the frame interval F. Therefore, we construct the worst-case quality surface $Q^u(f,F)$. This procedure is shown in Algorithm 2.

The MSE_{max} structure containing the maximum MSE values for B frames is calculated taking all possible concealment frames into consideration. For example, let us take the order of frames in group of pictures (GOP) as shown in Fig. 1.1. In particular, for the 4th B frame, there are three different possible concealment frames. If the 3rd B frame is not dropped, then it will replace the 4th B frame. If the 3rd B frame is dropped, then the P/I frame will replace 4th B frame in that order.

Algorithm 2 Computing worst-case quality surface for a video clip

Input: MSE_{max} - Maximum MSE values for B frames if replaced by possible preceding I/P frames. MSE values for I/P frames are set to 0.

Output: $Q^u(f,F)$ - Worst-case quality surface, f is the number of frames dropped in a frame interval of F

1: Record the frame indices

2: Sort MSE_{max} structure in descending order preserving the frame indices $\rightarrow MSE_{maxsort}$

3: Find F values within frame index range i to $(i+F-1)$ in $MSE_{maxsort}$: $\forall i, \forall F$ and $1 \leq i \leq N_{tot}-F+1$ and $0 \leq F \leq N_{tot} \rightarrow MSE_{maxF}(i,n,F)$ where $0 \leq n \leq F$

4: $MSE^u(f,F) = \max_{\forall i} \{ \sum_{n=0}^{f} MSE_{maxF}(i,n,F) \}$

5: $Q^u(f,F) = 10 \times log_{10} \frac{(255 \times 255 \times F)}{(MSE^u(f,F))}$

Since P frames are not dropped in our setting, P frame replaces the 4th B-frame if the 3rd B frame is lost. Therefore, MSE_{max} is constructed taking all such possible concealment frames.

Lines 1 and 2 record the indices of the B frame in the GOP decoding order and then sort the frames in decreasing order of the MSE values in MSE_{max} structure, while retaining the original indices after sorting. For each frame interval window F, the frame index ranges from i to $i+F-1$ where i is the variable used for sliding across the video clip. We search for the F frames within this index range from the sorted MSE structure shown as $MSE_{maxsort}$ (Line 3). We slide the window across the entire video clip and find the F frames for each i. These quantities are stored in the structure $MSE_{max}(i,n,F)$ where $0 \leq n \leq F$. The upper bound on MSE is then computed by searching for the maximum value across all windows of size F and for every drop count f which ranges from $0 \leq f \leq F$ (Line 4). Once the upper bound $MSE^u(f,F)$ is computed, the worst-case quality surface $Q^u(f,F)$ can be computed as given in Line 5.

Lemma 2.2.5 *A computation of $Q^u(f,F)$ is exponential in the total number of frames in the clip N_{tot} i.e. the complexity of computing the worst-case quality surface is $O(N_{tot} \times 2^{N_{tot}})$.*

Proof For any frame interval window F, where $1 \leq F \leq N_{tot}$, the number of iterations required to check the maximum MSE if f frames were dropped in that frame interval, where $0 \leq f \leq F$ is given by $\binom{F}{f}$. Here $\binom{F}{f} = \frac{F!}{(F-f)!f!}$. For all values of f, the total number of iterations required is $\sum_{f=0}^{F} \binom{F}{f}$. It

is well known from concept of combinations that $\sum_{f=0}^{F} \binom{F}{f} = 2^F$. As F ranges from 1 to N_{tot}, it is easy to prove that $\sum_{F=1}^{N_{tot}} 2^F = O(2^{N_{tot}})$ using sum of a geometric series. Finally since this frame interval has to be slid across the entire clip, it needs to be done N_{tot} times. Hence the total complexity of a straightforward method is $O(N_{tot} \times 2^{N_{tot}})$.

In Algorithm.2, it is seen that the complexity is $O(N_{tot}^3)$. Hence, the scheme we propose to construct the worst-case quality surface is more efficient.

2.2.5 Case Study (MPEG-2 Decoder)

In this section, we evaluate our proposed analytical framework using an MPEG-2 decoder application. In this case study, the MPEG-2 decoder tasks are mapped onto the two PEs in the MPSoC architecture shown in Fig. 2.2. The tasks mapped are Variable Length Decoding (VLD), Inverse Quantization (IQ), Motion Compensation (MC) and Inverse Discrete Cosine Transform (IDCT). VLD and IQ are mapped to PE_1 while MC and IDCT are mapped to PE_2. According to our setup, each buffer in Fig. 2.2 is composed of two buffers (as shown in Fig. 2.4) to separate the B frames from I/P frames. We only analyze the drops for B frames and therefore, we analyze only the B frame buffer. The buffer used for I/P frames is not analyzed here because it can be done using conventional Network Calculus techniques ([48]). PE_1 is allocated a frequency of 40 MHz, whereas PE_2 is allocated a frequency of 100 MHz. The various B frame buffer sizes used in the first stage are set to be 30 frames, 60 frames, 90 frames, 120 frames and 150 frames. The B frame buffer sizes used in the second stage are the same as in the first stage. However, the analysis of drops in the second stage is done by fixing the first stage buffer size to 90.

The cycle requirements for each task on the model of a processor was obtained using the SimpleScalar simulator ([8]). Here, we use a MIPS-like processor model using the Portable instruction set architecture (PISA). We use three MPEG-2 video clips in our experiments, namely, *susi*_080, *time*_080 and *orion*_2. The first two videos are taken from [1], where both have a total of 450 frames, i.e., $N_{tot} = 450$ with 1320 macroblocks (MBs) in each frame. The first clip is a motion video and the second one is a still video. The third video, taken from [64], is a combination of both motion and still frames. It has a total of 1171 frames, i.e., $N_{tot} = 1171$ with 1350 MBs in each

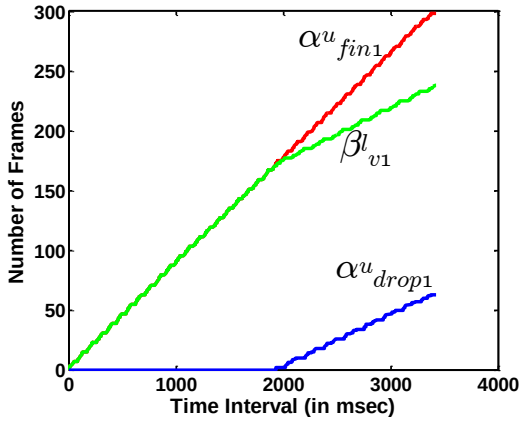
frame. All the three video clips have a bit rate of 8 Mbps.

2.2.5.1 First stage results

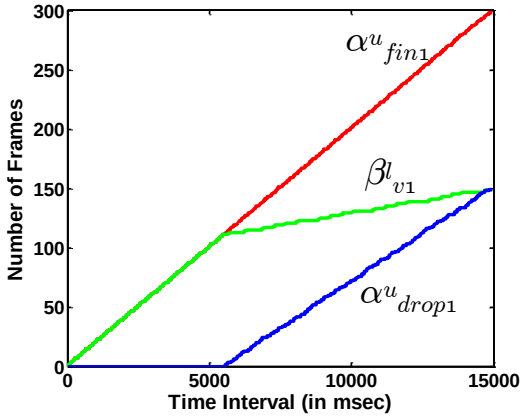
The first stage involves computing the drop bounds of the B frame buffer at PE_1 (denoted by B_{fin1}), which is of size B_{max1}. The arrival curves at the input of B_{fin1} are $\alpha_{fin1} = [\alpha^u_{fin1}, \alpha^l_{fin1}]$ as computed in Section 2.2.2. Similarly, the service curves offered to the frames in B_{fin1} are $\beta_{fin1} = [\beta^u_{fin1}, \beta^l_{fin1}]$.

Arrival curve, virtual processor service curve and drop bound (in time intervals): Fig. 2.7 shows the upper arrival curves of the B frames (α^u_{fin1}) and the lower service curve of the virtual processor (β^l_{v1}) for the three clips (computed using the techniques in Section 2.2.3). The worst case drop bound, α^u_{drop1}, obtained as a result of Lemma 2.2.3 is also shown in the three plots. In this experiment, $B_{max1} = 90$, which is in frames. It can be observed from the plots that, until a certain time interval, the drop bound is zero. After that interval, however, the drop bound increases. This is expected because the buffer size of 90 frames is insufficient to avoid buffer overflow. It is also seen that β^l_{v1} follows α^u_{fin1} until the former rises above the buffer size. From there onwards, β^l_{v1} starts dropping behind α^u_{fin1} as frames are dropped. Another interesting observation is that for still video *time*_080, β^l_{v1} is closer to α^u_{fin1} and hence, the drop bound is lower when compared to clips *susi*_080 and *orion*_2. However, it is interesting to notice that the video clip *orion*_2 has a higher drop bound value than *susi*_080. This is because the service required by the frames in *orion*_2 is higher than the the service required by the frames in *susi*_080 as the former has more macroblocks per frame.

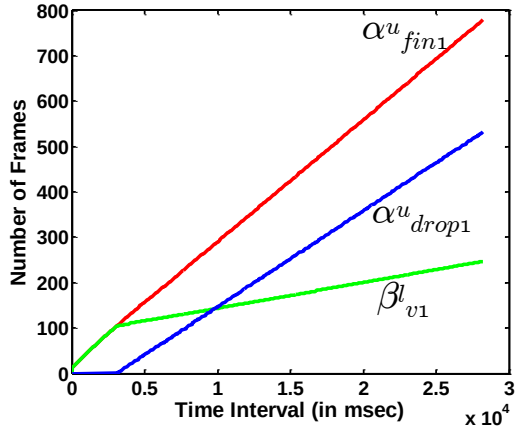
Validation of drop bounds (in frame intervals) with simulation: We validate the drop bounds $[\alpha^u_{dropF1}]$ computed using our analytical framework with the ones obtained by simulation. Here, the drop bound is in frame intervals and not time intervals. Once α^u_{drop1} is computed, $[\alpha^u_{dropF1}]$ can be computed according to Lemma 2.2.4. We show the comparison between simulation and analytical results for two buffer sizes, $B_{max1} = 60$ and $B_{max1} = 120$. It is clear from Fig. 2.8 that the analytical results emulate the simulation results very closely. Our analytical results are a little pessimistic because they consider the worst case in all the frame windows, whereas the simulation result depicts only one continuous run. It is also seen that $[\alpha^u_{dropF1}]$ decreases as the buffer size increases, which



(a)



(b)



(c)

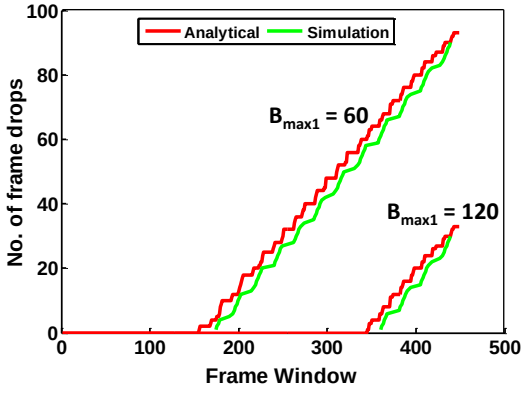
Figure 2.7: Generation of time interval based drop bound curves (α^u_{drop}) from the upper arrival (α^u) and lower virtual processor service (β^l_v) curves. Here $B_{max} = 90$. The three plots are for clips (a) *time*_080, (b) *susi*_080 and (c) *orion*_2.

is expected. It is interesting to note that the difference between simulation and analytical results is greater in *orion_2* than in the other two videos. The reason for this behaviour is that *orion_2* is a larger clip and the variability in the required service is larger. In the case of *susi_080* and *time_080*, the variability in required service is limited.

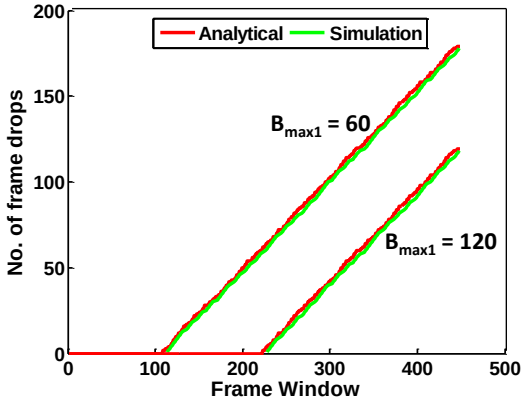
Worst-case quality surface: The worst-case quality surface computed using Algorithm 2 is presented for the three clips in Fig. 2.9. It is observed that the worst-case quality surface is an exponential surface as it represents the PSNR value for various frame drops within a frame interval. In all the Q^u plots shown, the PSNR value is highest when the least number of frames are dropped in the largest frame interval. The PSNR surface keeps falling from that point as the number of frames dropped increases and the frame interval decreases. This surface is an architecture independent feature of the video clips. According to Fig. 2.9, *time_080* has the highest Q^u values among all the video clips.

Comparison of q^u with simulation results: The comparison of frame interval based worst-case quality (q^u) is presented in Fig. 2.10(a), (b) and (c) for the three clips. The immediate observation from the plots is that, for *orion_2*, there is a considerable deviation of the analytical result from the simulation results in the lower frame intervals. This is because the clip is large and the analytical model considers the worst case across the entire clip. On the other hand, the simulation based result is the outcome of one continuous run. Therefore, if the worst case does not occur in the beginning of the clip, the deviation is large. However, the interesting point is that the curves converge closer towards the higher frame intervals. Hence, it is useful to use the higher frame intervals to explore the quality-buffer design space because they help to reduce the overestimation in buffer size required. However, even if overestimation exists, buffer dimensions can be reduced for a lower tolerable quality if the zero loss constraint need not be strictly adhered to.

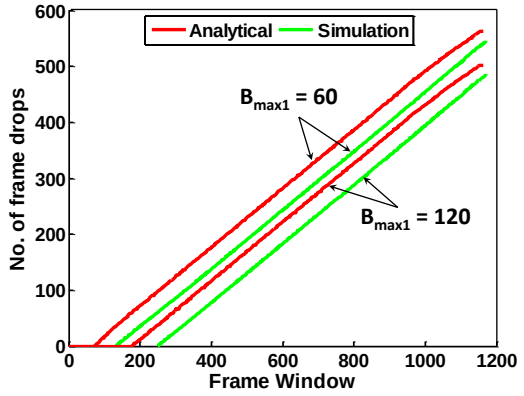
Variation of worst-case quality with buffer size: The variation of q^u with buffer size is shown in Fig. 2.11. As is expected, in Fig. 2.11(a), (b) and (c), q^u values increase as the maximum buffer capacity B_{max1} is increased. We explore the variation for five buffer sizes as shown in Fig. 2.11.



(a)

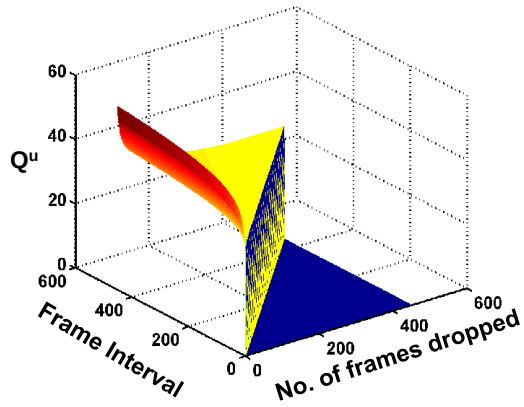


(b)

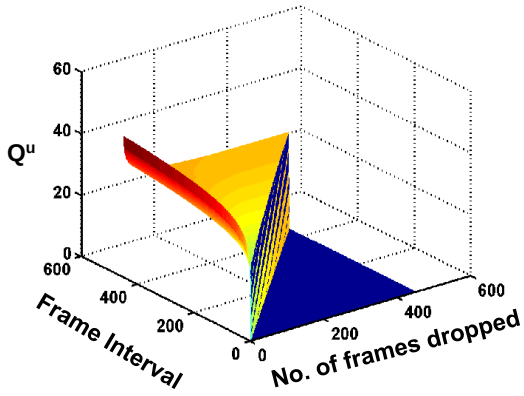


(c)

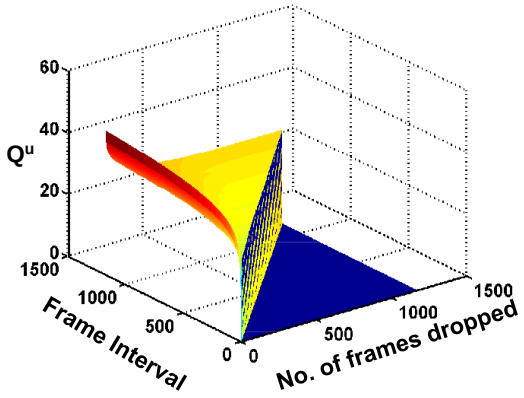
Figure 2.8: Comparison of Analytical and Simulation results of worst-case drop bound for two buffer capacities. The three plots are for clips (a) *time*_080, (b) *susi*_080 and (c) *orion*_2.



(a)

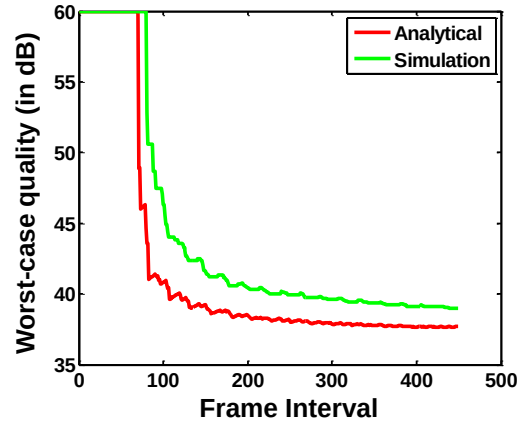


(b)

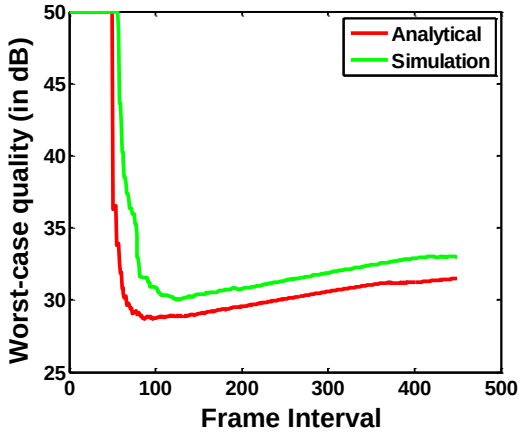


(c)

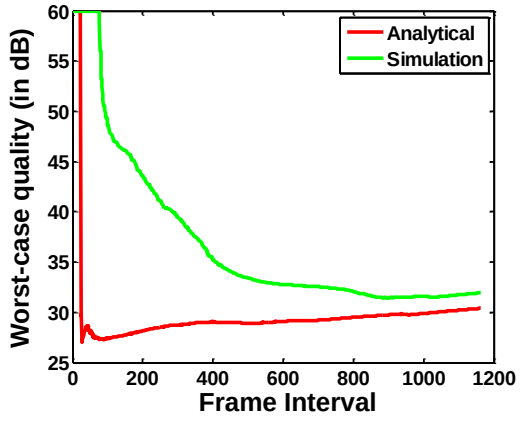
Figure 2.9: Worst case quality surface (Q^u in dB) for the clips (a) *time*_080, (b) *susi*_080 and (c) *orion*_2.



(a)

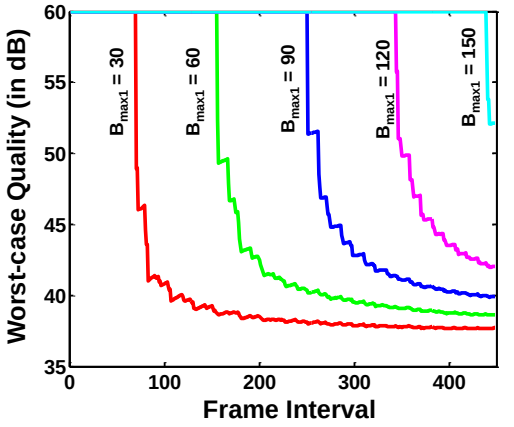


(b)

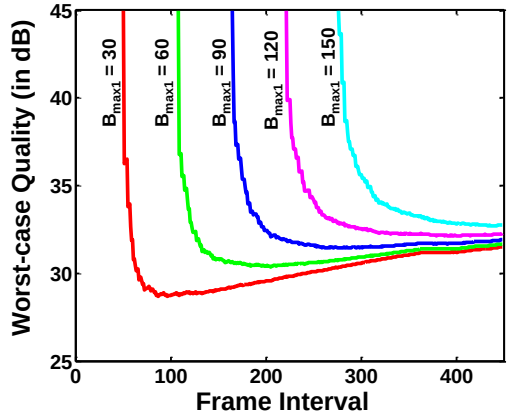


(c)

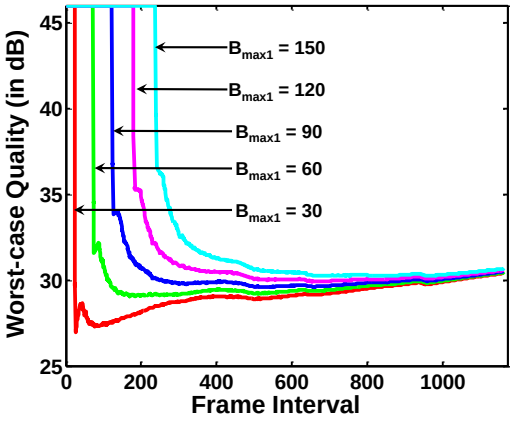
Figure 2.10: Comparison of analytical and simulation results of worst-case quality (q^u) for $B_{max1} = 30$ for three clips (a) *time*_080, (b) *susi*_080 and (c) *orion*_2.



(a)



(b)



(c)

Figure 2.11: Variation of worst case quality (q^u) with different buffer sizes for the clips (a) *time*_080, (b) *susi*_080 and (c) *orion*_2.

However, it is interesting to note here that, in all the three curves, the q^u value rises infinitely at some frame interval value. This is because below that frame interval, no frame drop is possible with the corresponding buffer size and therefore, the quality is maximum. As the first drop happens, the worst-case quality reduces and assumes a finite value. Another interesting aspect that this work highlights is shown clearly in Fig. 2.11. In the higher frame intervals, the worst-case quality values are very close to each other for different buffer sizes. This property could be exploited to reduce buffer dimensions for a small trade-off in q^u. For example in Fig. 2.11(a), if $40-45dB$ is an acceptable value for q^u, in a frame interval of 450, then $B_{max1} = 90$ can be chosen rather than $B_{max1} = 120$ in order to reduce the maximum buffer required. For an acceptable $q^u = 30-35dB$, it is seen in Fig. 2.11(b) that the least buffer size of 30 can be chosen for a frame interval of 450. Similar tradeoffs are evident in the third curve as well.

2.2.5.2 Second stage results

The second stage involves processor PE_2 and again two buffers. The frequency allocated to PE2 is 100 MHz. Again, we do not consider the I/P frame buffer, but analyze drop bounds for the B frame buffer only. Therefore, the resource parameter that we include for the analysis of the second stage is the buffer, labeled by B_{fin2}, which has size B_{max2}. The arrival curves at the input of B_{fin2} are $\alpha_{fin2} = [\alpha^u_{fin2}, \alpha^l_{fin2}]$ as computed in Section 2.2.2. Similarly, the service curves offered to the frames in B_{fin2} are $\beta_{fin2} = [\beta^u_{fin2}, \beta^l_{fin2}]$.

Effect of the second stage B frame buffer: The second stage B frame buffer size B_{max2} is set with three values of 40, 120 and 200 in the plots. In order to explore the second stage and finally the entire architecture, we apply the lemmas discussed earlier based on the output bounds obtained from the first stage. We present the results of this experiment for two clips, $time\_080$ and $orion\_2$. The results are presented in Fig.2.12 and Fig.2.13. In comparison to the first stage results, it is clearly seen that the worst-case quality bound increases in the second stage. The magnitude of increase depends on the capacity of second stage buffer. As the value of B_{max2} increases, the value of q^u increases as the drop bounds reduce. An interesting result observed in Fig.2.13 is that the quality bound does not vary much even when the buffer capacity is increased from $B_{max2} = 40$ to $B_{max2} = 200$.

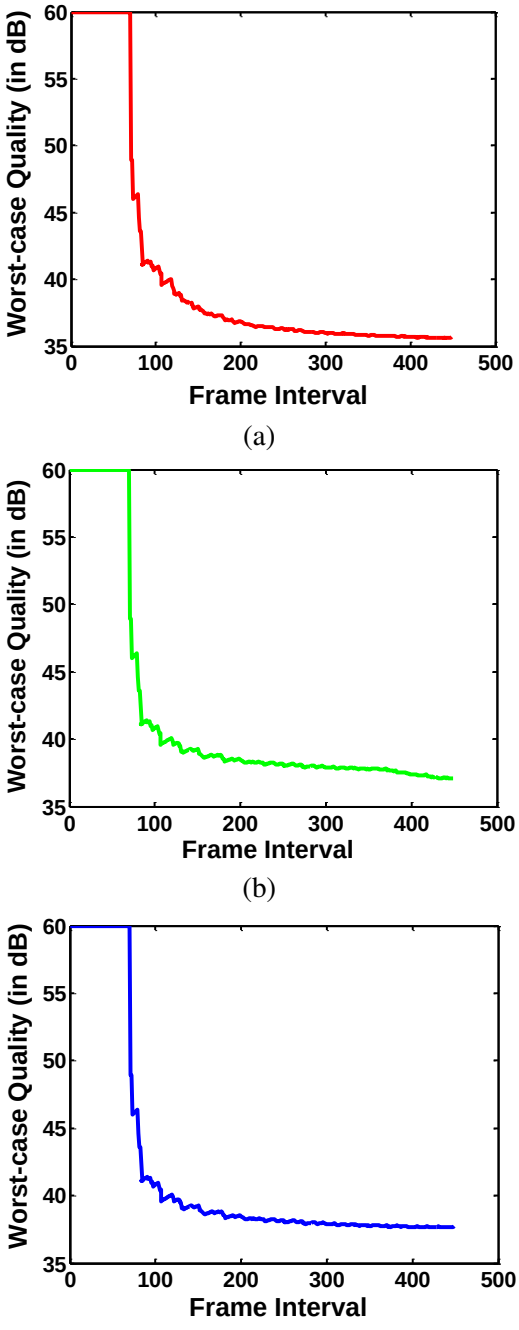


Figure 2.12: Worst case quality (q^u) with $B_{max1} = 30$ and (a) $B_{max2} = 40$, (b) $B_{max2} = 120$ and (c) $B_{max2} = 200$ for the clip *time*_080.

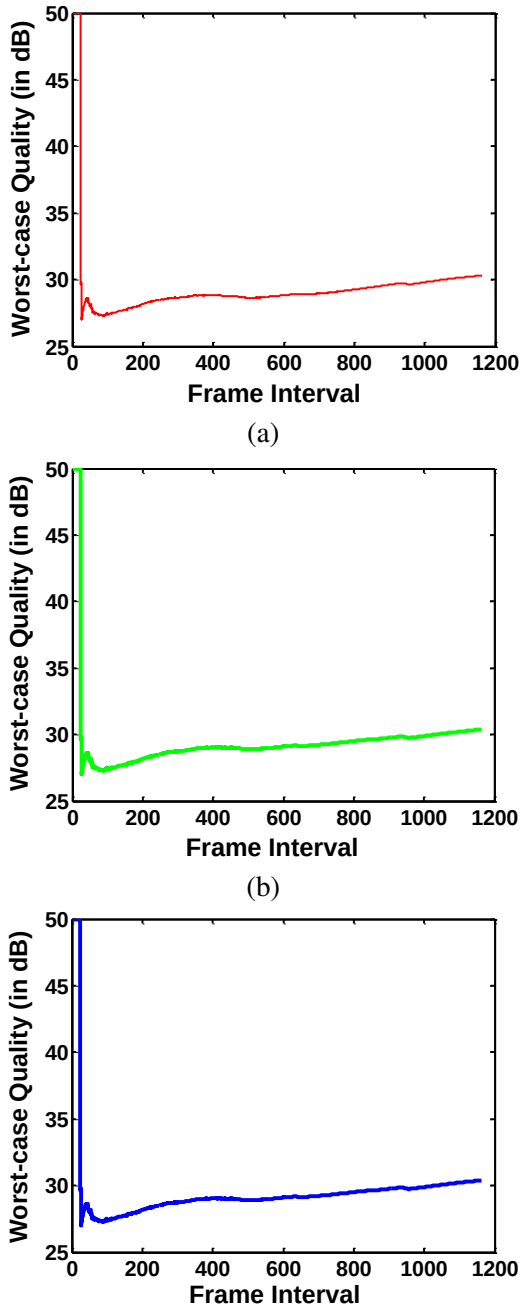


Figure 2.13: Worst case quality (q^u) with $B_{max1} = 30$ and (a) $B_{max2} = 40$, (b) $B_{max2} = 120$ and (c) $B_{max2} = 200$ for the clip *orion_2*.

Table 2.1: Buffer savings for the three video clips with quality variation

Buffer savings	clip	susi_080	time_080	orion_2
	PSNR (in dB)			
In Megabits	30	25.88	×	49
	35	3.53	5.09	6.16
	40	0.15	1.97	1.3
In percentage	30	28.6%	×	29.1%
	35	3.9%	39.4%	3.6%
	40	0.16%	15.5%	0.77%

2.2.5.3 Buffer savings

In this analysis, we highlight the significance of our mathematical framework. The final goal of the framework was to trade-off buffer size with quality. In the earlier results, we have seen that as the maximum buffer capacity is reduced, the quality reduces due to frame drops. However, if the resultant quality after frame drops is within tolerable limits, we can achieve significant savings in buffer. We present this result in Table 2.1. The savings shown consider drops only in the first stage. We find the buffer saving using $Bit^l(B_{nd}) - Bit^u(B_d)$. Here, B_{nd} is the buffer size (in frames) required for no drops and B_d is the buffer size (in frames) which allows drops within the tolerable quality shown in Table 2.1. Further, $Bit^u(F)$ and $Bit^l(F)$ are the maximum and minimum number of bits in F consecutive frames, respectively. It is known from multimedia literature that a PSNR value of 30-50 dB is an acceptable output quality. Hence, we vary the tolerable quality from 30-40 dB in steps of 5 dB. The $\times$ symbol against the clip $time\_080$ indicates that the quality never drops to 30 dB even if all the B frames are dropped. We can see from Table 2.1 that $time\_080$ shows more savings in terms of percentage when compared to the other two video clips. This is because $susi\_080$ and $orion\_2$ require a higher buffer size (in terms of Megabits) without any frame drops. Therefore, their savings (in percentage) is less.

2.3 Video Quality Driven Buffer Sizing via Prioritized Frame Drops

In this work, we first study the effect of existing quality aware frame dropping policy on the required buffer size such that a target PSNR value is achieved. Here, we study a frame dropping policy

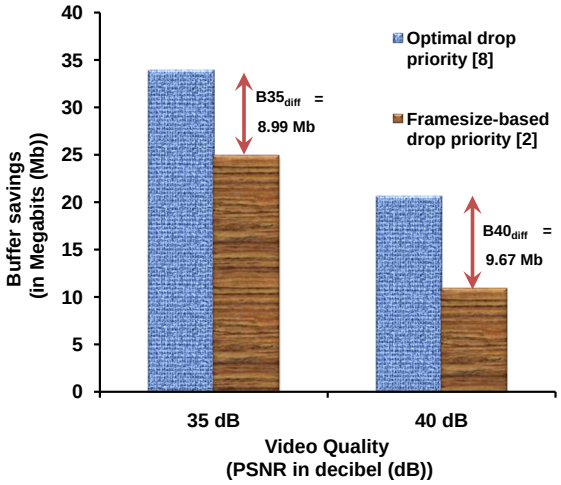


Figure 2.14: Evaluation of buffer savings using frame dropping policy from [2] versus optimal frame dropping policy from [3] for a benchmark MPEG-2 video *susi*_080 ([1]).

([2]) which prioritizes frames based on their frame sizes (in bits) i.e., frames are dropped in the increasing order of their frame sizes. This dropping policy is dynamic as it can be implemented online when the frames arrive on the MPSoC architecture. In contrast, though [3] provides an optimal frame dropping policy (as it drops frames in increasing order of distortion caused), it cannot be implemented online as it requires the complete decoding of the video stream. We conducted simulations to derive the buffer savings obtained by dropping frames using the frame drop policy from [2]. The results for the benchmark MPEG-2 video *susi*_080 ([1]) with target PSNR values 35 dB and 40 dB are presented in Fig. 2.14. The buffer savings obtained are compared with the optimal frame dropping policy. The difference in buffer savings are shown as $B35_{diff}$ and $B40_{diff}$ for PSNR values 35 dB and 40 dB, respectively. It is observed that these differences are considerably large values. The buffer savings using frame drop policy in [2] correspond to a drop of 181 and 81 frames respectively, for PSNR values of 35 and 40 dB. These figures for the optimal framedrop policy are 242 and 147 frames. In order to reduce the $B35_{diff}$ and $B40_{diff}$ values, we propose a simple prioritized frame dropping policy based on motion vectors, thereby enabling a buffer savings closer to the optimal. The motion vector based dropping policy can be implemented online. The on-chip frame buffer sizes, without frame drops, are of the order of 100 Mb ([54]). Therefore, in this context our savings are quite significant.

Our proposed motion vector based frame dropping policy paves the way for efficiently reducing

buffer size when the required output quality is known. However, determination of the minimum buffer size is a time consuming process because it requires many system simulations with various buffer sizes. Therefore, we further propose an efficient iterative strategy to derive the appropriate minimum buffer size for a given video stream. A system designer can perform this exercise with a representative set of video clips in the library (*covering all possible characteristics*). This is the current practice for evaluating architectures (which is similar to testing software for functional correctness using a representative test case suite). The buffer sizes obtained for individual clips can be used to decide the final buffer size requirement such that any encoded clip adhering to the bounds on input data bursts, exhibited by the library, will be decoded to achieve the required output quality.

2.3.1 Buffer Dimensioning Framework

The problem is formally defined here before getting into the components of the framework used.

2.3.1.1 Problem Formulation

In this work, we address the problem of buffer dimensioning for MPSoC platforms such that a target quality constraint is satisfied. Limited buffer sizes result in the loss of macroblocks/frames constituting an encoded video, which in turn leads to a drop in the quantitative quality level of the video decoded data received. Hence, given a library of video clips (that covers all types of video test cases), it is essential for a system designer to quickly find the minimum buffer size for the required output quantitative quality level, which here is measured in terms of PSNR.

The formal definition of the problem is stated as follows:

Given an exhaustive library of video clips V covering a wide variety of video characteristics, the operating frequency of the processing elements (PEs) f_{PE_i}, where $1 \leq i \leq N_{PE}$ (N_{PE} is the number of PEs in the MPSoC platform), workload values to execute each task in the video decoding application for all constituent blocks in the multimedia data, the task is to find

$$B_{min_j} = \max_{\forall v \in V}(\min B_j) \tag{2.5}$$

subject to

$$\min(psnr_{out}) \geq psnr_{req} \tag{2.6}$$

Here, $psnr_{out}$ is the PSNR value received at the output of the MPSoC platform, $psnr_{req}$ is the target PSNR value that needs to be achieved for the library of clips. Finally, B_{min_j} is the maximum of the minimum buffer sizes required for each video clip $v \in V$, at the input to the $j\text{-}th$ stage (which as shown in Eqn. 2.5 is the minimum of all possible buffer values B_j) in order to satisfy Eqn. 2.6. To efficiently solve the above problem, we need a proper quality-aware frame dropping function $FD()$ and a fast iterative strategy to derive the minimum buffer size B_j. Hence, we now highlight the importance of a quality-aware frame dropping policy in deriving a minimal buffer size satisfying a target PSNR value.

2.3.1.2 Quality-Aware Frame Dropping

Lemma 2.3.1 *Given the operating frequency of the PEs, the task workload values and the video library, Eqn. 2.5 can be strictly satisfied only if the frame dropping strategy is aware of the relative importance of the frames (in terms of distortion introduced) in the video stream.*

Proof Let us consider that $FD1()$ is a random frame dropping function and $FD2()$ is a frame dropping function aware of the importance of the frames and its contribution to PSNR, if dropped. Given the target PSNR value $psnr_{req}$ and the maximum buffer size Buf causing no frame drops for a video clip, let the maximum number of frames dropped by $FD1()$ before it achieves the PSNR output value of $psnr_{out}$ satisfying Eqn. 2.6 be $n1$ frames. Let the number of frames dropped by $FD2()$ for the same case be $n2$. As $FD2()$ is aware of the relative importance of the frames, it will drop frames in the increasing order of how they reduce the PSNR output value i.e. frames which reduce the PSNR output value least will be dropped first, while $FD1()$ does not do this strictly. Hence, it is easy to conclude that $n2 \geq n1$. If $Buf1$ is the minimum buffer size estimated (as in Eqn. 2.5) with $FD1()$, $Buf2$ is the minimum buffer size estimated with $FD2()$ and $FSIZE$ is the frame size, then the following conditions hold: $Buf1 \leq Buf - n1 \times FSIZE$, $Buf2 \leq Buf - n2 \times FSIZE$ and hence $Buf1 \geq Buf2$. This proves that a good estimate of minimum buffer size (in macroblocks) satisfying the PSNR requirement can be achieved only with a good frame dropping strategy. However, if $n2 >> n1$ (due to quality aware frame dropping of $FD2()$), we see that $Buf1 > Buf2$ (in bits).

A state-of-the-art frame dropping method [2] discussed in literature prioritizes frames to be dropped based on the frame types namely I-type, P-type or B-type (as in the MPEG-2/MPEG-4 decoder context) with frame size based prioritization within frame types. This strategy works well in comparison to a random frame dropping strategy, but does not take the aspect of *motion across frames* into consideration. Hence, the PSNR values obtained with the frame size based dropping strategy does not work particularly well for motion videos when a dropped frame is replaced with the previous processed frame which has considerable movement. Therefore, we adopt a frame dropping strategy whereby motion-vector based frame dropping is employed. The motion-vectors can be easily obtained by parsing the video stream. The output of this stage is the prioritized order in which frames have to be dropped for a video. The gaps created in buffer with such dropping can be consolidated with minimal logic.

2.3.1.3 Determination of B_{min_j}

In the second stage of the framework, we estimate the minimum buffer size required to achieve a prespecified PSNR value for a library of video clips using the task workloads and frame drop priorities obtained earlier. This stage employs a fast iterative process in order to determine the minimum buffer size. The main idea here is to maximize the frame drops subject to the condition that the prespecified PSNR value is met. As PSNR is dependent on the MSE of the dropped frames when replaced by the concealment frames, the condition of Eqn. 2.6 is translated from the PSNR domain into the MSE domain. As MSE is inversely proportional to PSNR, the condition is changed to $\max(mse_{out}) \leq mse_{req}$. Let this be the **MSE satisfaction criterion**. Here, mse_{out} is the MSE value received at the output. The goal of the entire framework is to achieve a mse_{out} value equal to the maximum possible MSE value less than or equal to the target MSE value (mse_{req}). MSE is preferred over PSNR as it is an easier quantity to work with. However, the problem of finding the minimum buffer size for a library of video clips which satisfies the above mentioned conditions is time consuming if all the possible buffer sizes are tested. We use a faster iterative approach to find B_{min_j}, which will be discussed in detail later.

2.3.2 Quality-Aware Frame Dropping

In order to improve the quality awareness in assigning priorities to frames during frame drops, we propose a motion-vector (MV) based prioritization of frames. This is a fast method because the motion vectors of each frame can be extracted rapidly from the encoded bitstream that is received. Moreover, as it takes the motion information into consideration, it takes into account the quality degradation experienced when a frame is dropped. One of the advantages of this prioritization method and the one discussed in [2] are that they are easy to compute. Motion vectors and frame sizes can be extracted from the bitstream quickly when compared to the MSE computation of dropped frames with all possible concealment frames (frames that replace dropped frames).

1. There are two motion vectors in the MPEG-2/MPEG-4 encoded bitstream - one for the upper half of the 16×16 macroblock and the other for the lower half. Each motion vector has a forward and a backward component. Each of the above four components have a vertical and a horizontal part. All these quantities are at the MB granularity. We now obtain a consolidated value combining the horizontal and vertical parts for each component. Let the MVs be denoted by $mv_{l,m,n}$, where l represents the upper or lower half (takes on the values $0/1$ respectively), m represents the forward/backward component (takes on the values $0/1$ respectively) and n denotes the vertical/horizontal parts (takes on the values $0/1$ respectively). The four consolidated values (per MB) combining the horizontal and vertical parts are

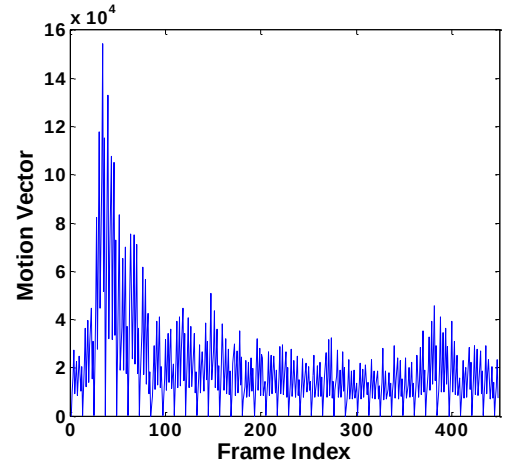
$$\begin{aligned} mv_{0,0/1} &= \sqrt{(mv_{0,0/1,0})^2 + (mv_{0,0/1,1})^2} \\ mv_{1,0/1} &= \sqrt{(mv_{1,0/1,0})^2 + (mv_{1,0/1,1})^2} \end{aligned} \tag{2.7}$$

   The consolidated motion vector components for each frame is computed as summation of MB level motion vector components as shown below

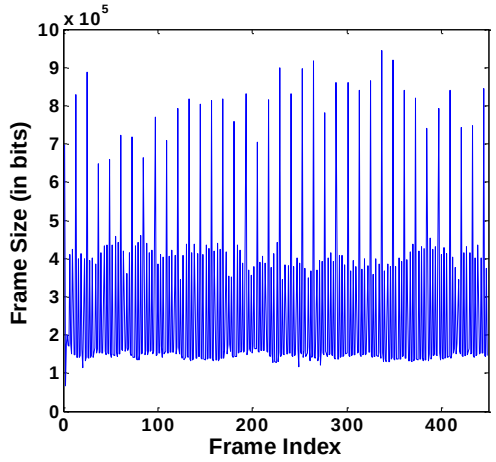
$$MV_{l,m} = \sum_{mbno=1}^{FSIZE} mv_{l,m}, l \in \{0,1\}, m \in \{0,1\} \tag{2.8}$$

   where mbno is the MB number.

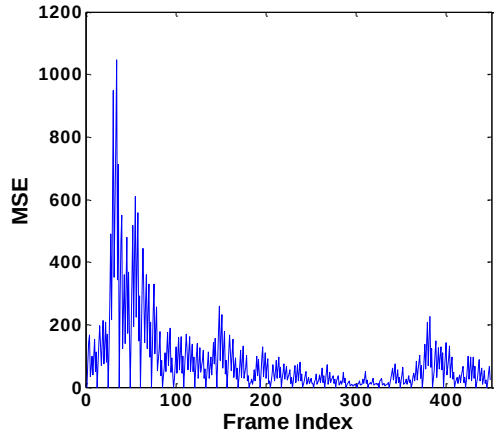
2. After obtaining the motion vectors per frame given by Eqn. 2.8, we find the priority of dropping the frames. As in any efficient frame dropping strategy, we drop B-frames first followed



(a)



(b)



(c)

Figure 2.15: (a) Motion Vector vs Frame Index, (b) Framesize vs Frame Index, and (c) MSE vs Frame Index for a motion video *susi*_080.

by P-frames and finally I-frames in the case of a buffer overflow. We implement this by creating three separate lists of priorities, one each for B-frame, P-frame and I-frame denoted by *b_priority*, *p_priority* and *i_priority*. These lists contain the indexes of the frames arranged in the order of increasing motion vector values.

3. *b_priority* holds the B-frame priority list. There are two types of B-frames - odd numbered B-frames and the even numbered B-frames. The even numbered B-frames are always dropped first when possible as they can then be replaced by the immediate odd numbered B-frame before it in the temporal sequence. Once all the possible even numbered B-frames are dropped, the odd numbered B-frames are discarded. Among the odd numbered B-frames, they are prioritized based on the increasing order of the sum of forward components i.e. the comparison metric is $MV_{0,0} + MV_{1,0}$. The comparison metric for the priority computation of even numbered B-frames is $MV_{0,0} + MV_{1,0} + MV_{0,1}^{prev} + MV_{1,1}^{prev}$. If the comparison metric of a frame is high, it will be dropped later. MV^{prev} are the motion vector components for the previous frame. For the even numbered B-frames, we need to consider the backward motion components of the previous odd numbered B-frame.

4. *p_priority* holds the P-frame priority list. Here it is desirable that all the P-frames in a GOP are kept together in the list, the last P-frame to be dropped being the one closest to the I-frame as it acts as a reference frame for a lot of following frames. Among the group of P-frames, the order is decided by the comparison metric $\sum\{MV_{0,0} + MV_{1,0}\}$, where the summation is over the number of P-frames in the GOP.

5. *i_priority* holds the I-frame priority list. It orders the I-frames based on the comparison metric used for P-frames.

We extracted the MSE information of every frame with its reference frame in some motion and still videos along with their motion vectors with respect to the reference frame and the frame sizes. The plots are shown in Fig. 2.15. From the plot for the motion video *susi*_080, we can see that the motion vector emulates the MSE behavior much better than framesize. This behavior has also been observed in MPEG-4 videos. Therefore, we conclude that it is desirable to use a quality-aware frame dropping mechanism with MV-based prioritization of frames.

2.3.3 Minimum Buffer Size Estimation

In this section, we propose an iterative procedure to estimate the minimum buffer size required at each stage of the MPSoC architecture so that a library of video clips satisfies the required target PSNR value. The input to this stage from the previous stage is an order in which frames should be dropped. In order to obtain the minimum buffer size, the best strategy is to drop the maximum number of frames but still satisfy the *MSE satisfaction criterion*. A straightforward approach of using brute force method to check all the buffer sizes is not desirable as it is very time consuming considering the number of clips in the library and the range of buffer sizes used. Therefore, we use an iterative mechanism to select the minimum buffer size.

In order to obtain the minimum overall buffer size considering all stages, the best strategy is to drop all the required frames in the first stage. This will not only reduce the buffer size B_1, but will also reduce the number of frames that stage 2 and buffer size B_2 have to handle. In order to start the iterative process of finding the minimum buffer size, we need to find the MSE values for a limited number of frames which are dropped first according to the drop order until the accumulated MSE just exceeds the target MSE value mse_{req}. The PSNR value of a video sequence with frame drops is expressed as in Equation 1.3. The target MSE value can be expressed in terms of the target PSNR value $psnr_{req}$ as $mse_{req} = \frac{(255 \times 255 \times N_{tot})}{10^{psnr_{req}/10}}$.

The minimum buffer size for the first stage can be calculated using Algorithm. 3. The terms used are:

mse_eval - Cumulative MSE value, frm_ind - Index of the frame dropped, *mbin_no* - Arriving MB number, *N_prev* - Value of the number of frames dropped in the previous iteration, MAX_BUFFER - Buffer size required if there are no frame drops, Buf_1 - Buffer size at stage 1 of the architecture, N_{new} - Number of frames to be dropped (in Step 2), which is computed every iteration until we achieve the maximum value which does not exceed *mse_eval* and finally *start* and *end* variables are used to speed up the search of the possible number of frame drops. In this algorithm, there are two steps.

Step 1 (Lines 1-9): Here, we find the maximum number of frame drops (N_{drop}) possible given the priority of frame drops and the mse_{req} value. It is assigned as $i-2$ because $mse\_eval > mse_{req}$ for $(i-1)$-th frame drop. We only consider the B-frame drops here because this itself leads to a large

Algorithm 3 Computing minimum buffer size for a clip in first stage

Input: $mse\_stat()$ - mse values for a threshold number of frames (B_{thr});
Output: B_1^{min} for a given f_{PE_1} and frame discard algorithm

1: $N \leftarrow 0$, $mse\_eval \leftarrow 0$ and $frm\_ind \leftarrow 0$;
2: **for** $i = 1$ to B_{thr} **do**
3: **if** $mse\_eval > mse_{req}$ **then**
4: $N_{drop} = i - 2$
5: break
6: **else**
7: $frm\_ind = b\_priority \rightarrow next$
8: $mse\_eval = mse\_eval + mse\_stat(frm\_ind)$
9: **end if**
10: **end for**
11: $N\_prev \leftarrow 0$, $N\_int \leftarrow 0$, $mse\_eval \leftarrow 0$, $Buf_1 \leftarrow 0$, $n\_drop \leftarrow 0$, $N_{new} \leftarrow 0$, $start \leftarrow 0$, $end \leftarrow N_{drop}$
12: Reset list next pointer to start of list
13: **repeat**
14: $N\_prev = N_{new}$, $N_{new} = N_{drop} - ((end - start)/2)$,
15: $N\_int = 0$, $mse\_eval \leftarrow 0$, $Buf_1 \leftarrow 0$, $n\_drop \leftarrow 0$
16: Reset list next pointer to start of list
17: **while** $mbin\_no \leq N_{tot} \times FSIZE$ **do**
18: **if** $Buf_1 = MAX\_BUFFER - N_{new} \times FSIZE$ **then**
19: $frm\_ind = b\_priority \rightarrow next$
20: $mse\_eval = mse\_eval + mse\_stat(frm\_ind)$
21: $Buf_1 = Buf_1 - FSIZE$
22: $n\_drop = n\_drop + 1$
23: **if** $mse\_eval > mse_{req}$ **then**
24: $N\_int = n\_drop - 1$
25: **else**
26: **if** $mse\_eval = mse_{req}$ **then**
27: $N\_int = n\_drop$
28: **end if**
29: **end if**
30: **else**
31: $Buf_1 = Buf_1 + 1$
32: **end if**
33: **end while**
34: **if** $N\_int = 0$ **then**
35: $start = N_{new}$, $end = end$
36: **else**
37: $start = start$, $end = N\_int$
38: **end if**
39: **until** $(start = end)||(start = end - 1)$
40: $B_{1,v}^{min} = MAX\_BUFFER - N\_prev \times FSIZE, \forall v \in V$

PSNR drop covering the PSNR ranges we are exploring and hence the B_{thr} (line 3) is equivalent to the number of B-frames.

Step 2 (Lines 10-39); Once N_{drop} is computed, we can compute $B_{1,v}^{min}$ as shown in line 39 of Algorithm 3. The minimum buffer size $B_{1,v}^{min}$ will be due to frame drops less than or equal to this maximum number N_{drop} computed in (line 4) Step 1. It can be less than N_{drop} if certain frames, with higher priority to be dropped, are already processed and sent to the next stage. This target number of buffer drops will be computed iteratively. Once the minimum buffer size is computed for all the clips, the resultant buffer size of stage 1 can be computed as $B_1^{min} = \max_{\forall v \in V}(B_{1,v}^{min})$. The iterative procedure can be run using $O(log(N_{drop}))$ iterations, where $N_{drop} << N_{tot}$ and N_{drop} is independent of the length of the clip, but dependent on nature of the clip and required PSNR value. In our example, the maximum number of iterations for $PSNR = 40dB$ was 7, which is low.

2.3.4 Experimental Results

In this section, we conduct two sets of experiments to validate our proposals earlier. The first experiment involves verifying the effect of MV-based frame dropping on buffer size reduction. The second result gives us the minimum buffer size values for various PSNR values at the first stage of the MPSoC architecture for the library of video clips we have used here. We use 11 video clips from [1] - 5 still clips and 6 motion clips. It has been observed that the still clips require lesser buffer size. Hence, we show the buffer sizes of motion clips only (as they decide minimum buffer size required) in both experiments. The MPEG-2 decoder source code used was from [65]. In a shared buffer scenario with multiple cores, the reduction of required buffer size helps in allocating the free buffer resources to other cores requiring more space.

2.3.4.1 Evaluation of MV-based frame dropping

It is clear from Fig. 2.16 and Fig. 2.17 that MV-based prioritized frame drops help in improving buffer savings when compared to the framesize based frame drops. As shown in Fig. 2.16, we achieve 22.89% and 55% more buffer savings over framesize based dropping for PSNR values of 35 dB and 40 dB respectively in *susi*_080. We also observe from Fig. 2.17 that the additional savings (over framesize based drop from [2]) in *tens*_080 is 2.87 times and 5.29 times for PSNR values of

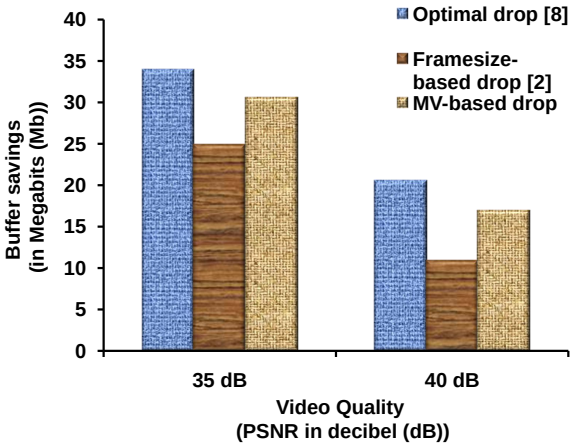


Figure 2.16: Comparison of buffer savings for *susi*_080

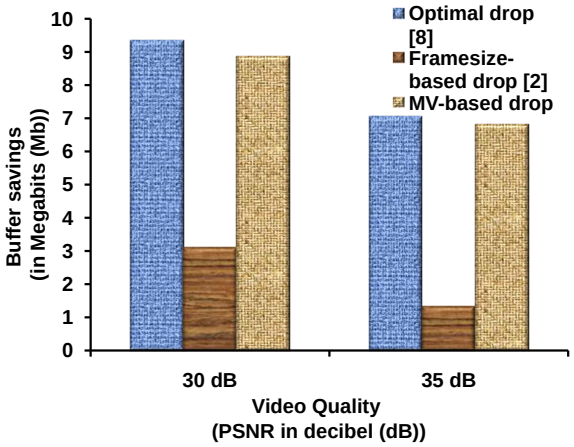


Figure 2.17: Comparison of buffer savings for *tens*_080

30 dB and 35 dB respectively.

2.3.4.2 Minimum Buffer Size Estimation

We conducted experiments to find the minimum buffer size required at the first stage of the MPSoC architecture shown in Fig. 2.2 using the video clips from [1]. The prespecified PSNR values for which we estimated the minimum buffer values are shown in Table. 2.2. Here we show the buffer sizes for 3 motion videos which required highest buffer sizes. For the entire library, with a target PSNR value of 30 dB, we achieved a buffer savings of 3.9 Mbits (Maximum buffer size for the entire library without drops - Maximum buffer size for PSNR = 30 dB (required for *flwr*_080)).

Table 2.2: Minimum buffer size (in Megabits) for various prespecified PSNR values with $f_{PE_1} = 25MHz$

clip PSNR (in dB)	susi_080	cact_080	flwr_080
30	×	87.97	95.43
35	68.75	92.77	97.87
40	82.43	95.81	98.75

The × symbol against the clip *susi*_080 indicates that the quality never drops to 30 dB even if all the B frames are dropped. Hence, the iterative process can be immediately terminated because the resultant buffer size will be lesser compared to other video clips.

2.4 Summary

In this chapter, we first study the effects of frame drops in a multiprocessor system-on-chip platform running video decoder applications. Towards this objective, we propose a novel mathematical model to compute the worst-case drop bound in a MPSoC architecture with finite buffers. This analytical model helps in exploring the buffer-quality design space by analyzing the worst-case quality when frames are dropped. One important aspect of this work is that we can explore the buffer-quality design space by trading off a significant buffer area for a tolerable loss in quality.

Subsequently we propose a quality-aware framed dropping scheme based on motion vectors that reduces the buffer size required for a prespecified quality constraint. Further, we also propose a fast iterative strategy to derive the minimum buffer size required for a target quality output.

Chapter 3

Quality-Driven Service Determination

Simultaneous viewing of multiple video streams has recently become a very common feature in televisions (TVs) and personal computers (PCs). These multiple video streams are either displayed adjacent to each other (as in the display of programs from multiple channels simultaneously on a TV or PC) or as a Picture-in-Picture (PiP) where one video stream is displayed on the full screen while the other is displayed in an inset window. In order to process these multiple streams with maximum quality, adequate number of processor cycles need to be provided. There are several works in literature that analyze multimedia stream processing (e.g., [53], [61] and [54]) with the objective of achieving maximum quality, i.e., without frame drops. Embedded devices with respource constraints (e.g., mobile devices) are currently providing PiP application [66] to allow users to watch two videos simultaneously. Here, in contrast to TVs and PCs, these devices have acute resource constraints, which make it difficult to process multiple video streams with maximum quality when other applications run simultaneously.

In this work, we propose a formal framework to design an appropriate scheduler that services the multiple incoming streams in a PiP application such that certain quality constraints are satisfied. This framework will be useful for scheduling multiple multimedia streams in an embedded device, where some frame drops can be tolerated without significant deterioration in the quality of the streams. In other words, it will also be possible to quickly determine if the available processor time will be sufficient to schedule the streams, while adhering to their required quality constraints. Although we illustrate our technique for a PiP application, the work is in general relevant for schedul-

ing multiple multimedia streams. These might be two video streams or video and audio streams or video + graphics/games streams.

3.1 Processor Service Determination Framework

This section presents an overview of our mathematical framework to derive the appropriate scheduler parameters such that the multiple incoming multimedia streams adhere to their individual target quality constraints (in terms of PSNR). Our framework uses the arrival and service curve concepts from Network Calculus to model the data arrival and service given by the resources, respectively. These arrival and service curves efficiently capture the variability in the data arrival and the processing required for the arrived data. We now describe the platform in detail before getting into our framework.

Platform Description: In this work, we analytically derive the scheduler parameters necessary to schedule multiple incoming streams on a resource constrained MPSoC architecture as shown in Fig. 3.1 with acceptable quality deterioration. The architecture consists of two processing elements (PEs) denoted by PE_1 and PE_2. Each PE services two individual incoming streams $a_1(t)$ and $a_2(t)$, which are cumulative functions that denote the total number of stream objects (such as macroblocks or frames in a video stream) that arrive over the time interval $[0,t]$. Moreover, here we assume that the PEs execute tasks from a video decoder application like MPEG-2. The buffers used by stream $a_1(t)$ have sizes B_1, B_3 and B_{a1} (in number of frames) at the input, intermediate and playout stages of the setup, respectively. Similarly, the buffers used by stream $a_2(t)$ have sizes B_2, B_4 and B_{a2} (in number of frames) at the input, intermediate and playout stages of the setup, respectively. $y_1(t)$ and $y_2(t)$ are the processed stream outputs from PE_1 corresponding to inputs $a_1(t)$ and $a_2(t)$, respectively. Similarly, $z_1(t)$ and $z_2(t)$ are the processed stream outputs from PE_2 corresponding to inputs $y_1(t)$ and $y_2(t)$, respectively. In our setup, $y_1(t)$, $y_2(t)$, $z_1(t)$ and $z_2(t)$ are also cumulative functions. The playout consumption functions for the two streams are denoted as $C_1(t)$ and $C_2(t)$, respectively, which are also cumulative functions. We use the same definitions for the terms frame interval, arrival curve and service curve as given by Definitions 3, 1 and 2 respectively.

In our setup, $\alpha_1 = [\alpha_1^u, \alpha_1^l]$ and $\alpha_2 = [\alpha_2^u, \alpha_2^l]$ are the arrival curves for the two streams at the input

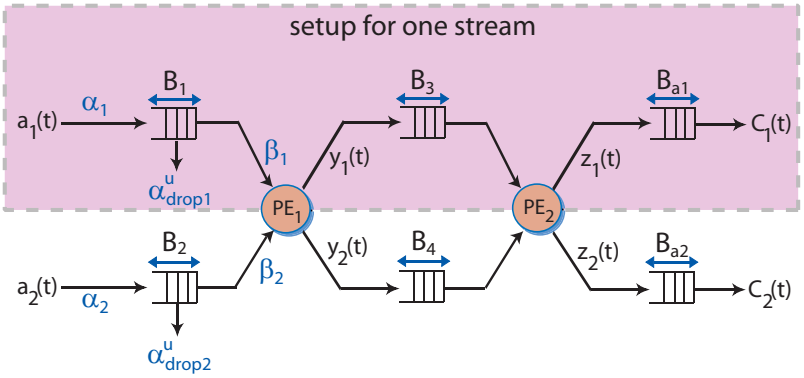


Figure 3.1: MPSoC platform setup for a PiP-like application with frame drops showing two streams with separate buffers, but sharing processing resources.

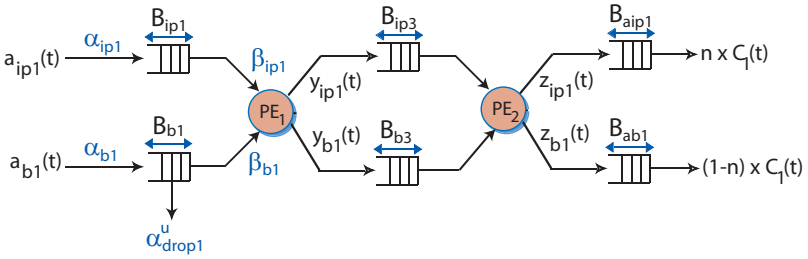


Figure 3.2: System model for the shaded portion representing data path for stream $a_1(t)$ in Fig. 3.1.

stage of the architecture as shown in Fig. 3.1, while $\beta_1 = [\beta_1^u, \beta_1^l]$ and $\beta_2 = [\beta_2^u, \beta_2^l]$ are the service curves offered to the two streams by PE_1.

Problem Definition: *Given the arrival curves α_1 and α_2, corresponding to the two video streams ($a_1(t)$ and $a_2(t)$, respectively) in a PiP-like application that are required to be decoded on a resource-constrained MPSoC platform, the sizes of input buffers (B_1 and B_2), the sizes of intermediate buffers (B_3 and B_4), the sizes of playout buffers (B_{a1} and B_{a2}) and the playout consumption functions ($C_1(t)$ and $C_2(t)$), we analytically derive the scheduler parameters of PE_1, such that the two video streams individually satisfy their respective target quality constraints, Q_1^{target} and Q_2^{target} (in PSNR).*

The first stage encounters frame drops as shown in Fig. 3.1, where the number of frame drops in any time interval Δ is bounded by $\alpha^u_{drop}(\Delta)$ called *Drop Bound*. The drop bounds for the two incoming streams are denoted by $\alpha^u_{drop1}(\Delta)$ and $\alpha^u_{drop2}(\Delta)$. In order to analyze the MPSoC platform shown in Fig. 3.1, where there are frame drops, we cannot directly follow the analysis method presented

in earlier works (e.g., [53]), which ensure that no buffer overflows and playout buffer does not underflow. In our analysis with frame drops (or buffer overflows), we use the system model shown in Fig. 3.2 for the shaded portion in Fig. 3.1. The analysis of the other half is similar to that of the shaded portion.

The incoming video stream is divided into two parts - a significant part denoted by $a_{ip1}(t)$ (no frame drops) and a less significant part denoted by $a_{b1}(t)$ (with frame drops) in Fig. 3.2. These two parts combine to form the original stream $a_1(t)$, i.e., $a_1(t) = a_{ip1}(t) + a_{b1}(t), \forall t \geq 0$. The partitioned arrival curves corresponding to the two parts are shown as $\alpha_{ip1} = [\alpha^u_{ip1}, \alpha^l_{ip1}]$ and $\alpha_{b1} = [\alpha^u_{b1}, \alpha^l_{b1}]$, respectively, while the partitioned service curves are $\beta_{ip1} = [\beta^u_{ip1}, \beta^l_{ip1}]$ and $\beta_{b1} = [\beta^u_{b1}, \beta^l_{b1}]$, respectively. As shown in Fig. 3.2, the buffer sizes for the two parts at the three stages of the architecture are $[B_{ip1}, B_{ip3}, B_{aip1}]$ and $[B_{b1}, B_{b3}, B_{ab1}]$, respectively, such that $B_1 = B_{ip1} + B_{b1}$, $B_3 = B_{ip3} + B_{b3}$ and $B_{a1} = B_{aip1} + B_{ab1}$. The partitioned outputs from PE_1 are $y_{ip1}(t)$ and $y_{b1}(t)$, while the partitioned outputs from PE_2 are $z_{ip1}(t)$ and $z_{b1}(t)$ as shown in Fig. 3.2.

Partitioned Consumption: If the display rate or consumption function at the playout stage for stream $a_1(t)$ is $C_1(t)$, then the consumption functions for the partitioned streams are $n \times C_1(t)$ and $(1-n) \times C_1(t)$ (as shown in Fig. 3.2), where n is the fraction corresponding to significant part. For example, for a frame sequence of IBBPBBPBBPBBI..., $n = \frac{1}{3}$. If the display rate is 30 frames/second, then the partitioned consumption functions are 10 frames/second and 20 frames/second for I/P frames (significant part) and B frames (less significant part), respectively.

3.2 Computing Quality-Driven Service Curves

The main objective is to find the bounds for the original service curves $\beta_1 = [\beta^u_1, \beta^l_1]$, which is quality driven. In order to compute these bounds, we first need to find the bounds on the partitioned service curves $\beta_{ip1} = [\beta^u_{ip1}, \beta^l_{ip1}]$ (significant part) and $\beta_{b1} = [\beta^u_{b1}, \beta^l_{b1}]$ (less significant part).

The computation of service curve for the significant part follows the method presented in [53]. The

β^l_{ip1} bound is therefore obtained by ensuring that buffer B_{ip1} does not overflow, i.e.,

$$\begin{aligned} & a_{ip1}(t) - B_{ip1} \leq y_{ip1}(t), \forall t \geq 0 \\ \Leftrightarrow\ & \beta^l_{ip1}(t) \otimes a_{ip1}(t) \geq a_{ip1}(t) - B_{ip1}, \forall t \geq 0 \\ \Leftrightarrow\ & \beta^l_{ip1}(t) \geq (a_{ip1}(t) - B_{ip1}) \oslash a_{ip1}(t), \forall t \geq 0. \end{aligned} \tag{3.1}$$

We can compute the β^u_{ip1} bound by ensuring that buffer B_{ip3} does not overflow, i.e.,

$$y_{ip1}(t) \leq z_{ip1}(t) + B_{ip3}, \forall t \geq 0 \tag{3.2}$$

In order to ensure that buffer B_{aip1} does not underflow, we have

$$z_{ip1}(t) \geq n \times C_1(t), \forall t \geq 0 \tag{3.3}$$

From Eq. 3.2 and Eq. 3.3, in order to strictly ensure that B_{ip3} does not overflow, we can deduce that

$$\begin{aligned} & y_{ip1}(t) \leq n \times C_1(t) + B_{ip3}, \forall t \geq 0 \\ \Leftrightarrow\ & \beta^u_{ip1}(t) \otimes a_{ip1}(t) \leq n \times C_1(t) + B_{ip3}, \forall t \geq 0 \\ \Leftrightarrow\ & \beta^u_{ip1}(t) \leq (n \times C_1(t) + B_{ip3}) \oslash a_{ip1}(t), \forall t \geq 0. \end{aligned} \tag{3.4}$$

Before we compute $\beta_{b1} = [\beta^u_{b1}, \beta^l_{b1}]$, let us define some quantities that will be used hereafter in this chapter.

Definition 5 ***Worst-case quality surface*** **(Q^u).** *For any frame interval F, the worst-case quality surface $Q^u(f,F)$, for all $0 \leq f \leq F_b$, is the worst-case quality of the video if f frames are dropped in any window of F consecutive frames. Here, F_b is the total number of less significant frames that can be dropped and $F_b < F$.*

All dropped frames are replaced by immediately preceding and successfully processed frames called *concealment frames*. The amount of quality loss depends on the MSE between the dropped and concealment frames.

Definition 6 ***Frame interval based time bound*** **($\delta^u(F)$).** *Given the original arrival curve before partitioning α_1, the upper bound on time required for arrival of F frames is given by*

$$\delta^u(F) = \min\{\Delta \geq 0 \mid \alpha^l_1(\Delta) \geq F\}$$

Lemma 3.2.1 *Given the target quality constraint Q_1^{target}, the worst-case quality surface $Q^u(f,F)$ and the frame interval based time bound $\delta^u(F)$, the upper bound on number of frames that can be dropped in any time interval Δ is given by*

$$f^u(\Delta) = f_{max}(F)$$

where $f_{max}(F)$ is the maximum number of frames that can be dropped in a frame interval F such that $Q^u(f_{max}(F),F) \geq Q_1^{target}$ and $\delta^u(F) \leq \Delta < \delta^u(F+1)$.

Proof This lemma can be proved by considering two instances of time intervals.
Case I: Let us first consider the straightforward case with time intervals given by $\Delta = \delta^u(F)$. These are the lowest time interval values where $\alpha_{b1}^l \geq F$. If $f_{max}(F)$ is the maximum possible number of frames that can be dropped in a frame interval F such that the quality constraint is satisfied, i.e., $Q^u(f_{max}(F),F) \geq Q_1^{target}$, then for $\Delta = \delta^u(F)$, we have $f^u(\Delta) = f_{max}(F)$.
Case II: Now let us consider the time intervals given by $\delta^u(F) < \Delta < \delta^u(F+1)$. These are the time intervals when $\alpha_1^l(\Delta) > F$ and $\alpha_1^l(\Delta) < F+1$. In these time intervals, the maximum number of frames that can be dropped should be at most $f_{max}(F)$ so that the quality constraint is satisfied. If the number of frame drops exceeds $f_{max}(F)$, then the quality constraint is violated because $\alpha_1^l(\Delta) < F+1$ and therefore any frame drop $f_d > f_{max}(F)$ will result in $Q^u(f_d,F) < Q_1^{target}$. Then, for $\delta^u(F) < \Delta < \delta^u(F+1)$, we have $f^u(\Delta) = f_{max}(F)$.
Hence, it is proved that $f^u(\Delta) = f_{max}(F), \forall \Delta \geq 0$.

Lemma 3.2.2 *Suppose $\alpha_{b1} = (\alpha_{b1}^u, \alpha_{b1}^l)$ are the arrival curves of the less significant stream as shown in Fig. 3.2, $\beta_{b1} = (\beta_{b1}^u, \beta_{b1}^l)$ are the service curves for the less significant stream on PE_1, and B is the size of the input buffer. Then, the number of input frames that can be dropped over any interval of length $\Delta \geq 0$ is upper bounded by $\alpha_{drop1}^u(\Delta)$, defined by*

$$\alpha_{drop1}^u = (\alpha_{b1}^u - \beta_{v1}^l) \mathbin{\overline{\otimes}} 0$$

where $\beta_{v1}^l \stackrel{\text{def}}{=} (\alpha_{b1}^l \otimes \beta_{b1}^l + B_{b1})^* \otimes \alpha_{b1}^u$.

Proof It is proved as in Lemma 2.2.3.

Lemma 3.2.3 *Define α_{b1}, β_{b1} and α^u_{drop1} as in Lemma 3.2.2. Also define f^u as in Lemma 3.2.1. Then, for any given time interval $\Delta \geq 0$, in order to satisfy the quality constraint, the lower service curve (β^l_{b1}) is given by*

$$\beta^l_{b1}(\Delta) \geq ((((((\alpha^u_{b1} - f^u) \mathbin{\overline{\otimes}} 0) \oslash \alpha^u_{b1}) - B_{b1}) \mathbin{\overline{\otimes}} 0) \oslash \alpha^l_{b1})(\Delta)$$

Proof Let us start from the expression for drop bound given in Lemma 3.2.2. In order to satisfy the quality constraint, the following relation needs to be maintained:

$$\begin{aligned}
& \alpha^u_{drop1}(\Delta) \leq f^u(\Delta) \\
\Leftrightarrow\ & ((\alpha^u_{b1} - \beta^l_{v1}) \mathbin{\overline{\otimes}} 0)(\Delta) \leq f^u(\Delta) \\
\Leftrightarrow\ & ((\alpha^u_{b1} - (\alpha^l_{b1} \otimes \beta^l_{b1} + B_{b1})^* \otimes \alpha^u_{b1}) \mathbin{\overline{\otimes}} 0)(\Delta) \leq f^u(\Delta) \\
\Leftrightarrow\ & (\alpha^u_{b1} - (\alpha^l_{b1} \otimes \beta^l_{b1} + B_{b1}) \otimes \alpha^u_{b1})(\Delta) \leq f^u(\Delta) \\
& (As\ g \mathbin{\overline{\otimes}} 0 \leq h \Rightarrow g \leq h) \\
\Leftrightarrow\ & ((\alpha^u_{b1} - f^u) \mathbin{\overline{\otimes}} 0)(\Delta) \leq ((\alpha^l_{b1} \otimes \beta^l_{b1} + B_{b1}) \otimes \alpha^u_{b1})(\Delta) \qquad (3.5)
\end{aligned}$$

The network calculus based transformations can be applied to Eq. 3.5 to derive the lower bound on β^l_{b1} given by

$$\beta^l_{b1}(\Delta) \geq ((((((\alpha^u_{b1} - f^u) \mathbin{\overline{\otimes}} 0) \oslash \alpha^u_{b1}) - B_{b1}) \mathbin{\overline{\otimes}} 0) \oslash \alpha^l_{b1})(\Delta)$$

Hence, the lemma is proved.

We can compute the β^u_{b1} bound by ensuring that buffer B_{b3} does not overflow, i.e.,

$$y_{b1}(t) \leq z_{b1}(t) + B_{b3} - f^u(t), \forall t \geq 0 \qquad (3.6)$$

In order to ensure that buffer B_{ab1} does not underflow, we have

$$z_{b1}(t) \geq (1-n) \times C_1(t), \forall t \geq 0 \qquad (3.7)$$

From Eq. 3.6 and Eq. 3.7, in order to strictly ensure that B_{b3} does not overflow, we can deduce that

$$\begin{aligned}
& y_{b1}(t) \leq (1-n) \times C_1(t) + B_{b3} - f^u(t), \forall t \geq 0 \\
\Leftrightarrow\ & \beta^u_{b1}(t) \otimes a_{b1}(t) \leq (1-n) \times C_1(t) + B_{b3} - f^u(t), \forall t \geq 0 \\
\Leftrightarrow\ & \beta^u_{b1}(t) \leq ((1-n) \times C_1(t) + B_{b3} - f^u(t)) \oslash a_{b1}(t), \forall t \geq 0. \qquad (3.8)
\end{aligned}$$

Lemma 3.2.4 *From Eq. 3.1, Eq. 3.4, Eq. 3.8 and Lemma 3.2.3, the aggregate service curve* $[\beta_1^u, \beta_1^l]$ *for the PE allowing drops can be computed as*

$$\beta_1^u \leq \max\{(\beta_{ip1}^l + \beta_{b1}^u), (\beta_{ip1}^u + \beta_{b1}^l)\}$$

$$\beta_1^l \geq \min\{\beta_{nd1}^l, (\beta_{ip1}^u + \beta_{b1}^l)\}$$

where β_{nd1}^l *is the lower bound on aggregate service curve with no frame drops.*

Proof Let us first consider the lower bound of the aggregate service curve represented by the tuple $[\beta_1^u, \beta_1^l]$ for the PE allowing frame drops (the first PE in our case from Fig. 3.1). The lower bound β_1^l should atleast service a minimum number of B frames such that the number of B frames dropped do not violate the quality constraints. This condition can be satisfied if atleast β_{b1}^l B frames are serviced. In order to ensure that none of the I/P frames are dropped, it is necessary that an additional β_{ip1}^u service is provided. This gives the lower bound on the aggregate service required given by $\{\beta_{ip1}^u + \beta_{b1}^l\}$. However, in order to ensure that β_1^l does not exceed the lower bound with no frame drops, the appropriate lower bound on aggregate service curve with frame drops is $\beta_1^l \geq \min\{\beta_{nd1}^l, (\beta_{ip1}^u + \beta_{b1}^l)\}$.

Now let us consider the upper bound of the aggregate service curve for the PE allowing frame drops. The upper bound of β_1^u can be a straightforward sum of the upper bounds of individual service for both I/P frames and B frames given by $\beta_{ip1}^u + \beta_{b1}^u$. However, this is a pessimistic estimate. In order to not result in buffer overflow at B_3, the upper bound can also be such that $\beta_1^u \leq \{\beta_{ip1}^l + \beta_{b1}^u\}$. But we need to ensure that $\beta_1^u \geq \beta_1^l$. Therefore, the appropriate β_1^u required is $\beta_1^u \leq \max\{(\beta_{ip1}^l + \beta_{b1}^u), (\beta_{ip1}^u + \beta_{b1}^l)\}$.

Hence, the lemma is proved.

However the aggregate service curve can be tuned more accurately if there is an integral relationship between the number of I/P frames and number of B frames. This is demonstrated in the next lemma.

Lemma 3.2.5 *Considering the quantities in Lemma 3.2.4 and Lemma 3.2.1, if there is an integral relationship between the number of I/P frames and the number of B frames in the stream, i.e., if the*

ratio $\frac{Number\ of\ B\ frames}{Number\ of\ I/P\ frames}$ *is an integral value, the aggregate service curve* $[\beta_1^u, \beta_1^l]$ *can be computed as*

$$\beta_1^u \leq \{\beta_{b1}^u + \frac{\beta_{b1}^u + f^u}{N}\}$$

$$\beta_1^l \geq \min\{\beta_{nd1}^l, (\beta_{b1}^l + \frac{\beta_{b1}^l + f^u}{N})\}$$

where N is the ratio of the number of I/P frames to the number of B frames and β_{nd1}^l *is as defined in Lemma 3.2.4.*

Proof As in Lemma 3.2.4, the lower bound of the aggregate service curve β_1^l has to process atleast β_{b1}^l frames to satisfy quality constraints. As the ratio $\frac{Number\ of\ B\ frames}{Number\ of\ I/P\ frames} = N$, where N is an integer, the maximum number of I/P frames that need to be processed in the same time interval are $\frac{\beta_{b1}^l + f^u}{N}$. This is because β_{b1}^l is the minimum number of B frames to be processed which does not include the B frames that are dropped. So, in order to find the maximum number of I/P frames, we need to add the upper bound on number of frames dropped to β_{b1}^l so as to find the total number of B frames (Dropped + Processed). The total when divided by N gives the maximum number of I/P frames processed, given by $\frac{\beta_{b1}^l + f^u}{N}$. The sum of β_{b1}^l and the previous quantity gives one part of the lower bound. In order to ensure that the lower bound with frame drops does not exceed the lower bound without frame drops (β_{nd1}^l), we derive the lower bound as $\beta_1^l \geq \min\{\beta_{nd1}^l, (\beta_{b1}^l + \frac{\beta_{b1}^l + f^u}{N})\}$.
The same explanation holds for the upper bound when β_{b1}^l is substituted with β_{b1}^u.

Although we have found the bounds on the required service in the presence of frame drops such that a quality constraint is satisfied, the service is currently in terms of the number of frames processed in any time interval. This has to be converted into bounds on the number of processor cycles provided in any time interval. Let us now define the tuple $[\sigma^u, \sigma^l]$, which denotes the upper and lower bound on the number of processor cycles provided in a specific time interval Δ. If the maximum number of cycles required to process k frames is denoted by $cmax(k)$, then the bounds on the required processor cycles such that an input stream is processed with target quality constraints is given by

$$\sigma^u(\Delta) \leq cmax(\beta_1^u(\Delta)),$$
$$\sigma^l(\Delta) \geq cmax(\beta_1^l(\Delta)). \tag{3.9}$$

Once we find the required number of processor cycles in terms of $[\sigma^u, \sigma^l]$, we can allocate the appropriate amount of processor resources so that the incoming multimedia stream is processed subject to a target quality constraint. The advantage of this framework is that we could individually analyze the multimedia stream (stored videos or a representative video clip). The processor cycle bounds computed can then be used to schedule the decoding of multiple streams. We further validate the results obtained in the next section.

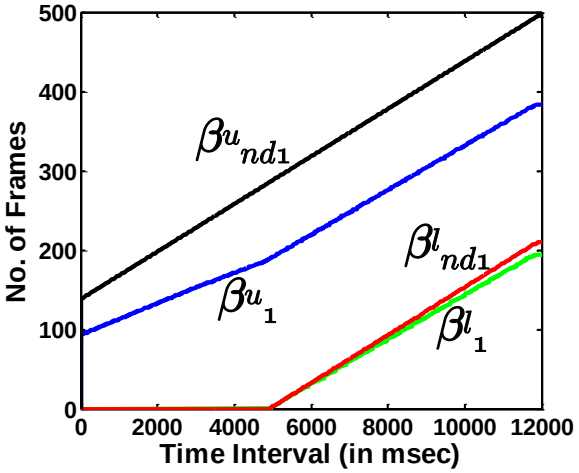
3.3 Experimental Results

In this section, we validate the formal framework presented in the previous section. The two main results presented here are:

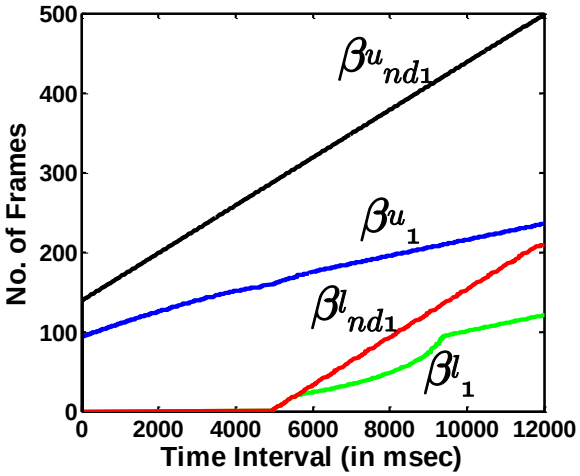
1. Reduction in the processor cycle requirements obtained as a result of the trade-off with quality.

2. Verification of the quality obtained in a scenario where multiple streams are processed adhering to their respective required processor cycle bounds derived using the formal framework.

In our experiments, we consider frame drops only in front of PE_1. Therefore, we compute the service required on PE_1 for two multimedia streams decoded simultaneously on a MPSoC platform with buffer and processor resource constraints. In particular, we first find the processor cycle bounds in accordance to the formal framework presented in Section 3.2 so that target quality constraints for both the multimedia streams are met. Then, we allocate processor cycles to the two streams such that the processor cycle bounds are not violated. The processor cycles required for the multimedia streams on PE_2 (without frame drops) can be computed without partitioning the processed stream at the output of PE_1. The procedure is similar to the computation of processor bounds for I/P frames at PE_1. The cycle requirement for each MPEG-2 task is obtained using SimpleScalar sim-profile simulator for a MIPS-like architecture. The buffer sizes are fixed at $B_{ip1} = B_{ip2} = 50$, $B_{b1} = B_{b2} = 100$, $B_{ip3} = B_{ip4} = 45$ and $B_{b3} = B_{b4} = 90$.



(a)

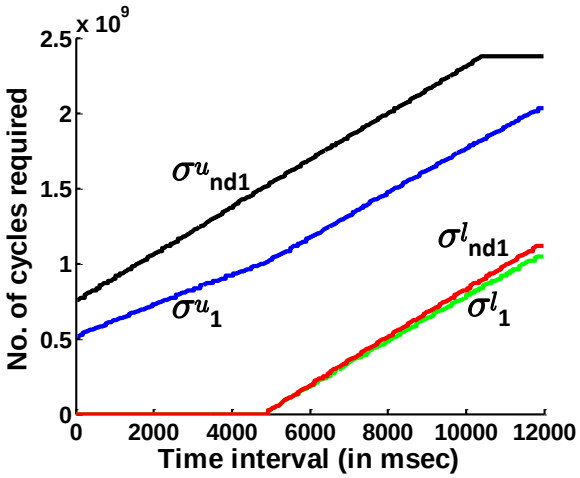


(b)

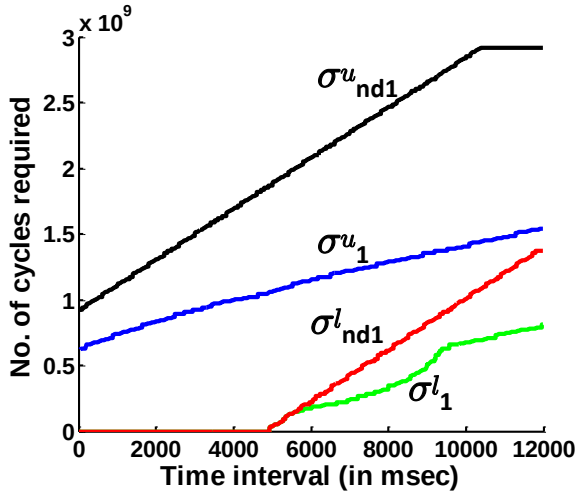
Figure 3.3: Aggregate service curves with and without frame drops for the clips (a) *cact*_080 and (b) *susi*_080.

3.3.1 Processor Cycle vs Quality trade-off

We explore this trade-off on PE_1 with buffer and processor bandwidth constraints. Here, we first present the results of processor cycle requirements obtained using the formal framework and compare it against the processor cycle requirements with no frame drops, also obtained using a formal analysis. In order to compute the processor cycle bounds with no frame drops, we use the method given in [53]. We use two videos (*cact*_080 and *susi*_080 [1]) to demonstrate the trade-off. First,



(a)



(b)

Figure 3.4: Processor cycle requirements with and without frame drops for the clips (a) *cact*_080 and (b) *susi*_080.

we present the result for the aggregate service curve $[\beta_1^u, \beta_1^l]$ with frame drops and compare it with the aggregate service curve $[\beta_{nd1}^u, \beta_{nd1}^l]$ without frame drops in Fig. 3.3. In this experiment, frames are dropped such that a worst-case quality of $30dB$ is not violated. It is observed from the graph that the lower service curves with (β_1^l) or without (β_{nd1}^l) frame drops start processing only after an initial latency during which the buffer is not full and there is no frame drop and therefore no loss in quality. After that initial latency, both the curves increase at different rates because with a tolerable

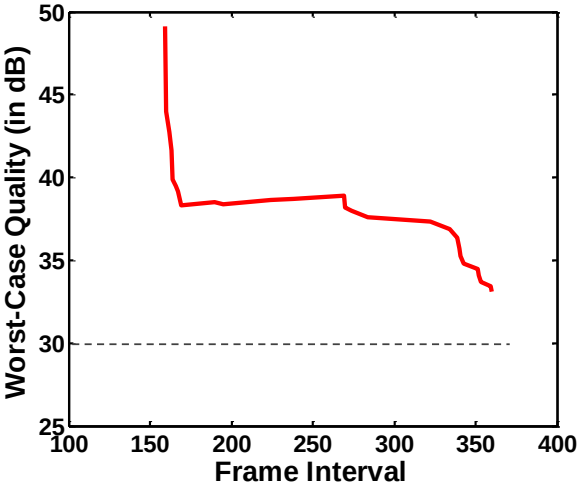
loss constraint, β_1^l need not process all the frames and some frames can be dropped. The reduction in the upper service curve is also observed for the frame drop case (β_1^u) because even though β_{nd1}^u will not cause buffer overflows in the intermediate stage, due to frame drops in the first stage, the upper aggregate service curve β_1^u decreases as shown in Fig. 3.3. The observations listed above are seen for both the video clips used to conduct experiments. However, it is evident from Fig. 3.3(b) that the reduction in service is more for *susi*_080 in comparison to *cact*_080 because the adjacent frames in *susi*_080 are more similar in comparison to the adjacent frames in *cact*_080, which allows more frames to be dropped for *susi*_080 with the same target worst-case quality constraint.

We also plot the curves for the processor cycle requirements for each video clip with a worst-case quality constraint of 30 *dB* (shown as the tuple $[\sigma_1^u, \sigma_1^l]$) and compare it with the processor cycle requirements without any quality loss (no frame drops) (shown as the tuple $[\sigma_{nd1}^u, \sigma_{nd1}^l]$) in Fig. 3.4. The characteristics of the aggregate service curves is reflected in these curves also as the processor cycle requirements increase with the increase in service requirements.

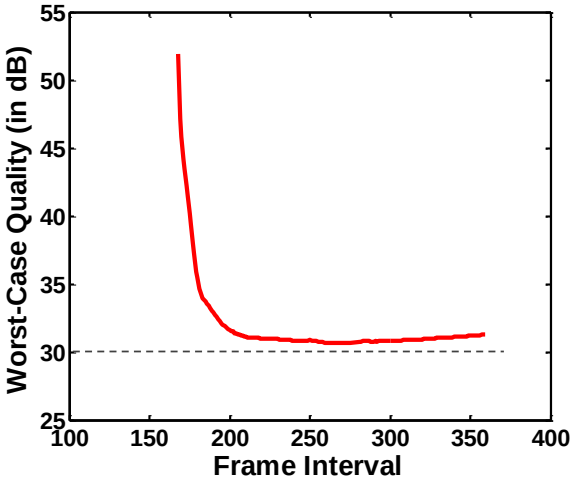
Significance of the result: This analytical framework allows the flexibility to trade-off processor bandwidth with application quality in the context of decoding multiple multimedia streams. For example, keeping all the above buffer sizes the same, it has been observed that for a quality value of $PSNR = 30$ *dB*, the video clip *cact*_080 gives a maximum processor bandwidth savings of 7196 processor cycles for any analysis interval of 1 *ms*, which corresponds to a bandwidth savings percentage of 21.77%. Similarly, for the video clip *susi*_080, it has been observed that the maximum processor bandwidth savings is 49703 processor cycles for any analysis interval of 1 *ms*, which corresponds to a bandwidth savings percentage of 48.73%. These results strengthen the case of this analytical framework, which will be very useful to provision appropriate processor bandwidth (using any scheduling scheme) such that each stream satisfies its own individual quality constraints. This further helps in saving the processor bandwidth allocated to one stream so as to serve other incoming streams.

3.3.2 Verification of the Processor Cycle Requirements

The processor cycle requirement curves with frame drops ($[\sigma_1^u, \sigma_1^l]$) obtained in the previous section for both the clips is used in this section to run simulations in a multiple video clip decoding scenario,



(a)



(b)

Figure 3.5: Simulation results for quality in a multiple stream decoding scenario for (a) *cact*_080 and (b) *susi*_080.

which is one of the scenarios that can use this framework. PE_1 is assigned a frequency of 500 MHz. The processing cycles are allocated to the video clips in accordance to the cycle requirement bounds $[\sigma_1^u, \sigma_1^l]$ obtained in the previous section. The processor cycles are also allocated in an as late as possible (ALAP) manner such that the video clips are processed at the end of every time interval. This is done in order to ensure that buffer occupancy is the maximum and does not result in quality reduction below the target worst-case quality of 30 *dB*. It is observed from Fig. 3.5 that the obtained

quality for both the videos using the processor cycle bounds does not fall below 30 dB, which is the target for the experiment. On the other hand, it is also seen that *susi*_080 achieves a quality much closer to the target worst-case quality of 30 dB, while *cact*_080 is a little above 30 dB. This is because the variation in video is much higher for *cact*_080 and so the cycle requirements obtained for it are more pessimistic in comparison to those obtained for *susi*_080.

3.4 Summary

In this chapter, we present a mathematical framework to derive the service requirements for a video clip given a target quality constraint, which in turn helps to find the processor cycle requirements for the clips in a time interval. This framework was verified in a multiple video processing setup as would be found in a PiP-like application where the analytically obtained processor cycle requirements helps in processing the videos, while meeting the target quality constraints. Although, this framework was verified using in a multiple video stream processing scenario, it would be similarly applicable for processing video+audio or video+graphics/games stream. The experimental results verify our claim that a quality-driven service dimensioning helps in saving vital processor bandwidth for processing multimedia streams. The observed processor bandwidth savings for the video clips *cact*_080 and *susi*_080 were 21.77% and 48.73% respectively. The experimental setup was also verified to see if the quality constraints were satisfied with the derived quality-driven processor bandwidths for two video streams scheduled in an ALAP schedule.

There are several possible directions in which this work can be extended. Firstly, it can be extended to the scenario with other media streams (e.g., audio and graphics) apart from just multiple video streams where processor bandwidth share is derived for the mix of multimedia streams. This framework can also be extended to take into account the quality-driven buffer dimensioning objective in conjunction with service dimensioning discussed here to derive a pareto set of buffer and service values for a target quality constraint.

Chapter 4

Quality and Thermal Aware Multimedia Processing

With technology scaling, there is a steep rise in the power consumption of systems-on-chip (SoCs). The higher power densities lead to undesirable hot spots (i.e., localized high temperature points) on chip. As a SoC is subjected to higher temperatures, its reliability is affected adversely. This has a long term impact on the life of the system as the desired functionality is disturbed over a period of time. Traditionally, expensive cooling packages were designed for the worst-case peak temperatures [67]. However, this is not a viable solution for portable devices which run on tight power and cost budgets. Therefore, researchers in the embedded systems community have widely accepted Dynamic Thermal Management (DTM) techniques to keep a check on the on-chip peak temperature.

There are two types of DTM techniques widely employed namely Dynamic Voltage/Frequency Scaling (DVFS) and Dynamic Power Management (DPM). DVFS has been widely used for power and energy optimization ([68], [69], [70]), but recently there have been efforts to use DVFS for thermal optimization. In [71], DVFS is used to minimize peak temperature, where task voltage selection is done at design time with thermal optimization objectives. Similarly some other works take temperature into consideration while selecting voltage for energy optimization ([72], [73]). However, the two main issues with DVFS are the requirement of a support for multiple voltage/frequency settings and the overhead associated with switching from one voltage/frequency to the other. On

the other hand, DPM consists of operation modes and the system can be put into low power modes during times with low/no activity [74].

As power hungry multimedia applications are widely executed on portable devices, DTM has been extensively employed to control the thermal profile of a system running multimedia players. Here, DTM has been used to control temperature with the help of two strategies namely *reactive* and *predictive*. In reactive DTM algorithms, the temperature is monitored at fixed intervals and response mechanisms are invoked if the temperature reaches a trigger value. However, this strategy requires responses with low time overhead [75]. In contrast, predictive DTM algorithms exploit the properties of multimedia applications to schedule the appropriate frequency and/or architecture configuration so that the peak temperature is always under control [75]. In [76], the complexity of a MPEG-4 video frame is predicted using information from previous group of picture (GOP), which in turn is used to determine the appropriate frequency setting such that the performance does not degrade under thermal constraints. However, all the above methods do not tolerate frame drops and hence some quality loss. All the above methods also use DVFS which has its issues as discussed earlier.

4.1 Motivation

There exist several works in literature that trade-off quality for keeping the temperature below safe limits. In [77], the authors present a DTM technique that allows spatiotemporal quality degradation for MPEG-2 frame decoding. The spatial quality degradation is also referred to as Intra-frame spatial degradation, but does not completely ensure the safe thermal state of operation. On the other hand, temporal or Inter-frame quality degradation ensures safe thermal state at the cost of more quality degradation. Therefore, the authors have used one or a combination of these two degradation mechanisms according to the available slack and the thermal state of the processor. A similar quality trade-off mechanism is used in another DTM technique that is employed at the group of pictures (GOP)-level [78]. A DTM mechanism was proposed in [40], where the application characteristics are captured as a probability distribution of the decoding cycle requirement of a frame. This probability distribution is obtained at runtime by profiling the application. The authors presented that the peak temperature of the processor running multimedia codecs was lowered compared to the

state-of-the-art, while reducing the number of frame drops. Although the above mentioned works trade-off quality for safe thermal profile, they do not obtain the safe thermal profile by mathematically bounding quality degradation. In [79], a power-aware H.264 encoder-decoder pair is proposed that enables power savings at the expense of controlled quality reduction. Here, they use prioritized slice groups within a frame, which, if skipped, lead to varying known quality degradations. However, in comparison to [79], our work focuses on thermal management. In addition, our *Quality of Service* (QoS) measure is a frame interval based quantity which is defined in Section **??**. Moreover, we do not prioritize macroblocks within frames and we simply prioritize dropping of B frames.

In this work, we use DPM to control the peak temperature in the context of video decoder applications running on MPSoC platform. In the DPM technique that we use, there are active and idle modes of operation. The system is put into idle mode when there is a necessity to keep the temperature below the maximum temperature limit. The addition of idle times increases the end-to-end delay of the system as processing is stalled. Hence, reduction of end-to-end delay is an important requirement in a DPM method. Recently, a theoretical framework was presented in [74] to derive the scheduling strategy that minimizes the end-to-end delay (by minimizing the inserted idle times) under thermal constraints, for a set of tasks mapped onto distributed systems. However, the work in [74] derives an optimal task schedule in order to minimize the idle times inserted and does not consider application quality constraints for idle time insertion. Our work is different from the problem addressed in [74] as we attempt to process the multimedia streams in a quality-aware manner under thermal constraints. In the case of a video decoder application, it is possible to use frame drops to reduce the idle times while adhering to a certain quality and peak temperature constraint. Therefore, in this work, we present an online DTM policy combined with an application level technique that uses frame drops to reduce end-to-end delay, under thermal constraints, of a video decoder application mapped onto a MPSoC platform. Although we use the example of a video decoder application, the idea is applicable in other multimedia applications also.

As reliability of a system running power hungry video player applications is adversely affected by increase in temperature, peak temperature has become an important factor in the design of portable devices. Therefore, thermal management techniques based on DPM insert idle times interspersed with intervals when processing takes place. In order to keep the peak temperature T_{peak} below a

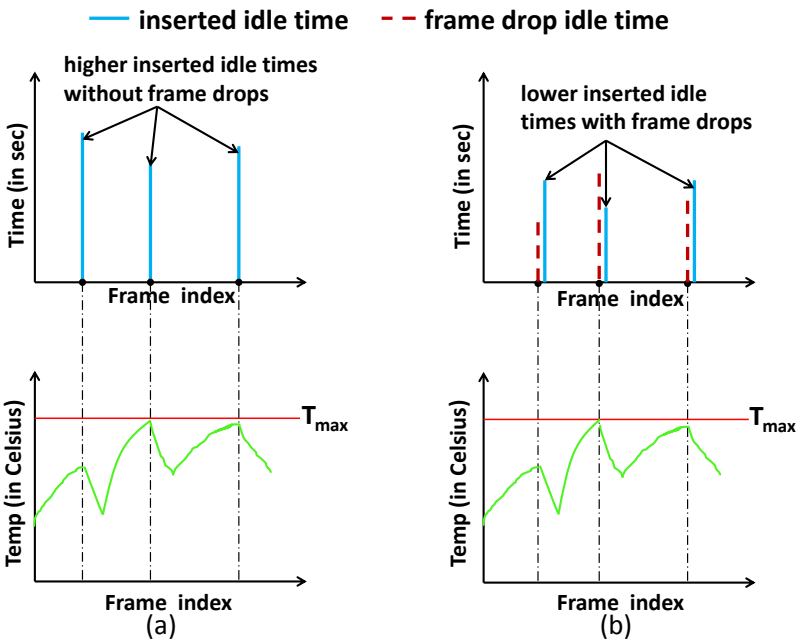


Figure 4.1: Illustration of reduction in inserted idle times using frame drops: (a)inserted idle times without frame drops and (b)inserted idle times with frame drops

threshold value T_{max}, idle times (shown as vertical lines) are inserted as shown in Fig. 4.1(a). *The height of the vertical line denotes the value of idle time inserted.* It shows a toy example depicting the idle times inserted before the frame represented by the frame index value. The lower graph in Fig. 4.1(a) shows the temperature after the processing of each frame when idle times are taken into consideration. However, addition of idle times increases the end-to-end latency. Therefore, in Fig. 4.1(b), we depict a possibility that T_{peak} can be kept under T_{max} and the idle time values inserted can be reduced if certain frames (shown as dotted vertical lines) are dropped such that the prespecified quality constraint is satisfied. *The height of the dotted vertical line denotes the value of processing time of dropped frame*. Let us now understand the multimedia frame properties so as to decide a scheme for frame dropping.

The sequence of a typical MPEG-2/MPEG-4 encoded video was explained in Section **??**. From the description, we deduce that it is possible to drop some B frames in a decoder mapped onto a MPSoC platform such that the video quality does not deteriorate significantly. This property of a video clip can be used to find a specific frame drop pattern such that a predetermined quality constraint is satisfied. This frame drop pattern can be used to compute the reduced idle times required as shown

in Fig. 4.1(b). It is further envisaged that frame drops can be used to completely eliminate insertion of idle times in certain video clips under thermal constraints. Although this work demonstrates the effectiveness of the approach in video clips with I,P and B frames, the same technique is applicable if there are no B frames in the video clip. In such a scenario, we drop P frames instead of B frames in such an order that the P frame that is not a reference frame is first dropped. However, this could increase the idle time inserted if the quality losses due to concealment of P frames is high. This is dependent on how the stream has been coded. On the other hand, if B frames are used as references in any standard, the frame drops will be decided based on the existing frame dependencies. In this work, we only consider the case with streams having B frames where B frames are not reference frames. However, the technique presented here could also be adapted to the other scenarios mentioned above.

4.2 Proposed Framework

This section presents an overview of our combined offline and online method that employs application level technique of frame drops along with DTM to control the peak temperature of the system, thereby reducing the magnitude of inserted idle times. This in turn enables to reduce the end-to-end latency of the system. We use a quality constraint to upper bound the number of frames dropped in a particular window of frames. In addition, we use a thermal model to compute the thermal profile of the platform. We first describe the underlying MPSoC platform.

4.2.1 Platform Description

In this work, we use frame drops to reduce the value of idle times inserted to adhere to quality and peak temperature constraints on an MPSoC architecture as shown in Fig. 4.2.The architecture consists of two PEs, PE_1 and PE_2, each being serviced at a rate that allows the playback rate to be satisfied under thermal and quality constraints. Each PE is mapped with a set of tasks from the target decoder application. The PEs also each have a buffer in front of them, shown as B_1 and B_2. Both the buffers are sufficient enough to handle the bursts in incoming data and therefore do not cause overflow. Frames are dropped in front of the first PE and both the PEs are put into idle mode during

the processing time slots of dropped frames. This application level technique helps in reducing the idle times required to keep the peak temperature of the PEs, denoted by T_{peak1} and T_{peak2}, below the allowed maximum temperatures denoted by T_{max1} and T_{max2}. Frames are also dropped in such a manner that the worst case quality for any interval of L frames, denoted by $Q^l(L)$ does not fall below the target quality constraint Q_{target}.

4.2.2 Preliminaries

Before presenting a formal problem definition, we introduce the two integral models used in this framework, namely the QoS model and the thermal model. The QoS model quantifies the quality provided by the MPSoC platform. The thermal model aids in obtaining the thermal profile of the platform components, especially the PEs.

QoS model: The QoS model used in this work defines the quality metric as frames are dropped in order to efficiently control the thermal profile. We use the worst case quality metric denoted by $Q^l(f,L)$, which is PSNR value of the input stream at the output of the decoder. PSNR value is computed by calculating the MSE between the dropped frame and the concealment frame or the frame that replaces the dropped frame (computed as in [80]). There are many possible concealment frame candidates for a dropped frame as shown in Fig. 1.1. Of all the possible candidates, we choose the frame that causes maximum distortion or MSE as the concealment frame. Therefore, we compute the maximum MSE ($MSE_{max}(f,L)$) if f number of frames are dropped in any interval of L frames across the entire length of the video clip by examining all combinations of f frames. The

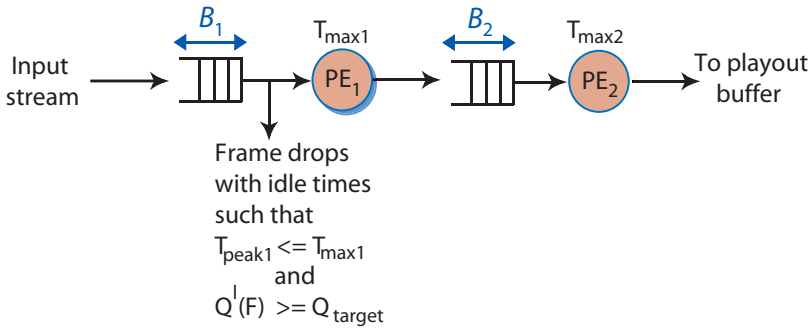


Figure 4.2: MPSoC platform using frame drops to reduce idle times under thermal and quality constraints

worst case quality metric $Q^l(f,L)$ is then computed as

$$Q^l(f,L) = 10 \times log_{10} \frac{(255 \times 255 \times L)}{(MSE_{max}(f,L))} \tag{4.1}$$

If Q_{target} is the worst quality constraint value that should be satisfied always, then for any frame interval L, the worst case quality $Q^l(f,L) \geq Q_{target}$. This constraint places a restriction on the number of frames that can be dropped in any frame interval. Let us denote this dropped frame set as $D = \{f_{drop}(L) | 1 \leq L \leq N\}$ where $f_{drop}(L)$ is the maximum number of frames that can be dropped in an interval of L frames and $f_{drop}(L) \in [1,L]$. N is the total number of frames in the video clip. If $f_{drop}(L)$ frames are dropped in an interval of L frames, we denote the worst-case quality as $Q^l(L)$, which is shown in Fig. 4.2. The two important aspects associated with the QoS model that we use are

1. **Non propagation of errors due to frame losses beyond the GOP**: As only the B frames are dropped, the concealment frame for each dropped frame belongs to the same GOP as the dropped B frame. Hence, the errors due to concealment are not propagated beyond the GOP. In short, the concealment frame of a dropped frame cannot be a frame that belongs to a completely different scene.

2. **Usage of QoS model**: The QoS model can be used in two contexts. For stored videos, we can generate the QoS metric defined in Eqn. 1.3 for the specific videos and use them online to take appropriate frame drop decisions. On the other hand, if the input video clip is not a stored video, then the QoS metric is generated using a set of representative clips (can be called a representative QoS metric) and the incoming video is expected to adhere to the representative QoS metric. Representative video clips are widely used to design a system running multimedia applications. However, in this particular work, we generate the QoS metric for each video clip.

Thermal model: In this work, we use the lumped RC model presented in [70] to compute the thermal profile of our PEs. If the average power dissipated by the PE over a time t is P Watts, R is the thermal resistance in $^\circ C/Watt$, C is the thermal capacitance in $Joules/^\circ C$, T_{amb} is the ambient temperature and T_{init} is the initial temperature of the PE, then the temperature of the PE at the end

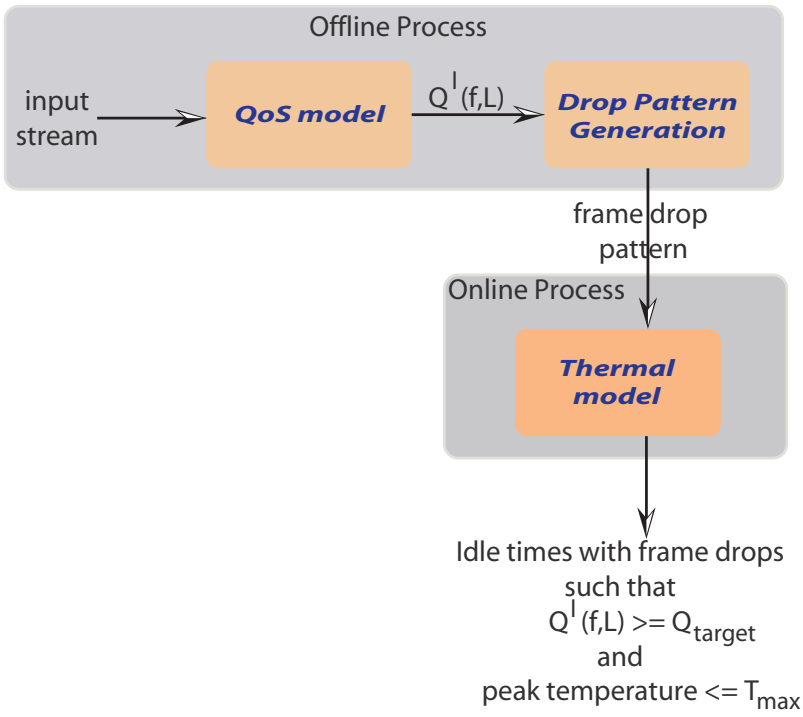


Figure 4.3: High level schematic diagram of Quality and Thermal-aware Idle time Insertion

of time t is given by

$$T(t) = T_s + (T_{init} - T_s)e^{-\frac{t}{RC}} \tag{4.2}$$

where T_s is the steady state temperature of the system given by $T_s = P \times R + T_{amb}$. We use two modes of operation of the PE to implement DPM namely the **active** and **idle** mode. Therefore, we denote the steady state temperatures in the active and idle modes as T_a and T_i respectively. For a proper thermal model, $T_a > T_i$. The temperature rises towards T_a in the active mode and falls towards T_i in the idle mode. Let us denote the idle times introduced in the idle mode by the set $I = \{t_{idle}(L) | 1 \leq L \leq N\}$, where $t_{idle}(L)$ is the magnitude of idle time introduced before the frame index L.

4.2.3 Problem Definition

Given the input video clip, the QoS model, the thermal model, the target quality contraint Q_{target} and the maximum allowed peak temperature T_{max}, compute the dropped frame set D in order to reduce

the idle times in I.

The two sets D and I are computed using the two stages of the framework. The first stage is an offline process that first computes the dropped frame set D given Q_{target}. D is then used to find the frame drop pattern in the video clip adhering to quality constraints for all frame intervals. The frame drop pattern is then used along with the thermal model to compute idle times (or set I) inserted after the dropped frame such that the temperature does not exceed T_{max} until the next dropped frame. This is an online process. We now explain the two stages namely *Drop Pattern Generation* and *Quality and Thermal Aware Idle Time Insertion*. The high level schematic diagram illustrating the overall scheme is shown in Fig. 4.3.

4.3 Drop Pattern Generation

The inputs that this stage require are the worst case quality values if f frames are dropped in an interval of L frames given by $Q^l(f,L)$ and the target quality constraint value Q_{target}. This stage is an offline process and consists of two parts. The first part uses the constraint $Q^l(f,L) >= Q_{target}$ to find out the maximum values of f for each L (which is $f_{drop}(L)$) by iterating over all values of $f \in [1,L]$. However, once we get the maximum number of frames that can be dropped for any frame interval, we need to find the drop pattern in the decoding order of the frames. This is obtained using Algorithm.4.

We now explain Algorithm.4. The input to the algorithm is $f_{drop}(L)$ as discussed earlier. The output expected from the algorithm is a list of frames that are dropped in accordance to the constraint $Q^l(L) >= Q_{target}$. We use a variable *sum_seq* to keep a record of how many frames have been dropped in a frame interval. The variables are initialized as shown in Line 1. The algorithm is iterated over the entire length of the video, i.e., for all the frames 1 to N. We drop only the B frames and therefore check if the current frame is a B frame (in Line 4). If the current frame corresponding to index i is a B frame, then we initially assume that it is going to be dropped. This is reflected in Line 5 and 6 where we assign 1 to *sum_seq* and $drop\_pattern(i)$, which signifies that the current B frame is dropped. Then, a sliding window is used in which the window size is increased from 2 to i. This is implemented using the for loop in Line 7. For each window size, the sum of frame drops

Algorithm 4 Computing drop pattern

Input: $f_{drop}(L)$ - Maximum number of frames dropped for any frame interval L satisfying Q_{target};
Output: $drop\_pattern(1:N)$

```
1:  sum_seq ← 0, drop_pattern(1:N) ← 0;
2:  —Computing the Drop pattern—
3:  for i = 1 to N do
4:    if (Bframe) then
5:      sum_seq = 1;
6:      drop_pattern(i) = 1;
7:      for j = (i−1) to 1 do
8:        sum_seq = sum_seq + drop_pattern(j);
9:        if (sum_seq > f_drop(i−j+1)) then
10:         drop_pattern(i) = 0;
11:         break;
12:       end if
13:     end for
14:   end if
15: end for
```

is counted assuming that the current B frame is dropped. The *sum_seq* value is then compared with $f_{drop}(i-j+1)$, where $i-j+1$ is the window size (Line 9). If the *sum_seq* value is greater than $f_{drop}(i-j+1)$ for any j, then the current B frame is not dropped as it violates the target quality constraint value. In this algorithm, it is possible that $f_{drop}(i-j+1)$ value for small window sizes can override the $f_{drop}(i-j+1)$ value for the larger window sizes. This means that even if the larger window sizes allow many frame drops, the lower number of frame drops allowed in smaller window sizes do not allow the $f_{drop}(i-j+1)$ value for the larger window sizes to be attained. The speed of execution of Algorithm.4 depends on the size of the video, but the large execution time for large clips is acceptable as this algorithm runs offline.

In our experiments, we use the more restricted $f_{drop}(i-j+1)$ values corresponding to the smaller window sizes, but the quality metric can be a bit relaxed. This trade-off will enable more frames to be dropped and therefore higher reductions in idle times that are introduced. The online process of inserting reduced idle times with frame drops is shown in the next section.

4.4 Quality and Thermal Aware Idle Time Insertion

Algorithm 5 Computing the quality and thermal aware idle times

Input: $drop\_pattern(1:N)$ and $\gamma^{u}(l)$ for $1 \leq l \leq N$;
Output: $temp(1:N)$ and $idle\_time(1:N)$
1: $temp(1:N) \leftarrow T_{init}$, $idle\_time(1:N) \leftarrow 0$,
$T\_idle\_end \leftarrow 0$ and $H \leftarrow 0$;
2: —Computing the idle times—
3: **for** $i = 1$ to N **do**
4: **if** $(drop\_pattern(i) == 0)$ **then**
5: Find the longest active run k from $drop\_pattern(1:N)$;
6: **if** $(H == 0)$ **then**
7: Compute T_idle_end using the offline $\gamma^{u}(k)$;
8: **else**
9: Compute T_idle_end using the online updated $\gamma^{u}(k)$ as in (4.5);
10: **end if**
11: Compute $idle\_time(i)$ as in (4.4);
12: **end if**
13: **if** $(drop\_pattern(i) == 0)$ **then**
14: **if** $(idle\_time(i) == 0)$ **then**
15: Compute $temp(i)$ as in (4.2) with $T_s = T_a$, $T_{init} = T_{prev}$ and $t = proc\_cycles(i)$;
16: **else**
17: Compute $temp(i)$ as in (4.2) with $T_s = T_a$, $T_{init} = T\_idle\_end$ and $t = proc\_cycles(i)$;
18: **end if**
19: **else**
20: Compute $temp(i)$ as in (4.2) with $T_s = T_i$, $T_{init} = T_{prev}$ and $t = proc\_cycles(i)$;
21: **end if**
22: **if** $(H < HIST\_MAX)$ **then**
23: Increment H;
24: **end if**
25: **end for**

This stage is an online process and requires inputs from the offline process, i.e., it requires the drop pattern obtained as discussed in Section 4.3. Once we know the frames that will be dropped, the entire execution on PEs can be divided into two operation modes. The frames that are not dropped constitute the active mode of operation and the other frames result in idle operation mode. In the idle operation mode, some additional idle time is inserted if required. In this section, we present an online algorithm to derive this idle time such that the peak temperature T_{peak} does not exceed the maximum allowed temperature on the PE, i.e., T_{max}.

In the context of frame drops, to compute the appropriate idle times, we find the worst case processing time for the longest active run, i.e., the maximum number of consecutive frames that are not

dropped. The longest active run is computed by considering all the active runs between two dropped frames. Let the longest active run be denoted by k frames. If $W(M)$ is the workload cycles required for processing the first M frames, then γ^u is defined by $\gamma^u(n) = \max_{\forall M \geq 0}\{W(M+n) - W(M)\}$. $\gamma^u(n)$ is the maximum number of processor cycles required for processing n consecutive frames. Therefore, the maximum number of processor cycles required for the longest active run is given by $\gamma^u(k)$. Given $\gamma^u(k)$, we can find the safe temperature at which we should start the active mode so that the temperature does not exceed T_{max}. Here, we should also consider the overhead required to perform this online idle time insertion. Let us denote this overhead cycles as OC. The safe temperature at the end of the idle period can be denoted as T_idle_end. This safe temperature can be such that the temperature at the end of the longest active run be less than or equal to T_{max}. Then, we have

$$\begin{aligned} & T_a + (T\_idle\_end - T_a)e^{-\frac{(\gamma^u(k)+OC)\times T_p}{RC}} \leq T_{max} \\ \Leftrightarrow\ & T\_idle\_end \leq T_a + (T_{max} - T_a)e^{\frac{(\gamma^u(k)+OC)\times T_p}{RC}} \end{aligned} \tag{4.3}$$

where T_p is the time period. We choose the highest value of T_idle_end that satisfies the above constraint.

The idle times introduced are computed based on the extra idle times obtained due to frame drops. These introduced idle times are appended to the end of the dropped frame. Let the temperature at the end of the dropped frame be T_{prev}. Now we compute the introduced idle time (t_{idle}) as

$$\begin{aligned} & T_i + (T_{prev} - T_i)e^{-\frac{t_{idle}}{RC}} \leq T\_idle\_end \\ \Leftrightarrow\ & t_{idle} \geq R \times C \times \log\Big(\frac{T_{prev} - T_i}{T\_idle\_end - T_i}\Big) \end{aligned} \tag{4.4}$$

We choose the lowest value of t_{idle} that satisfies the above inequality.

The computation of T_idle_end can be made more tighter by deriving a tighter estimate of $\gamma^u(k)$. This is done by using a history based approach. Let the length of the history or the number of previous frame cycle requirements that is maintained be H. The accumulated processor cycles for the last H frames is therefore denoted by $proc_H$. Then the maximum processor cycles for the longest active run k is given by

$$\gamma^u(k) = \min_{0 \leq H \leq HIST\_MAX}(\gamma^u(k+H) - proc_H) \tag{4.5}$$

The entire online process is shown as Algorithm.5. First, the longest active run (k) is obtained from $drop\_pattern(1:N)$ (Line 5). Then T_idle_end is computed based on the available history of processing cycles (Lines 6-10). Once T_idle_end is obtained, we compute $idle\_time(i)$ in Line 11. Finally, we compute the temperature of the PE at the end of processing of the current frame (Lines 13-21). History length recorded in H is incremented until its maximum value $HIST\_MAX$ (Lines 22-23). These H recent history values of frame processor cycles are used to obtain tighter temperature values. Although the amount of history maintained affects OC introduced in the analysis, it is envisaged to be insignificant in comparison to the cycle requirements for the active run.

Although, in this work, we insert idle times at the end of each frame drop if required, it is possible to insert the idle time at any position after the frame drop and before it reaches T_{max}. Therefore, we need to first prove that the frame drops help in reducing the idle time introduced in comparison to the scenario without frame drops. Inserting idle times affects the end-to-end performance of the system. Hence, we prove that if a certain set of idle times without frame drops satisfies the display rate, then the idle times introduced in the same slots in the context of frame drops also satisfies the display rate.

Before presenting the results, we prove two theorems that validate the usefulness of the quality aware multimedia processing for adhering to thermal constraints.

Theorem 4.4.1 *The value of inserted idle times with frame drops is always less than or equal to the inserted idle times without frame drops.*

Proof Let us consider two scenarios here - one where inserted idle time interval is less than frame drop interval and the other where inserted idle time interval is greater than frame drop interval. Let us denote the idle time interval by L_I and the frame drop interval by L_{FDI} as shown in Fig. 4.4(b) and Fig. 4.4(a) respectively. The idle time insertions in the absence of frame drops is shown in Fig. 4.4(b).

Case I: When $L_I < L_{FDI}$.

The idle times (in the frame drop context) are inserted in the same slot as in the scenario without frame drops. In this scenario every pair of successive inserted idle times denoted by I_i where $i \in [1,N]$, has zero or more frame drops in between them (in the context of frame drops). If there

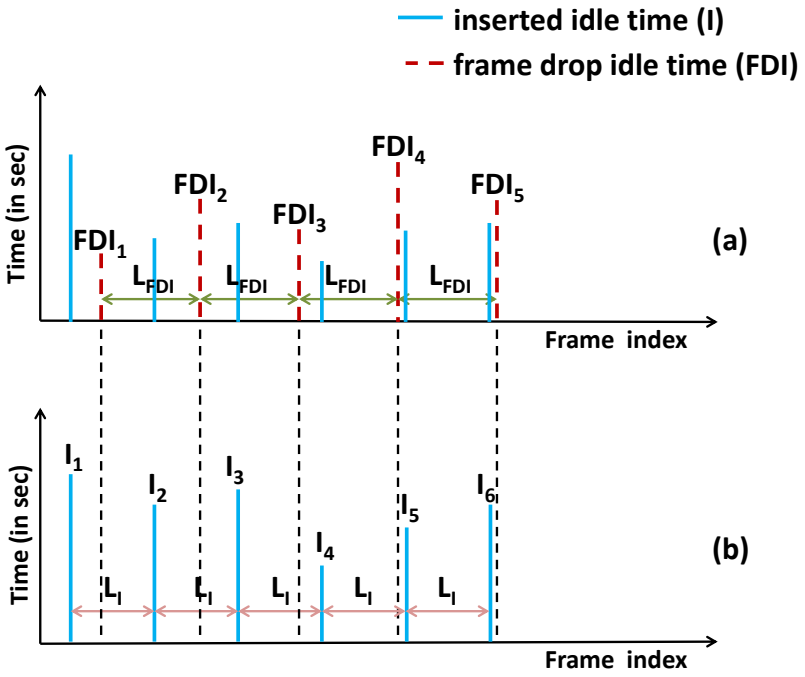


Figure 4.4: (a) Lower inserted idle time with Frame drop idle time (with frame drop interval L_{FDI}) and (b) Inserted idle time with no frame drops (with idle time interval L_I).

are no frame drops, then the value of idle times inserted is similar to the case without frame drops. However, if there is any frame drop in between any two successive inserted idle times, then the temperature (in the frame drop scenario) at the end of the frame drop slot is lower than the temperature at the same point, without frame drops. Then by *thermal monotonicity property* (If at some point of execution T, one sequence has higher temperature than the other, then at $T+\Delta$, the first sequence will be at a higher temperature given that during Δ both sequences executed in the same mode of operation), the temperature just before the next idle time slot is lower for the scenario with frame drops. Hence, a smaller idle time value is inserted at this point. In Fig. 4.4(a), the first such point is just before I_2.

Case II: When $L_I > L_{FDI}$.

The theorem is valid in this case because there are always more than one frame drops between two successive inserted idle times and then it follows the argument of Case I.

Case III: When $L_I = L_{FDI}$
The frame drop idle time is utilized to insert lower idle times.

Hence, the theorem is proved.

Theorem 4.4.2 *If a set of inserted idle times satisfies the end-to-end performance of the video or adheres to the display rate, the set of inserted idle times with frame drops also satisfies display rate.*

Proof This proof is directly obtained by using Theorem 4.4.1. As the inserted idle time decreases with addition of frame drops as shown in Fig. 4.4(a), the end-to-end delay reduces. Therefore, the display rate is satisfied as it is satisfied in the case of Fig. 4.4(b).

Finally, we theoretically evaluate the use of workload history. As mentioned before, we use the workload history to dynamically update $\gamma^{\mu}(k)$. Let us denote $\gamma^{\mu}(k)$ with or without history as $\gamma^{\mu}_{h}(k)$ and $\gamma^{\mu}_{nh}(k)$. Let the idle time introduced at any stage i with or without history be denoted by $t^{h}_{idle,i}$ and $t^{nh}_{idle,i}$ respectively. We prove now that using workload history helps in reducing the accumulated idle time and therefore the delay.

Theorem 4.4.3 *Accumulated idle time inserted with $\gamma^{\mu}_{h}(k)$ is always lesser than or equal to the accumulated idle time inserted with $\gamma^{\mu}_{nh}(k)$, i.e., $\sum_{i=0}^{n} t^{h}_{idle,i} \leq \sum_{i=0}^{n} t^{nh}_{idle,i}$.*

Proof Let us first denote the temperature at the end of $i-th$ frame drop with and without workload history as $T^{h}_{afd,i}$ and $T^{nh}_{afd,i}$ respectively. The safe temperature at the end of the idle time insertion such that the temperature never exceeds T_{max} for an active run $\gamma^{\mu}_{h}(k)$ and $\gamma^{\mu}_{nh}(k)$ are $T\_idle\_end^{h}_{i}$ and $T\_idle\_end^{nh}_{i}$. As $\gamma^{\mu}_{h}(k) \leq \gamma^{\mu}_{nh}(k)$, substituting these in Eqn. 4.3 gives $T\_idle\_end^{h}_{i} \geq T\_idle\_end^{nh}_{i}$. By ignoring the overhead cycles, the temperatures before the $i-th$ frame drop can be derived as

$$\begin{aligned} T^{h}_{bfd,i} &= T_a + (T\_idle\_end^{h}_{i-1} - T_a)e^{-\frac{c_i \times T_p}{RC}} \\ T^{nh}_{bfd,i} &= T_a + (T\_idle\_end^{nh}_{i-1} - T_a)e^{-\frac{c_i \times T_p}{RC}} \end{aligned} \tag{4.6}$$

where c_i is the actual number of processor cycles required for the active run before the frame drop.

Now the temperature at the end of the frame drop can be calculated as

$$T^{h}_{afd,i} = T_i + (T^{h}_{bfd,i} - T_i)e^{-\frac{cdrop_i \times T_p}{RC}}$$
$$T^{nh}_{afd,i} = T_i + (T^{nh}_{bfd,i} - T_i)e^{-\frac{cdrop_i \times T_p}{RC}} \tag{4.7}$$

where $cdrop_i$ is the number of processor cycles required for processing the dropped frame if it was not dropped. From Eqn. 4.6, Eqn. 4.7 and subtracting $T^{nh}_{afd,i}$ from $T^{h}_{afd,i}$, we can deduce that

$$T^{h}_{afd,i} - T^{nh}_{afd,i} = (T\_idle\_end^{h}_{i-1} - T\_idle\_end^{nh}_{i-1})e^{-\frac{(c_i + cdrop_i) \times T_p}{RC}} \tag{4.8}$$

This shows that the difference in the temperatures (with or without workload history) after the $i-th$ frame drop is lesser than the difference in temperatures at the end of $(i-1)-th$ idle time. We can express the temperature at the end of the $i-th$ idle time as

$$T\_idle\_end^{h}_{i} = T_i + (T^{h}_{afd,i} - T_i)e^{-\frac{t^{h}_{idle,i}}{RC}}$$
$$T\_idle\_end^{nh}_{i} = T_i + (T^{nh}_{afd,i} - T_i)e^{-\frac{t^{nh}_{idle,i}}{RC}} \tag{4.9}$$

From Eqn. 4.9, we can derive the required idle times as

$$t^{h}_{idle,i} = R \times C \times \log\Big(\frac{T^{h}_{afd,i} - T_i}{T\_idle\_end^{h}_{i} - T_i}\Big)$$
$$t^{nh}_{idle,i} = R \times C \times \log\Big(\frac{T^{nh}_{afd,i} - T_i}{T\_idle\_end^{nh}_{i} - T_i}\Big) \tag{4.10}$$

The summation of the idle times derived in Eqn. 4.10 is given by

$$\sum_{i=1}^{n} t^{h}_{idle,i} = R \times C \times \log\Big(\prod_{i=1}^{n} \frac{T^{h}_{afd,i} - T_i}{T\_idle\_end^{h}_{i} - T_i}\Big)$$
$$\sum_{i=1}^{n} t^{nh}_{idle,i} = R \times C \times \log\Big(\prod_{i=1}^{n} \frac{T^{nh}_{afd,i} - T_i}{T\_idle\_end^{nh}_{i} - T_i}\Big) \tag{4.11}$$

With the knowledge that $T^{nh}_{afd,1} = T^{h}_{afd,1}$, the idle time difference is computed as

$$\begin{aligned}
&\sum_{i=1}^{n} t^{nh}_{idle,i} - \sum_{i=1}^{n} t^{h}_{idle,i} = \\
&R \times C \times \log\left(\prod_{i=1}^{n} \frac{T^{nh}_{afd,i} - T_i}{T_idle_end^{nh}_{i} - T_i} \times \prod_{i=1}^{n} \frac{T_idle_end^{h}_{i} - T_i}{T^{h}_{afd,i} - T_i} \right) \\
\Leftrightarrow\ &\sum_{i=1}^{n} t^{nh}_{idle,i} - \sum_{i=1}^{n} t^{h}_{idle,i} = \\
&R \times C \times \log\left(\frac{T^{nh}_{afd,1} - T_i}{T^{h}_{afd,1} - T_i} \times \right. \\
&\left. \left(\prod_{i=2}^{n} \frac{T^{nh}_{afd,i} - T_i}{T^{h}_{afd,i} - T_i} \times \frac{T_idle_end^{h}_{i-1} - T_i}{T_idle_end^{nh}_{i-1} - T_i} \right) \times \frac{T_idle_end^{h}_{n} - T_i}{T_idle_end^{nh}_{n} - T_i} \right) \\
\Leftrightarrow\ &\sum_{i=1}^{n} t^{nh}_{idle,i} - \sum_{i=1}^{n} t^{h}_{idle,i} = \\
&R \times C \times \log\left(\left(\prod_{i=2}^{n} \frac{T^{nh}_{afd,i} - T_i}{T^{h}_{afd,i} - T_i} \times \frac{T_idle_end^{h}_{i-1} - T_i}{T_idle_end^{nh}_{i-1} - T_i} \right) \times \right. \\
&\left. \frac{T_idle_end^{h}_{n} - T_i}{T_idle_end^{nh}_{n} - T_i} \right)
\end{aligned} \tag{4.12}$$

It is known that $T\_idle\_end^{h}_{n} \geq T\_idle\_end^{nh}_{n}$. From Eqn. 4.8 and the fact that $T^{nh}_{afd,i} > T\_idle\_end^{nh_{i-1}}$ and $T^{h}_{afd,i} > T\_idle\_end^{h_{i-1}}$, we can deduce that

$$\frac{T^{nh}_{afd,i} - T_i}{T^{h}_{afd,i} - T_i} \geq \frac{T\_idle\_end^{nh}_{i-1} - T_i}{T\_idle\_end^{h}_{i-1} - T_i} \tag{4.13}$$

From Eqn. 4.13 and Eqn. 4.12, we obtain the relation $\sum_{i=0}^{n} t^{h}_{idle,i} \leq \sum_{i=0}^{n} t^{nh}_{idle,i}$.

4.5 Experimental Results

In this section, we present four experimental results that support our claim that frame drops help in reducing inserted idle times in a quality and thermal aware multimedia processing setup. In our first experiment, we show that frame drops can be used to completely eliminate idle time insertions. The second experiment shows that the magnitude of total idle time insertion decreases with increase in

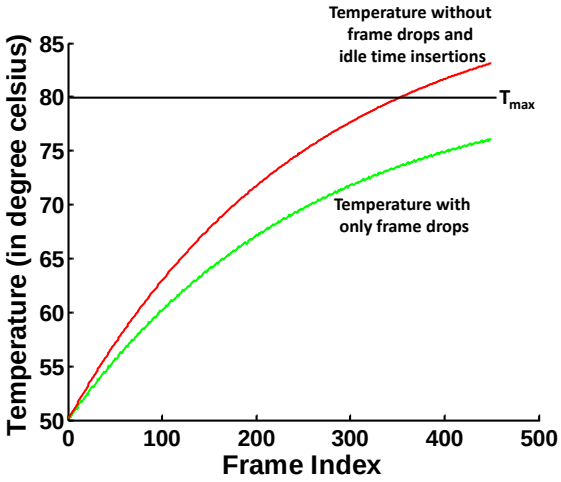


Figure 4.5: Temperature control without insertion of idle times

frame drops. The third experiment shows the reduction in end-to-end delay as a result of reduced idle times in the context of frame drops. Finally, we demonstrate how the introduced idle times are reduced by maintaining lightweight workload histories. The parameters that we use for the thermal model (*used in our experiments*) are from [81]. We set $C = 112.2mJoules/^{\circ}C$, $R = 1.83^{\circ}C/Watt$ and $T_p = 1.25\ ns$ as the parameter values for our experiments. We use a PE similar to the one used in [81]. The OC value used in the experiments is 3000 processor cycles. Additionally, we have $T_a = 90^{\circ}C$ and $T_i = 38^{\circ}C$ as the active and idle mode steady state temperatures respectively. The initial temperature we choose for our experiments is $T_{init} = 50^{\circ}C$.

The processor cycles for all the MPEG-2 tasks are obtained using the SimpleScalar simulator [8]. We obtain the processor cycles for a MIPS-like processor model using Portable Instruction Set Architecture (PISA). The tasks mapped to PE_1 are Variable Length Decoding (VLD), Inverse Quantization (IQ) and Inverse Discrete Cosine Transform (IDCT). Motion Compensation (MC) task is mapped to PE_2. We use 5 video clips (from [1]) in our experiments- *susi*_080, $time$_080, *cact*_080, $flwr$_080 and *mobl*_080. All the video clips except $time$_080 are motion videos, whereas $time$_080 is mostly a still clip. All the results discussed further are obtained for the first stage of the PE. The advantages observed are compounded in the second stage of the PE. Now we discuss our results in detail.

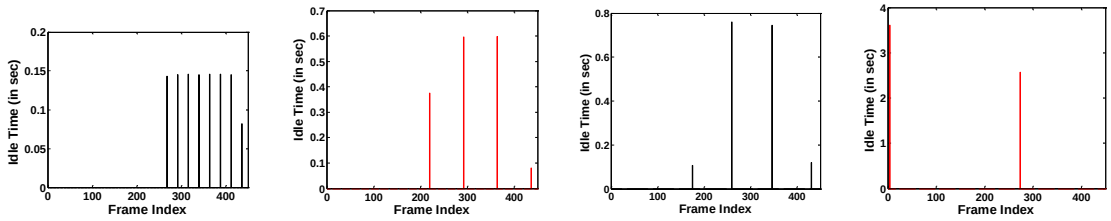


Figure 4.6: Idle times introduced with $T_{max} = 80°C$ for video clip (a) *susi*_080 at 30 dB, (b) *susi*_080 at 35 dB, (c) $flwr$_080 at 30 dB and (d) $flwr$_080 at 35 dB.

4.5.1 Elimination of idle times

We use $time$_080 video clip in the first experiment. Two sets of simulations are conducted using this clip. In the first, we apply our quality and thermal aware idle time insertion strategy to keep the temperature always below $T_{max} = 80°C$ and the quality for any interval above 35 dB. The maximum workload history used in these experiments is $HIST\_MAX = 24$. In the second simulation, we find the temperature profile for the same clip without introducing any frame drops and additional idle times. The results are shown in Fig. 4.5.

There are two curves corresponding to the two simulations in Fig. 4.5. The maximum temperature is marked using the horizontal dotted line at T_{max} value. It is observed that the application level technique of frame drops is sufficient to keep the temperature of PE_1 below T_{max}. Additional idle times were not required to control the temperature. On the other hand, it is also seen that without frame drops and idle time insertions, the temperature profile of PE_1 increases above T_{max}. This highlights the advantage that our application level technique of frame drops aids in completely avoiding idle times for certain video clips. Elimination of extra idle times has a direct effect of reducing the end-to-end delay at PE_1.

4.5.2 Reduction of idle times with quality

In this experiment, we show how the idle times inserted to control temperature are reduced with increasing number of frame drops or lower quality. Here, we show the results for the clips *susi*_080 and $flwr$_080, but the results are similar for all the other motion clips. In the case of $time$_080, there is no reduction in idle time because idle times are not required in the presence of frame

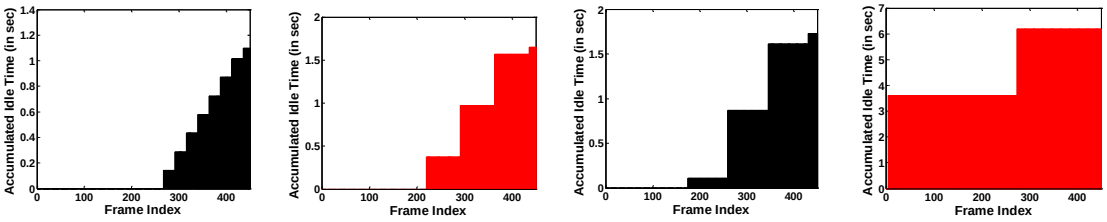


Figure 4.7: Accumulated idle times with $T_{max} = 80°C$ for video clip (a) *susi*_080 at 30 dB, (b) *susi*_080 at 35 dB, (c) $flwr$_080 at 30 dB and (d) $flwr$_080 at 35 dB.

drops as discussed earlier. We show the results for *susi*_080 with two quality values 30 dB and 35 dB in Fig. 4.6(a) and Fig. 4.6(b) respectively. Similarly, the results for $flwr$_080 are shown in Fig. 4.6(c) and Fig. 4.6(d) respectively. The $HIST\_MAX$ is fixed at 24. It is evident from the plots that the required idle time for $PSNR = 35$ dB is higher than that for $PSNR = 30$ dB even though the number of dropped frames decreases with increase in quality. The maximum idle time inserted with $PSNR = 35$ dB is approximately $1.5\times$ the maximum idle time inserted with $PSNR = 30$ dB for *susi*_080. Similarly, the idle time inserted with $PSNR = 35$ dB is approximately $3.5\times$ the maximum idle time inserted with $PSNR = 30$ dB for $flwr$_080. As higher target quality implies lesser number of possible frame drops, the inserted idle times are increased to adhere to the peak temperature constraint.

In order to see the effect of reduced idle times with frame drops on end-to-end delay, it is necessary to observe the accumulated idle times after the end-to-end delay of each frame. The accumulated idle times for *susi*_080 are shown with two quality values $PSNR = 30$ dB and $PSNR = 35$ dB in Fig. 4.7(a) and Fig. 4.7(b) respectively, whereas the same results for $flwr$_080 is shown in Fig. 4.7(c) and Fig. 4.7(d). The $HIST\_MAX$ is fixed at 24. As expected, it is observed from the plots that for each frame index, the accumulated idle times are greater for $PSNR = 35$ dB in comparison to those for $PSNR = 30$ dB. For *susi*_080, we see that the maximum accumulated idle time is 553 *msec* less for $PSNR = 30$ dB than the same for $PSNR = 35$ dB. In the case of $flwr$_080, the maximum accumulated idle time for $PSNR = 30$ dB is 4.45 *sec* less than the same for $PSNR = 35$ dB. This also gives us the idea that the end-to-end delay reduces with increase in number of frame drops. By extrapolating this observation, we can also deduce that without frame drops, if idle times are introduced at intervals larger than the frame drop intervals for $PSNR = 30$ dB, then the inserted idle

Table 4.1: PE_1 delay for benchmark video clips with varying quality and *HIST_MAX* values

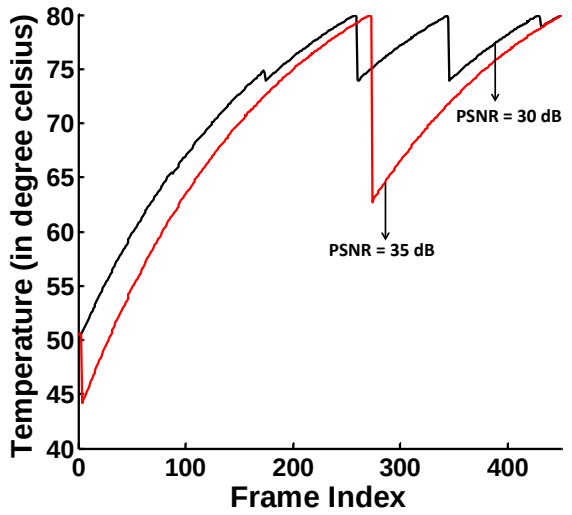
PE_1 Delay (in sec)	*HIST_MAX*	0	5	10	15	20	25
clip	PSNR (in dB)						
*susi*_080	30	1.1108	1.0997	1.0993	1.099	1.0975	1.0975
	32	1.2853	1.2793	1.279	1.2788	1.2773	1.2773
	35	1.6594	1.654	1.6528	1.6524	1.6508	1.6508
*cact*_080	30	1.7342	1.728	1.7275	1.7271	1.7265	1.7261
	32	2.6078	2.6016	2.6013	2.6012	2.5987	2.5985
	35	x	x	x	x	x	x
*flwr*_080	30	1.7372	1.7372	1.7313	1.7309	1.7309	1.7309
	32	2.2741	2.2684	2.2605	2.2596	2.2594	2.2594
	35	6.3119	6.1816	6.1816	6.1816	6.1816	6.1816
*mobl*_080	30	1.9024	1.8944	1.8865	1.8862	1.8862	1.8862
	32	3.4996	3.4929	3.4881	3.4881	3.4881	3.4873
	35	x	x	x	x	x	x

times and hence delay will be higher.

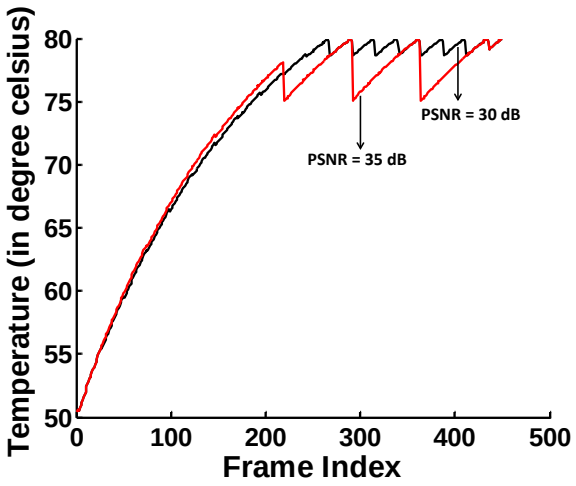
4.5.3 Reduction in delay with varying quality and *HIST_MAX* values

In this experiment, we present the trade-off between delay at PE_1 and quality under thermal constraints. The *HIST_MAX* value is also varied across 6 different values 0, 5, 10, 15, 20 and 25. As shown in the previous result, the idle times inserted are reduced with increasing frame drops. This leads us to believe that the PE_1 delay can also be reduced in a quality aware manner under thermal constraints. Here, we look at PE_1 delay which is one of the major factors affecting the end-to-end delay. The end-to-end delay value is obtained by measuring the time difference between the frame appearance at output and input of the platform. The characteristics of PE_1 delay are followed by the end-to-end delay. Therefore, a reduction in PE_1 delay also reduces the end-to-end delay. We show the results for three quality values - $PSNR = 30\ dB$, $PSNR = 32\ dB$ and $PSNR = 35\ dB$. We conduct this experiment for benchmark video clips - *susi*_080, *cact*_080, *flwr*_080 and *mobl*_080. The maximum allowed temperature is set to $T_{max} = 80°C$. The PE_1 delay values obtained for different quality values are shown in Table 4.1. The delay values corresponding to $PSNR = 35\ dB$ is given as x for video clips *cact*_080 and *mobl*_080 because there cannot be frame drops in these clips for this target quality.

From Table 4.1, it is clear that there is a considerable reduction in the PE_1 delay with a small



(a)



(b)

Figure 4.8: Temperature profile (with frame drops and idle time insertions) for (a) $flwr\_080$ with $T_{max} = 80°C$ and target quality of $30dB$ and $35dB$ for video clips (a) $flwr\_080$ and (b) $susi\_080$.

decrease in quality ($PSNR = 32\ dB$ to $PSNR = 30\ dB$). This reduction for a $HIST\_MAX = 25$ value is very prominent in $mobl\_080$ (1.6 sec less), $flwr\_080$ (0.5285 sec less) and $cact\_080$ (0.8724 sec less). There is a small reduction in $susi\_080$ also. However, this reduction in PE_1 delay is even more prominent when the quality decreases from $PSNR = 35\ dB$ to $PSNR = 32\ dB$, i.e., for other setting remaining the same, $flwr\_080$ shows a delay reduction of approximately 3.9 sec. Some video clips exhibit larger reduction in PE_1 delay when compared to others because in videos with large variation, the number of frames that can be dropped for a target quality value is lesser when

compared to videos with lesser variation. Therefore, the intervals between successive frame drops is larger in the former. This causes higher idle times to be inserted to control temperature and hence higher PE_1 delay.

We now discuss the effect of using workload history ($HIST\_MAX$) on the PE_1 delay values. It is observed from Table 4.1 that with a $HIST\_MAX = 25$ (which amounts to an insignificant overhead of 3000 processor cycles), it is possible to reduce the PE_1 in all the video clips in comparison a scenario without workload history ($HIST\_MAX = 0$). We observe a maximum PE_1 delay reduction of 16 *msec* as the $HIST\_MAX$ is increased from 0 to 25 for the video clip *mobl*_080. It can be attributed to the large variation in $\gamma^u(k)$ for an interval of k frames across the entire clip. This variation can be lowered by maintaining workload history values and dynamically updating $\gamma^u(k)$ online.

We also present the thermal profile of the video clips $flwr$_080 and *susi*_080 for two quality values with $T_{max} = 80°C$ (Fig. 4.8). The $HIST\_MAX$ value was fixed at 24. It is observed that the temperature falls by a larger amount when target quality is $PSNR = 35\ dB$ when compared to $PSNR = 30\ dB$ in both the video clips. This is explained by our previous result that the idle times inserted for higher quality is higher and hence the temperature falls steeply. In addition to that, we clearly observe that the idle times introduced in video clip $flwr$_080 is higher than in *susi*_080 as we see larger falls in temperature in $flwr$_080.

4.6 Summary

In this work, we present a combined offline and online approach to process multimedia streams in a quality and thermal aware manner. We utilize an application level technique of frame drops to enable the insertion of smaller idle times. The frames are dropped under target quality constraints. The reduced idle times are inserted under quality and peak temperature constraints. The inserted idle times are made more tighter by recording a workload history. This workload history is used online and in conjunction with the workload curve obtained offline to estimate the worst-case workload that would be encountered in the future. Therefore, the pessimism in worst-case workload was reduced resulting in the reduction of idle times introduced. Our experiments validate the claim that

inserted idle times and end-to-end delays can be reduced with small quality reductions. From our benchmark video clips, we obtain a maximum reduction of 1.6 *sec* in PE_1 delay for a small quality reduction of 2 dB. Moreover, it has also been observed that for still video clips with low scene variations, it is possible to completely eliminate idle times with acceptable quality losses.

In this chapter, we presented a scheme to insert idle times along with bounded frame drops in a quality and thermal-aware manner to adhere to a peak temperature constraint. We drop frames such that it does not violate the quality constraint for any time interval. It would be an interesting future work to derive the desirable frame drops that minimize the idle times introduced under quality and thermal constraints.

Chapter 5

Fast Simulation Frameworks for Multimedia MPSoC platforms

In this chapter, we present two simulation frameworks to perform fast performance analysis for multimedia MPSoC platforms. In the first work, we use analytical models for the multimedia task workloads to estimate the required processing workloads for the incoming streams. This is then used to classify the incoming video streams to derive representative workload sets, which are used to reduce the simulation time by avoiding the simulation of all the video clips in the library.

In the second work, we introduce a hybrid simulation framework in order to accurately predict the mutlimedia task workloads. The accuracy in prediction helps in computing the data drops at various stages of the architecture for various system parameter configurations.

The performance analysis techniques described in earlier chapters benefit from the fast simulation techniques proposed in this chapter. These simulation techniques are used to either rapidly find the representative test clips, which would further speed up the analytical or simulation based performance analysis techniques to analyze the required system resources or to rapidly obtain the trace data that will be used by the proposed performance analysis techniques.

5.1 Model-Based Performance Analysis

Simulation-based system performance analysis is a very widely adopted methodology for multimedia-MPSoC platforms. In the context of a video processing application such as an MPEG-2 decoder, these simulations take a library of test video clips as input. When simulated with this library, the MPSoC platform is considered to be appropriately designed if it behaves in accordance to all the performance constraints. It is analogous to the common software functional testing methodology. However, unlike in the software testing scenario, the simulation of MPEG-2 decoder application with the library of video clips is very expensive with respect to time. As mentioned in an earlier work [10], it may take tens of hours for the simulation of only a few minutes of video in a decoding application. This is mainly due to the heterogeneous and complex nature of multimedia MPSoC architectures like the Eclipse template from Philips [6] and the Viper SoC architecture [5]. Therefore, the performance analysis time for such architectures steeply increases with the input library size.

In order to reduce the performance analysis time, there have been many efforts in the past [82–84] to identify representative test inputs. They classify the test inputs into well defined subsets with minimum correlation. However, many of these works were in the area of microprocessor design and the test input characteristics used for classification were instructions per cycle (IPC), cache miss rates, branch misprediction rates etc. A detailed description of the related work will be presented in the Section 5.1.1.

Performance analysis of multimedia-MPSoC platforms requires a completely different approach for deriving input characteristics towards the identification of representative workloads. A previous work [85] introduced a novel concept of VCCs where each video clip was represented using its VCC. This concept of VCCs was also suggested to be appropriate in [86] for identification of different application scenarios.

The intuition behind using VCC as the performance model is the hypothesis that video clips with similar VCCs would exhibit similar maximum buffer backlogs and maximum delays for one macroblock. However, in order to compute the VCCs, we first need to compute the workload values for each task. A straightforward way to compute these workload values uses time consuming simplescalar simulations. In this work, motivated by a workload model for MPEG-2 decoder tasks

presented in [87], we propose a fast model-based performance analysis method which integrates our workload model of the decoder tasks with a performance model (using VCCs) of the MPSoC architecture, thereby providing a fast and efficient clustering of the video clips. Here, simplescalar simulations to obtain workload values for each task is completely avoided and bitstream analysis (incorporating our MPEG-2 workload model) is used instead. Consequently system simulations can be run with only one video clip from each cluster, thereby considerably reducing the total simulation time. In addition, we also perform fine grained classification of video clips in each stage of the MPSoC architecture for a MPEG-2 decoder. This provides a way to identify the VCCs relevant to each stage of the architecture.

5.1.1 Related Work

The concept of representative workloads, in order to reduce the number of test inputs, has been comprehensively studied in the area of microprocessor design. Some of these have dealt with classifying program-input pairs based on microarchitecture dependent characteristics [82, 84]. The microarchitecture dependent program characteristics typically used were instructions per cycle (IPC), cache miss rates, branch misprediction rates and many other such characteristics. There has been some work performed to identify representative workloads based on microarchitecture independent characteristics such as register traffic, working-set size, data stream strides and instruction-level parallelism [83]. These are not instruction set architecture (ISA) or compiler independent. However, in the context of a multimedia MPSoC architecture, the characteristics used in the microprocessor domain do not capture the variabilities inherent in the test inputs (for example MPEG-2 video clips).

As there are many program input characteristics, they have been classified using Principal Component Analysis (PCA) in most of the earlier works. This reduces the correlation among the program inputs and thereby resulting in a smaller subset of inputs which have minimum correlation. Eeckhout et al. [82] suggest the need to select a representative workload for a target domain of a microprocessor. They mainly propose a method of selecting the benchmarks and input data per benchmark as representative workloads. Selecting a large number of them prohibitively increases the simulation time as they are constituted of many instructions. The authors used statistical analysis techniques like PCA and cluster analysis to extract representative workloads from the entire work-

load space. It was performed by measuring the similarity in behavior of the programs and finally establishing the fact that programs which are close in the workload space have similar behavior. To elaborate on the PCA method, the workloads are initially characterized in a s-dimensional space, where s represents the number of program characteristics that influence the performance. As s is too large and as there is some correlation among the s characteristics, the s-dimensional workload space is reduced to a p-dimensional space such that $p << s$.

PCA [88] is used to transform the s characteristics $X_1, X_2, \ldots, X_s$ into s principal components $Z_1, Z_2, \ldots, Z_s$ (which are linear combinations of the original variables such that the principal components are uncorrelated) such that

$$Z_i = \sum_{j=1}^{s} a_{ij} X_j \tag{5.1}$$

The transformation exhibits the following properties:

1. $Var[Z_1] > Var[Z_2] > \ldots > Var[Z_s]$, which signifies that the information content is the most in Z_1 and the least in Z_s.

2. $Cov[Z_i, Z_j] = 0, \forall i \neq j$, which signifies that principal components do not have any overlaps.

The total variance remains the same after the transformation, but some principal components have a large variance while some have a small variance. The ones which have smaller variances can be eliminated without much loss of information. This reduces the workload space into a p-dimensional space with p principal components. In this p-dimensional space, it is seen that different benchmarks will be far away from each other while the inputs from a benchmark are clustered together. Strong clustering indicates that one or a few inputs can be used to represent the cluster, while weak clustering might require the selection of many inputs. This concept led to our intuition that video clips with similar VCCs and clustered together will exhibit similar performance characteristics.

Cluster analysis is a method to group n program-input pairs depending on the values of s workload characteristics. This hierarchical clustering algorithm starts by considering each program-input pair as one cluster. It also has a $n \times n$ matrix of the distances at which each program-input pair is located with respect to the other. Each iteration groups two clusters having the shortest linkage

distance into a new cluster. This continues until one cluster remains in the end with all the program-input pairs. Different distance measurements are used in the literature. A dendrogram is used to graphically represent the linkage distance between two clusters that are grouped together in one iteration. Another clustering method used is k-means clustering [89].

John et al. [84] propose the characterization of workloads based on application's intrinsic properties like memory access behavior, locality, control flow behavior, instruction level parallelism, etc., which helps in the formulation of a program behavior model. This can then be used in conjunction with a processor model for analytical performance modeling. A study of memory reference locality using some generic metrics was also proposed. The measures used were the inter-reference temporal density function and the inter-reference spatial density function. The inter-reference temporal density function $f^T(x)$ is the probability of having x unique references between successive references to the same item. Similarly, the inter-reference spatial density function $f^S(x)$ is the probability of reference to a location x units away between references to the location of origin. According to the reasons already mentioned, multimedia workload characterization using properties like memory access behavior, locality, control flow behavior, instruction level parallelism etc. will not work well for multimedia MPSoC performance analysis as they do not capture the burstiness and variability in multimedia workloads.

Characterization of video stream inputs is somewhat different from the workload characterization in the microprocessor domain. It needs a platform independent approach to identify scenarios across the media streams. Hamers et al. [90] use such a method for resource prediction in media stream applications. The approach proposes to use macroblock profiling to group frames with identical decode complexity from various streams into scenarios. The resources that were predicted for evaluation were the decode time, quality of service and energy consumption. However, extraction of these parameters to group frames takes more time than our method which groups video clips using VCCs.

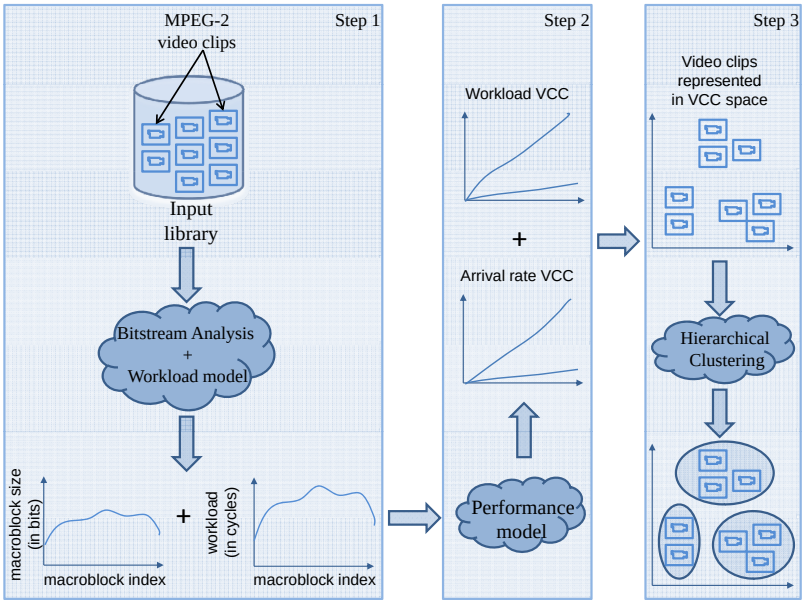


Figure 5.1: Overview of video stream classification using bitstream analysis

5.1.2 Overview of our framework

A schematic overview of our performance analysis framework is shown in Figure 5.1. Given a library of video clips, we perform the following steps in order to classify them into clusters

1. We perform bitstream analysis of each clip in accordance with a workload model. This gives us the execution cycle requirements of each decoder task in an MPEG-2 decoder. Here, we also extract the arrival rate of the video streams. The arrival rate of video streams can be easily obtained once the number of bits per macroblock of the stream (from bitstream analysis) and constant bit rate of the stream are known. The curves of these quantities are shown in Step 1 of Figure 5.1.

2. The two parameters extracted in Step 1 are then used to derive the corresponding VCCs in accordance to our performance model.

3. These VCCs are first used to transform the video clips into the VCC space. Then a hierarchical clustering of the video clips is performed based on a distance measurement between the clips in the VCC space. As a result, it is then possible to use one video clip from each cluster and perform simulations. The system designer can control the number of required clusters.

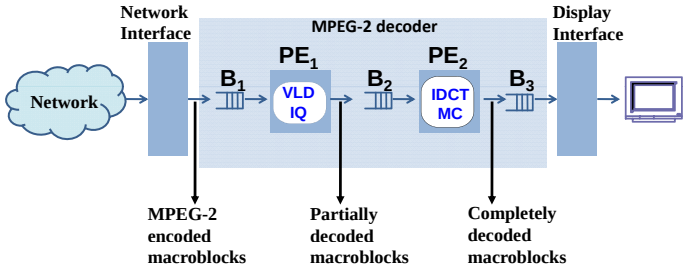


Figure 5.2: MPSoC platform architecture for MPEG-2 decoder

The MPSoC platform architecture used for a case study of the MPEG-2 decoder application consists of multiple interconnected processing elements (PEs) as shown in Figure 5.2. The tasks are split and efficiently allocated to the PEs. The PEs communicate by passing data units or stream objects between them. PE_1 and PE_2 are the two programmable processors. It also consists of the input/network and output interface. After mapping the MPEG-2 decoder application onto the MPSoC platform, PE_1 performs the Variable Length Decoding (VLD) and Inverse Quantization (IQ) tasks, while PE_2 performs the Inverse Discrete Cosine Transform (IDCT) and Motion Compensation (MC) tasks. The stream objects on which the PEs operate are macroblocks (MBs). Partially decoded MBs are sent from PE_1 to PE_2 through buffer B_2 while fully processed MBs are sent out of PE_2 to the output interface through buffer B_3.

5.1.3 Variability Characterization Curves

The hypothesis of this work is that video clips with similar VCCs cluster together in the VCC space and exhibit similar performance characteristics namely worst case buffer backlog and worst case delay for one MB. There is a strong indication that this is true because VCCs accurately characterize the data-dependent variability in the (i) execution times and (ii) input-output rates of the multimedia processing tasks. This process of quantitatively modeling the input stream variability constitutes our performance model. The burstiness in the arrival of streams can also be characterized using this method. These factors collectively contribute to the values of the performance characteristics.

VCCs specify the best and the worst case quantities of the variable characteristic with respect to an input parameter. It can be sequences of consecutive executions of a task or sequences of consecutive

time intervals of some specified length. A VCC is composed of a tuple $[(v^u(k), v^l(k)]$, where k is the input parameter representing the length of a sequence. $v^l(k)$ represents the lower bound on some characteristic that holds for all subsequences of length k within some larger sequence. $v^u(k)$ is the corresponding upper bound. More specifically, if $P(n)$ denotes the measure of a property for the first n items in the sequence, then

$$v^l(k) \leq P(i+k) - P(i) \leq v^u(k) \ldots \forall i, k \geq 1 \tag{5.2}$$

Based on the above definition of VCC, a workload VCC $[\gamma^u(k), \gamma^l(k)]$ can be defined as execution requirement bounds for a task mapped onto a PE in terms of the number of processor cycles for any k consecutive MBs. In other words, if $W(k)$ represents the number of processor cycles required by a task for the first k MBs in the video stream, then we can define for any i

$$\begin{aligned} \gamma^u(k) &= \max_{\forall i}\{W(i+k) - W(i)\} \\ \gamma^l(k) &= \min_{\forall i}\{W(i+k) - W(i)\} \end{aligned} \tag{5.3}$$

Similarly, the consumption and production VCCs can be represented by $\kappa = [\kappa^u, \kappa^l]$ and $\pi = [\pi^u, \pi^l]$ respectively. $[\kappa^u(k), \kappa^l(k)]$ are the bounds on the number of activations of a task for any k consecutive stream objects. Likewise $[\pi^u(k), \pi^l(k)]$ are the bounds on the number of stream objects produced by k consecutive activations of a task. It is hypothesized that video streams having similar VCCs will have similar worst/best case behaviors (e.g.: maximum backlogs in buffers).

An important aspect of VCCs that is understood here and which works in its favour, is that it is a more realistic model that can be used for the estimation of the resource requirements on a platform. Let us analyze this property of the VCCs. Let us denote the maximum execution cycle requirement for the execution of a single MB as e_{max} and the minimum execution cycle requirement for a single MB on the same video clip as e_{min}. We can obtain the worst case execution time denoted by $k \times e_{max}$ and best case execution time $k \times e_{min}$ for k consecutive MBs by linear interpolation of the corresponding execution times for 1 MB. Further, let us denote the upper and lower workload VCCs of this task for k consecutive MBs to be $\gamma^u(k)$ and $\gamma^l(k)$, respectively. It can be proved from the definition of a VCC that, for k consecutive macroblocks in a video stream

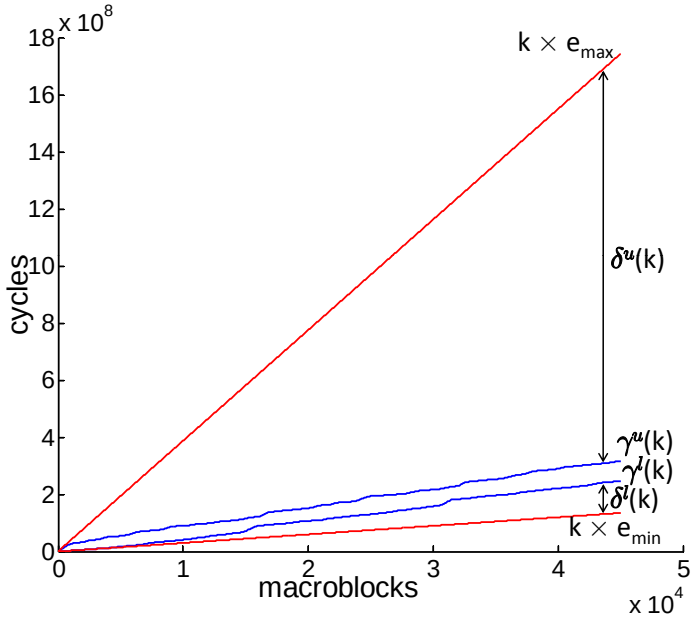


Figure 5.3: Differential errors $\delta^u(k)$ and $\delta^l(k)$ encountered when conservative linear interpolations $k \times e_{max}$ and $k \times e_{min}$ are used instead of Workload VCCs $\gamma^u(k)$ and $\gamma^l(k)$ respectively for VLD of k consecutive MBs

$$k \times e_{min} \leq \gamma^u(k) \leq \gamma^l(k) \leq k \times e_{max} \tag{5.4}$$

The above equation is shown graphically in Figure 5.3. The differences $\delta^u(k)$ and $\delta^l(k)$ shown in Figure 5.3 are defined as

$$\begin{aligned} \delta^u(k) &= k \times e_{max} - \gamma^u(k) \\ \delta^l(k) &= \gamma^l(k) - k \times e_{min} \end{aligned} \tag{5.5}$$

These differences show how much a worst case estimate and a best case estimate deviate from a more realistic estimation using VCCs. Hence, the performance model using VCCs does not take the extreme resource requirements for a task. At the same time, it does not under estimate the resource requirement for a task as is observed when the linear interpolation of the best case execution time is used. A more realistic estimate using VCCs makes sure that the MPSoC platform resources are well utilized. This is Step 2 in our performance analysis framework shown in Figure 5.1.

5.1.4 MPEG-2 Decoder Workload Model

The major tasks involved in MPEG-2 decoding are VLD, IDCT and MC. The computational workload required for other tasks such as IQ is negligible. The MPEG-2 decoder workload model depicts the computational workload (at MB granularity) required for each of the major tasks in MPEG-2 decoding. The workload model was developed for a RISC processor (similar to a MIPS3000) without any MPEG specific instructions. The MPEG-2 decoder application used for simulations was Test Model 5 (TM5) [91]. The simulations here refer to the simulations required for one time development of the workload model of the decoder tasks that are mapped onto the MPSoC platform and not simulations to obtain task workload values of every new video clip added to the input library. This workload model is employed in Step 1 of our performance analysis framework shown in Figure 5.1 to extract workload and arrival rate information.

5.1.4.1 VLD Task

It was experimentally found that the processor workload depends on the length of the Huffman codes which implied that the workload for VLD depended on the number of non-zero IDCT coefficients. The simulations showed this relation and in fact established that it was a linear relationship. Hence, the processor workload for the VLD task at MB granularity is modeled as:

$$Workload_{vld} = a \times n_{coeff} + b \tag{5.6}$$

where $Workload_{vld}$ is the estimated number of processor cycles for VLD decoding of the MB, n_{coeff} is the number of non-zero coefficients in the MB and a and b are constants that depend mainly on the processor architecture. This straight line fitting for VLD workload is supported by a plot of number of processor cycles required (from simplescalar simulation) versus the number of non-zero coefficients obtained for a video clip. This is shown in Figure 5.4.

From simulations, the values of a and b for the above mentioned processor were fixed at 140 and

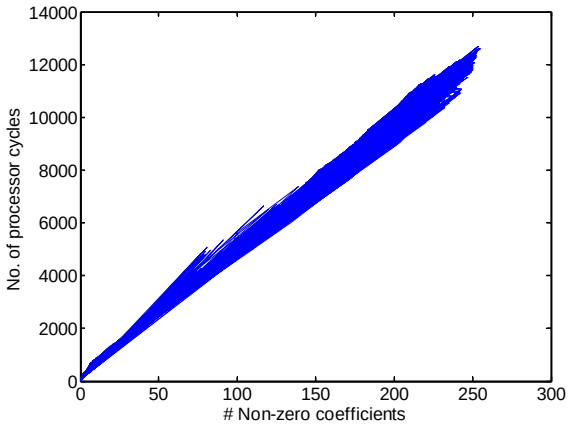


Figure 5.4: Workload versus number of non-zero coefficients for VLD task from simplescalar simulation of a video clip

3000. The VLD workloads obtained for 50 macroblocks using the workload model based on Equation(5.6) and simplescalar simulation using the *ffmpeg* open source decoder code are plotted in Figures 5.5(a) and 5.5(b). It is observed from the two graphs that although the VLD workload model was derived by instrumenting a different source code, both graphs are very similar, exhibiting identical characteristics for VLD processing. This demonstrates the validity of the VLD workload model.

5.1.4.2 MC Task

MC is another expensive task in MPEG-2 decoding. There are three types of MBs in MPEG-2 bitstream namely I-type (do not require motion compensation), P-type (require only forward motion compensation) and B-type (require both forward and backward motion compensation). Hence it was intuitively concluded that P-type MBs require half the number of processor cycles than B-type MBs while I-type MBs do not consume processor cycles for MC. However, this rough prediction does not suffice for MC. There are other parameters on which a MC function depends such as

1. Y component's (in YUV color space) x-dimension is HALF-PIXEL
2. Y component's y-dimension is HALF-PIXEL
3. U or V component's x-dimension is HALF-PIXEL

4. U or V component's y-dimension is HALF-PIXEL
5. forward or backward motion compensation is required
6. the motion compensation window size is 16x8 or 16x16

Depending on the MB, the MC routine is called with different parameters, each requiring different workloads. A look-up table (LUT) was built with 64 values of processor workloads as there are 6 parameters. The MC routine may be called different number of times by each MB, which was also taken into consideration. The MC workloads obtained for 50 macroblocks employing the MC function described above and simplescalar simulation using the *ffmpeg* open source decoder code are plotted in Figures 5.6(a) and 5.6(b). It is observed from the two graphs that both graphs are very similar exhibiting identical characteristics for MC processing. This demonstrates the validity of the MC functional model.

5.1.4.3 IDCT Task

We estimate the IDCT workload requirement for each MB in a video clip based on the position of the IDCT coefficients in the 8x8 block structure in the MB. The MPEG-2 stream that was used to run the experiments had the 4:1:1 chroma format. This implies that each MB had 6 blocks with 64 IDCT coefficients each. The workload requirements for these MBs varies with two types of frame formats namely Intra-Frames and Inter-Frames. The number of non zero IDCT coefficients in significant positions of the 8x8 block were extracted and then used to estimate the workload requirement for each block. Here, significant positions are those positions which are the main contributors to the workload values in the IDCT task. Let the number of non zero IDCT coefficients in the significant positions be n_{idct}. This value can be negative if the number of zero IDCT coefficients in certain positions exceed the number of non zero IDCT coefficients in other significant positions. Then the IDCT workload estimate for each MB can be calculated as

$$Workload_{idct} = W_{basis} + \alpha \times n_{idct} \tag{5.7}$$

where W_{basis} is the base workload value that is the minimum required workload if there is atleast one

non zero IDCT coefficient in a significant position. It varies depending on whether the frame type is Intra-Frame or Inter-Frame, the values being 10782 for Intra-Frame MBs and a linear combination of the values 374, 1863 and 1981 for Inter-Frame MBs. The value of α has been found to be 118. Hence, we did not require a LUT.

The IDCT workloads obtained for 50 macroblocks using the workload model based on Equation(5.7) and simplescalar simulation using the *ffmpeg* open source decoder code are plotted in Figures 5.7(a) and 5.7(b). It is observed from the two graphs that the IDCT workload model exhibits similar workload requirements as obtained using simulation. This demonstrates the validity of the IDCT workload model.

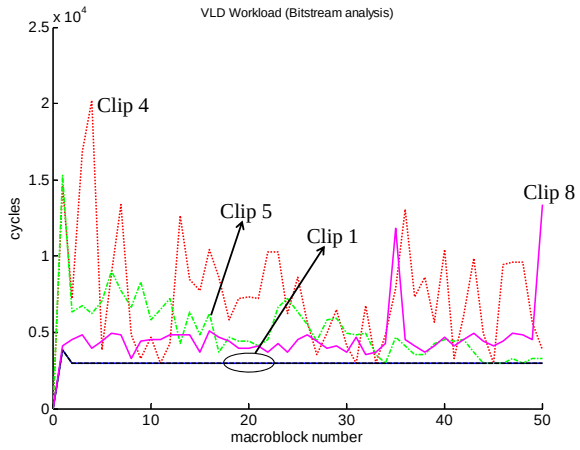
Earlier works on IDCT workload modeling were performed for workload-scalable transcoding as in [92]. The authors use a look up table to predict the workload values for IDCT based on whether the frame type is Inter-Frame or Intra-Frame and also based on the position of the most important non-zero IDCT coefficient. As they considered skipped frames also, they required a 3x64 LUT to predict the workload value. In [93], the number of significant non-zero IDCT coefficients is decided by an energy threshold.

5.1.4.4 Total Workload

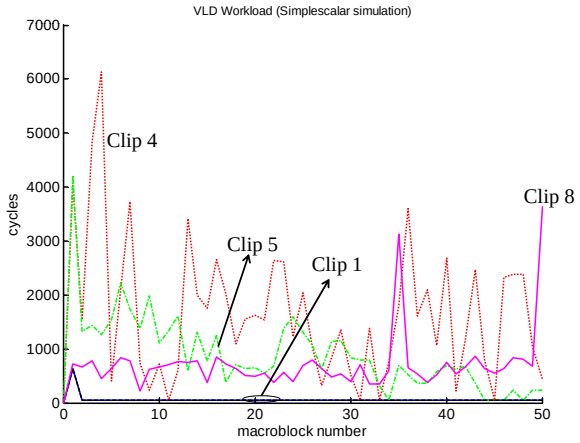
The total workload for MPEG-2 decoding can therefore be obtained by adding up the values predicted for the VLD, MC and IDCT tasks. These workload values are now used to generate the VCCs at the various stages in the architecture.

5.1.5 Test Case Classification

We utilize the bitstream analysis method incorporating the workload model described in Section 5.1.4. In addition to the workload values, we also extracted macroblock sizes (in bits) of the encoded bitstream (in order to obtain arrival rate information) by just parsing through the frame structure of the video clips. The VCCs obtained from these two quantities are used to perform classification of the MPEG-2 clips shown in Table 5.1. This turns out to be a faster design methodology from a system



(a)



(b)

Figure 5.5: Workload values for different tasks for 50 macroblocks of 5 video clips from Table 5.1: (a) VLD workload using bitstream analysis, (b) VLD workload using simplescalar simulation.

designer's perspective compared to simplescalar simulation as the time required for classification using bitstream analysis is much lower. The metrics used for performance analysis of the MPSoC architecture are worst case buffer backlog and worst case delay for one MB.

To classify two streams based on a single variability, a dissimilarity measure is used. The dissimilarity between two VCCs for each of the points $k = 1, 2, \ldots n$ is found using the City Block metric [94]. The pairwise dissimilarity between two streams i and j, with respect to a VCC of type r, is then

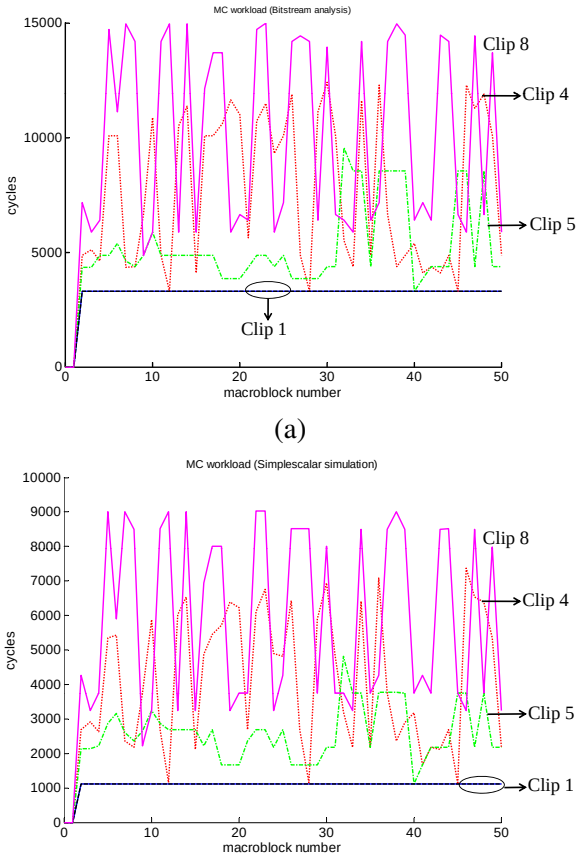
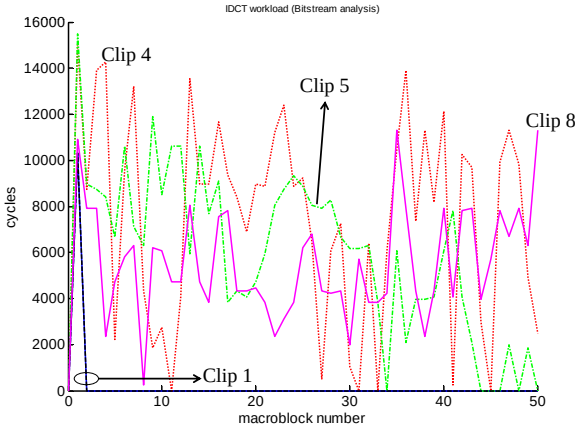


Figure 5.6: Workload values for different tasks for 50 macroblocks of 5 video clips from Table 5.1: (a) MC workload using bitstream analysis, (b) MC workload using simplescalar simulation.

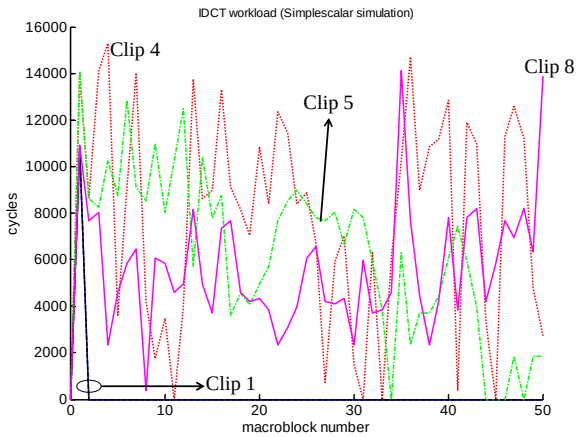
computed using

$$d_{rij} = \sum_{k=1}^{n} \omega_r(k) \left| \Theta_{ri}(k) - \Theta_{rj}(k) \right| \tag{5.8}$$

where $\Theta_{ri}(k)$ represents a VCC of type r associated with the ith stream and $\omega_r(k) = 1/k$ are weights to normalize the differences $\left|\Theta_{ri}(k) - \Theta_{rj}(k)\right|$ over the length k of the analysis interval. With more VCCs, the pairwise dissimilarity between the streams for each VCC is calculated using Equation(5.8). This is combined to form the overall pairwise dissimilarity measure between two streams i and j with respect to VCCs of type $r = 1, 2, \ldots p$ as



(a)



(b)

Figure 5.7: Workload values for different tasks for 50 macroblocks of 5 video clips from Table 5.1: (a) IDCT workload using bitstream analysis and (b) IDCT workload using simplescalar simulation.

$$d_{ij} = \sum_{r=1}^{p} d_{rij} \tag{5.9}$$

The overall pairwise dissimilarity measure is obtained by giving equal weightage for each VCC. The complete linkage algorithm is used to classify the streams based on the dissimilarity measure computed in Equation(5.9). This is Step 3 of our performance analysis framework shown in Figure 5.1. A dendrogram of the hierarchical cluster tree is then obtained as a result of the classification. Next, we discuss the experimental framework that is used to validate the claim that the bitstream

Table 5.1: MPEG-2 video clips used in our experiments [ftp://ftp.tek.com/tv/test/streams/Element/MPEG-Video/]

clip number	video clip	clip number	video clip
1	100b_080.m2v	7	pulb_080.m2v
2	bbc3_080.m2v	8	susi_080.m2v
3	cact_080.m2v	9	tens_080.m2v
4	flwr_080.m2v	10	time_080.m2v
5	mobl_080.m2v	11	v700_080.m2v
6	mulb_080.m2v		

analysis approach actually results in proper identification of representative workloads for a MPSoC platform.

5.1.5.1 Experimental Framework

Here, the concepts discussed in Sections 5.1.3, 5.1.4 and 5.1.5 are integrated and applied to the different stages of the multiprocessor architecture as shown in Figure 5.2.

The video stream is first parsed to extract the required characteristics, namely workload requirement per macroblock and bit sizes of each macroblock. For this, we use TM5 as our decoder source code in order to implement the workload model for the VLD+IQ and IDCT+MC tasks. The code to compute the workload values of different task sets mapped to each PE was inserted into the appropriate modules of TM5. The bit sizes per macroblock are also computed by keeping track of the count of bits as the procedure for decoding a macroblock is entered. The executable is then run for each of the clips used in the test set. It is interesting to note here that a certain group of clips exhibits higher variation in output workload values in comparison to other groups. A similar observation was also made for the number of bits per macroblock. This led us to the intuition that VCC curves obtained from these values can be used to classify the videos as it characterizes the bursty nature of video data and the accompanying variation in the workload requirements.

Once the bitstream analysis is performed, the next task in the process of classification is the generation of the VCCs. In this step, we produce the workload VCCs as described in Section 5.1.3, but there is a variation in the idea of what VCC curves to generate. As we already obtained the workload values for the tasks VLD+IQ and IDCT+MC by bitstream analysis, we generate separate workload VCCs for these sets of tasks denoted by $[\gamma^u_{vld}, \gamma^l_{vld}]$ and $[\gamma^u_{idct}, \gamma^l_{idct}]$, respectively. In addition

to these, we also obtain a VCC from the bits per macroblock statistics. As the input bit rate of the video clips is constant at 8 Mbps, we compute the input arrival rate of each macroblock, which is then used for the generation of the macroblock arrival rate VCC denoted by $[\kappa^u_{vld}, \kappa^l_{vld}]$. In this setup, we perform classification at three stages of the MPSoC architecture depicted in Figure 5.2.

Input Stage: Firstly, classification of the video streams is performed at the input stage of the architecture which consists of the input processing element PE_1 and the input buffer B_1. The VCCs $[\gamma^u_{vld}, \gamma^l_{vld}]$ and $[\kappa^u_{vld}, \kappa^l_{vld}]$ are used for the input stage classification as they are the decisive parameters that control the input side architecture with the former characterizing the workload variability possible in PE_1 and the latter signifying the degree of input burstiness that PE_1 can experience. The trace of γ^u_{vld} and γ^l_{vld} curves for the earlier listed 8 Mbps video clips are shown in Figures $5.8(a)$ and $5.8(b)$, respectively. Similarly the κ^u_{vld} and κ^l_{vld} curves are shown in Figures $5.8(c)$ and $5.8(d)$, respectively. The dendrogram of the cluster tree obtained as a result of the classification exercise performed at the input stage is shown in Figure $5.9(a)$.

Intermediate Stage: The second stage of classification takes both the processing elements PE_1 and PE_2 into consideration along with the intermediate buffer B_2. The three parameters that determine the classification at this stage are the two workload VCCs $[\gamma^u_{vld}, \gamma^l_{vld}]$, $[\gamma^u_{idct}, \gamma^l_{idct}]$ and the arrival rate VCC of the VLD decoded frames represented by $[\kappa^u_{vld}, \kappa^l_{vld}]$. The traces of the γ^u_{idct} and γ^l_{idct} curves are shown in Figures $5.8(e)$ and $5.8(f)$, respectively. The dendrogram of the cluster tree obtained as a result of the classification exercise performed at the intermediate stage is shown in Figure $5.9(b)$.

Playout Stage: The third stage of classification is performed at the playout stage of the MPEG-2 decoder MPSoC architecture. As is evident now from the architecture, the parameter playing the sole role in architecture specification at this stage is the $[\gamma^u_{idct}, \gamma^l_{idct}]$ VCC.

Observations: It is clearly evident from the obtained VCC curves and dendrograms that the VCCs obtained as a result of bitstream analysis of the MPEG-2 clips provide the basic clustering into motion and still videos. The classification is specific to the different stages in the architecture which

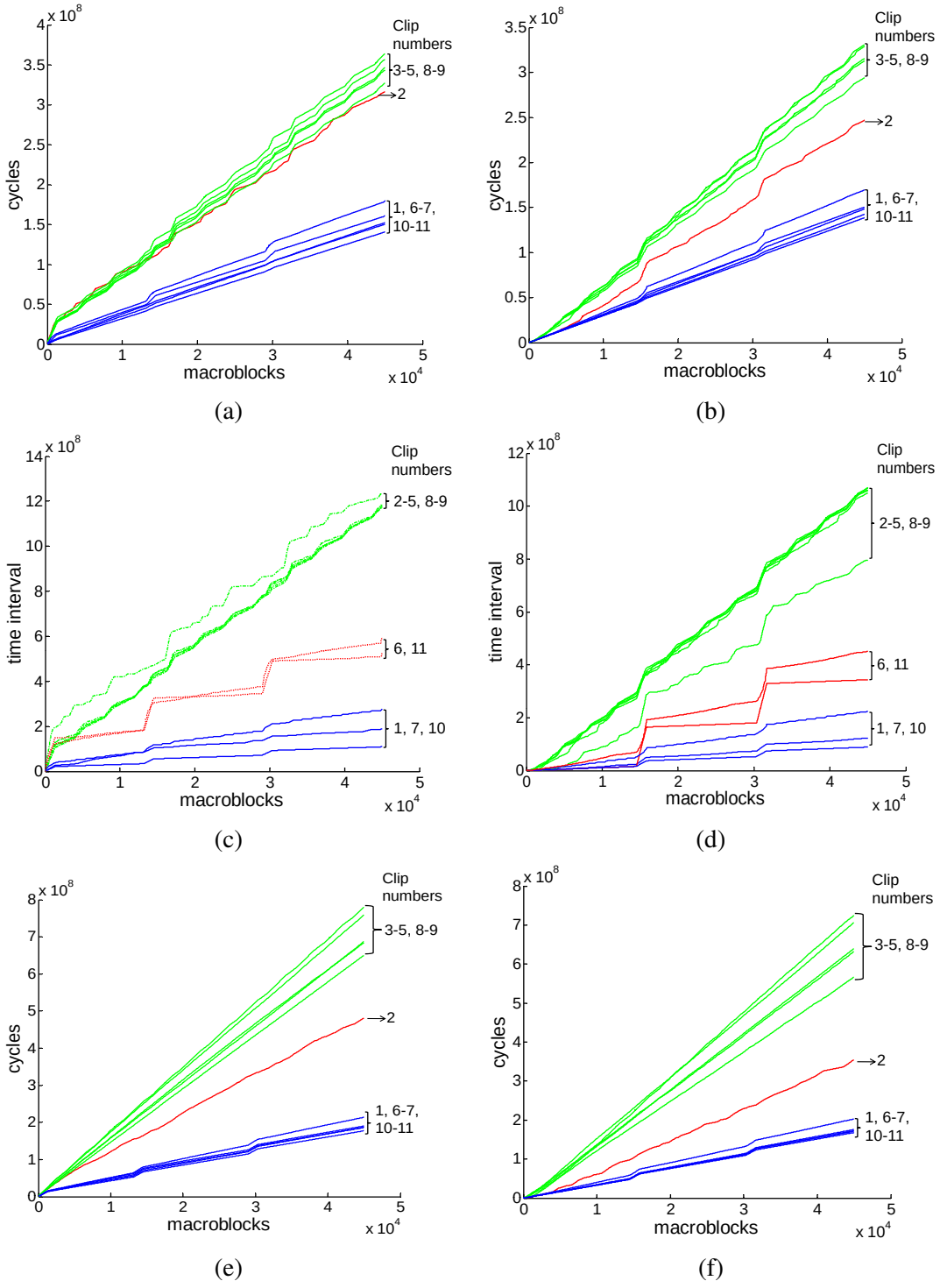
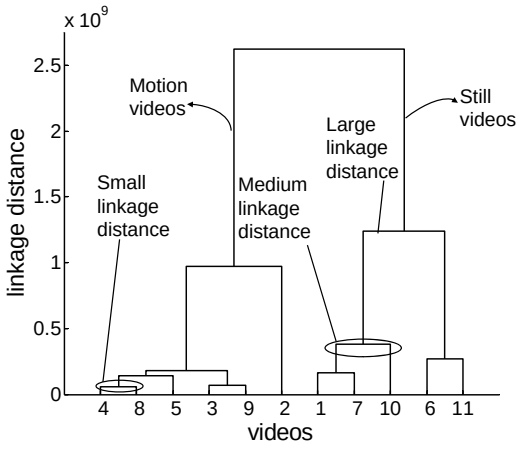
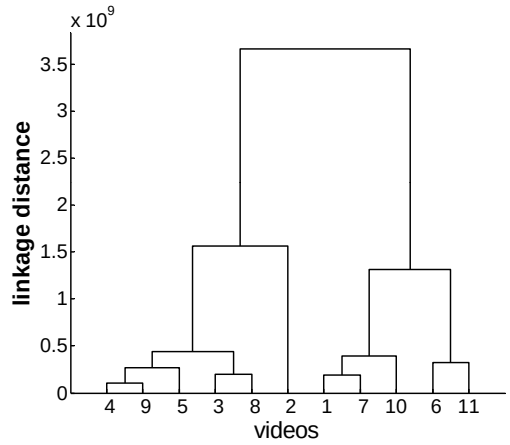


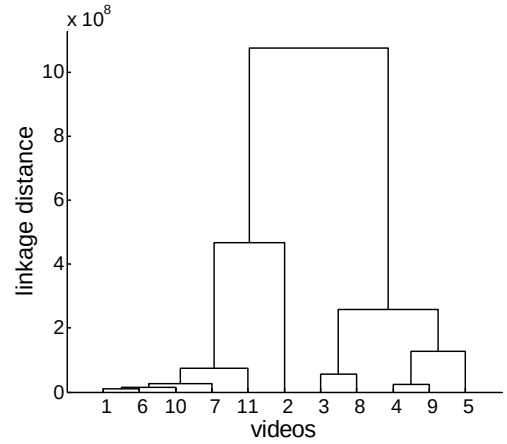
Figure 5.8: Variability characteristic curves for 11 video clips (each cluster is marked with the clip numbers of videos from Table 5.1) used for classification: (a) VLD Upper workload curve (γ^u_{vld}), (b) VLD Lower workload curve (γ^l_{vld}), (c) Upper arrival rate curve to $PE1$ (κ^u_{vld}), (d) Lower arrival rate curve to $PE1$ (κ^l_{vld}), (e) IDCT+MC Upper workload curve (γ^u_{idct}) and (f) IDCT+MC Lower workload curve (γ^l_{idct}).



(a)



(b)



(c)

Figure 5.9: Cluster trees of video clips at the various stages of the architecture (a) Input (b) Intermediate and (c) Playout

is a more fine grained approach than performing it for the entire architecture. This gives a more accurate classification of the video clips as different combinations of VCCs play a decisive role in the determination of the architecture specifications at various stages. Next we discuss the setup and procedure to validate the claim that bitstream analysis based generation of VCCs actually aids in classification of workloads.

5.1.6 Validation

The integral architectural parameters of the MPSoC platform shown in Figure 5.2 are the processor frequencies and the sizes of various buffers, namely the input buffer, the intermediate buffer and the playout buffer. In the current step, we fix a particular frequency pair corresponding to the two PEs. This selection is currently not based on any analytical framework as we are not concerned about any playout buffer underruns in this experiment. Here, we are more concerned about the various buffer occupancies and try to establish the claim that similar videos that are nearer to each other in the cluster trees shown in Figures $5.9(a)$, $5.9(b)$ and $5.9(c)$ also exhibit similar buffer occupancies. This claim can be emphasized even more by showing that the pair of video clips which are closer than others exhibit less difference in their maximum buffer occupancies than other pairs. This provides strong evidence for the validity of the bitstream based classification of video clips. The maximum buffer size required for each video clip is computed using the equation

$$\begin{aligned} Buf_i &= (Buf_{i-1}+1)\ldots\forall i\left(\tau_{arr_i} < \tau_{proc_{i-1}}\right) \\ Buf_i &= (Buf_{i-1}-1)\ldots\forall i\left(\tau_{arr_i} > \tau_{proc_{i-1}}\right) \\ Buf_i &= Buf_{i-1}\ldots\ldots\ldots\forall i\left(\tau_{arr_i} = \tau_{proc_{i-1}}\right) \\ Bufferbacklog &= \max_{\forall i}\left(Buf_i\right) \end{aligned} \tag{5.10}$$

where $i = 2,3,\ldots,N$ (N is the last macroblock number in the video stream) as τ_{proc_0} is not defined and τ_{proc_1} is the time instant at which the 1st MB is processed, Buf_i is the buffer backlog when the $(i)th$ macroblock is inserted into the system, $Buf_0 = 0$, τ_{arr_i} is the arrival time of the $(i)th$ macroblock, $\tau_{proc_{i-1}}$ is the time when $(i-1)th$ macroblock is processed completely and $Bufferbacklog$ is the maximum backlog in the input buffer. The interpretation of the above equation is straightforward. The buffer occupancy keeps increasing as new MBs enter the particular stage of the architec-

Table 5.2: Simulation results for maximum buffer backlogs (in number of MBs) at various stages in the architecture

videos	B_1	videos	B_2	videos	B_3
1	186886	1	339229	1	571560
4	247738	4	135798	4	472560
7	215236	7	374037	5	480480
8	259071	8	186373	6	571560
10	155674	9	167807	10	571560

ture and it reduces as they are completely processed by the PE and sent to the next stage. The worst case delay for one MB can be computed using the equation

$$Delay = \max_{\forall i} \left(\max \left(\tau_{proc_{i-1}}, \tau_{arr_i} \right) + \tau_{mbcyc_i} - \tau_{arr_i} \right) \tag{5.11}$$

where τ_{mbcyc_i} is the processor cycle time required for $(i)th$ macroblock. The expression of worst case delay for 1 MB given by Equation(5.11) takes the following two cases into consideration

1. All the previous MBs have been processed before or when the new MB arrives in which case the delay for the arriving MB is τ_{mbcyc_i}.

2. If previous MBs have still not been processed while a new MB arrives, then the processing of the new MB can start only after all the MBs ahead in the buffer are processed.

In order to check the above mentioned validity, we have simulated the multiprocessor architecture using a SystemC simulator with the workload cycles obtained from simplescalar simulation (*sim-safe* configuration). The PE_1 frequency was fixed at 40 MHz while the PE_2 frequency was fixed at 200 MHz. The results obtained are very much in support of the idea we started with and are presented in Table 5.2.

It is immediately observed from the results that the motion and still videos that form separate clusters also give similar buffer occupancies in their respective clusters. However, more importantly we can observe that some pairs of video clips which have smaller linkage distances in the cluster trees exhibit similar buffer occupancies. In the case of B_1, videos 4 and 8 have very similar maximum backlogs when compared to videos 1 and 7, the difference in maximum backlogs being 11333 and 28350, respectively. For B_2, videos 4 and 9 exhibit the most similar backlog difference (32009)

Table 5.3: Simulation results for maximum delay (in seconds) for one MB at each PE

videos	D_{PE1}	videos	D_{PE2}
1	0.848	1	3.535
4	10.639	4	7.849
7	0.7698	7	3.598
8	11.5	8	11.952
10	0.289	9	9.939

compared to videos 4 and 8 (50575) and videos 1 and 7 (34808). Videos 1 and 6 are more similar in the playout stage compared to 4 and 5 which is also evident from the cluster tree of the playout stage.

The similar worst case delays for one MB among video clips from the same cluster are also evident from the simulation results for the maximum delays for one macroblock shown in Table 5.3. In the case of $PE1$, it is seen that videos 1 and 7 have similar maximum delays while videos 4 and 8 are closer to each other in their maximum delays. It is also seen that video clip 10 is much closer in maximum delay to video clips 1 and 7 than video clips 4 and 8. This behavior is also seen in $PE2$.

5.2 Hybrid Simulation for Quality-Driven Performance Analysis

Multimedia decoders are widely used in state-of-the-art mobile devices. These devices are usually small in size and are designed with limited system resources (buffer capacity, processor's clock frequency, etc.) in order to adhere to some system design constraints such as optimum power and cost. Downsizing the system resource capacity has a direct effect on the quality of the decoded video. However, it is often a plausible scenario that the application using the decoded video is designed to tolerate a certain amount of quality degradation. Reduction in system resources is also motivated by trade-offs in system design constraints. Therefore, it becomes an important task for a system designer to narrow down on the optimal resource values (from the available resource options) so that the desired output video quality is achieved.

The conventional method to select the optimal system resources, given the required output quality, starts by first running system simulations where encoded video clips (from a large input test library) are processed by the tasks in a decoder. Here, the decoder tasks mapped onto a model of the multiprocessor system-on-chip (MPSoC) platform, are simulated in a system simulator (like the

SimpleScalar instruction set simulator [8]) in order to find the execution requirements of each task for the specific input video. The execution cycles obtained for each task are then used for the complete system analysis with varying input resource values (buffer and frequency). The optimal configuration which provides the desired video quality is then selected for the decoding function. This clearly is an inefficient exercise as these system simulations require long run times. It becomes worse when there are a large number of video clips in the input library. The example described next will substantiate the inefficiency of a conventional system simulation-based method.

5.2.1 Motivational Example

We ran simulations of five standard video clips having different characteristics (e.g., *100b_080*, which is a still video, and *susi_080*, which is a motion video clip, both 8 Mbps, 15 second clips from a standard benchmark [1]) to find the execution cycle requirements of each task in the MPEG-2 decoder at the macroblock (MB) granularity. We used high bitrate video clips as portable devices are increasingly becoming more powerful. The quality of our results will be similar for low bitrate clips with appropriate scaling down of resources. These simulations were run using the *sim-profile* configuration of SimpleScalar. The MPEG-2 decoder source code used for task profiling was from [65]. The results are shown in Fig.5.10. Though, we work with an MPEG-2 decoder, due to similarity in some of the tasks, our work is applicable to other decoders as well.

According to the results shown in Fig.5.10, it was observed that most of the time in system simulation was required to profile the Motion Compensation (MC) and Inverse Discrete Cosine Transform (IDCT) tasks. We ran the simulations for a number of motion videos and still videos. The results shown in Fig.5.10 more or less represent the kind of execution requirements observed in each motion or still video. The simulation with profiling enabled for all tasks required 27 mins 10 secs for *100b_080* and 39 mins 40 secs for *susi_080*. On the other hand, simulation with profiling enabled only for the VLD task required 3 mins 54 secs for *100b_080* and 8 mins 33 secs for *susi_080*. Hence, the simulation of MPEG-2 decoder with profiling enabled for VLD only is nearly 5 times faster for motion videos and 8 times faster for still videos in comparison to simulation with profiling enabled for all the major tasks.

These results motivated us to consider a *hybrid* simulation approach for system simulation such

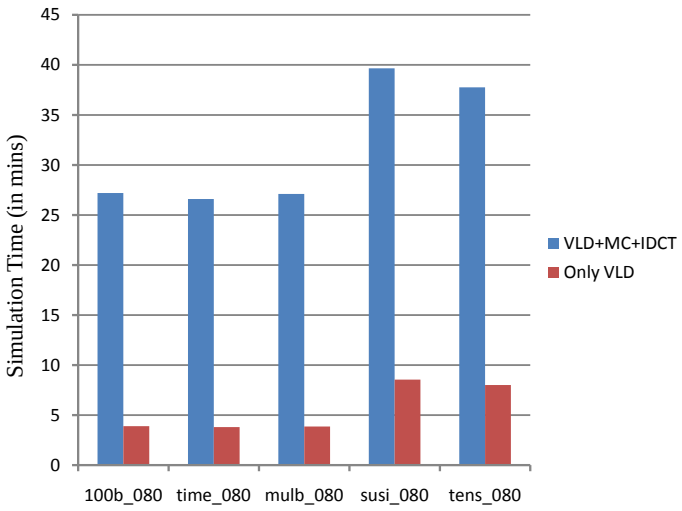


Figure 5.10: System simulation times for evaluating the execution times of various tasks in an MPEG-2 decoder. Simulating the VLD task is less expensive compared to the MC or IDCT tasks.

that the execution requirements of simulation heavy tasks in the decoder can be estimated using analytical models (which we will further refer to as the workload model). Accuracy (described in Section 5.2.4 in terms of a frame drop deviation condition) is an important requirement in our approach because the end goal of the framework is to *quantitatively* estimate the quality degradations experienced.

5.2.2 Related Work

There are not many works in the embedded systems domain which have delved into studying the behavior of decoded video quality in the context of constrained resources, especially in a MPSoC setup. Yanhong et. al. [95] investigated trade-offs between quality of MPEG-4 decoded video and processor frequency. However, this work used expensive simplescalar simulations to find the processor workload values. A recent work [96] proposed an end-to-end video quality prediction framework taking into account the packet loss in a network transmission scenario, but an accurate PSNR value estimation is not obtained.

Decoder workload prediction model is the most important component of our framework. There are few works ([97]- [92]) which discuss workload prediction models for MPEG-2 and later decoders, but these workload prediction models are not accurate enough for our framework. Frame discard

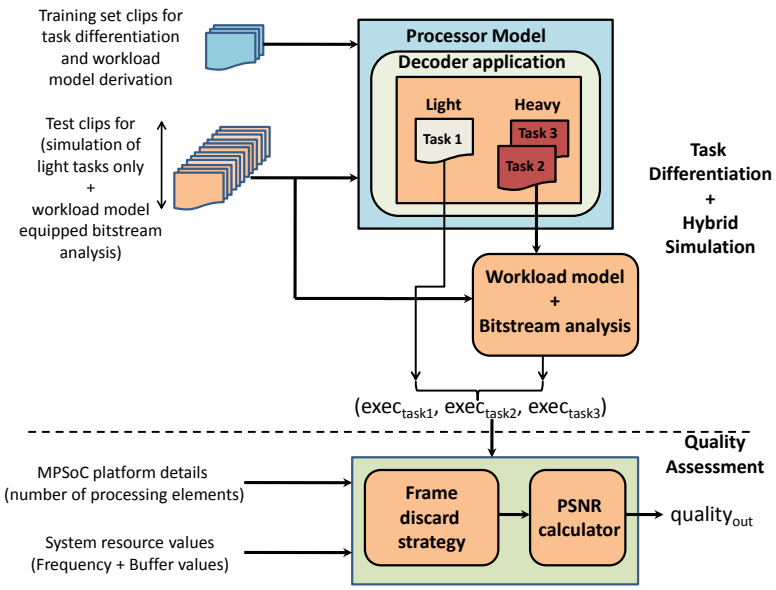


Figure 5.11: Overview of hybrid simulation-based quality assessment

algorithm is an integral part of the quality assessment framework. There has been lot of prior work ([2]- [98]) with many advanced algorithms in this area. Here, in the context of constrained resources, frames are dropped according to their relative importance in the group of pictures. However, in this work, we use a very basic frame dropping strategy.

5.2.3 Hybrid Simulation-based Quality Assessment Framework - An Overview

A schematic overview of our hybrid simulation-based quality assessment framework is shown in Fig.5.11. Given a library of encoded video clips, and the available capacity of system resources the overall task of the framework is to estimate the quality degradation experienced. An evident bottleneck in achieving this task is the huge amounts of time required in the simulation of certain tasks in the decoder. Therefore, the main steps for a fast quality assessment are as follows

1. The first step of our framework is **task differentiation**, i.e., the tasks in the decoder application are simulated in simplescalar with a few video clips from the training set to identify simulation heavy and light tasks. The task differentiation is done based on the fraction of total simulation time taken for each task. The video clips in the training set are also used to derive a workload model for each task. If the workload model is not accurate for a task, then that

task is simulated. If $\{t_k, \forall k\}$ are the tasks in the application, where $1 \leq k \leq N_T$ and N_T is the total number of tasks, the condition to find light tasks is $T_k \leq F_{th} \times T_{tot}$, where T_k is the time taken for the simulation of task t_k, F_{th} is a threshold fraction and T_{tot} is the total simulation time taken for the training video clip.

2. The next step is the process of **hybrid simulation**. Here, the execution requirements of heavy tasks from first step are estimated using bitstream analysis of the decoder application. This approach extracts important decoder parameters which are then used for estimating the execution requirements. The parameter extraction is done in the compressed domain and hence is fast. In the context of MC task, an accurate workload model was derived by tuning the model using simulation results with the training set clips. On the other hand, the IDCT task was made accurate by taking into account the algorithm implementation details. The output from this stage are task execution requirements $exec_{taski}$, where i represents the task index. The light tasks are simulated in simplescalar. In the MPEG-2 decoder context, the three major task execution requirements are $exec_{task1} = exec_{VLD}$, $exec_{task2} = exec_{MC}$ and $exec_{task3} = exec_{IDCT}$.

3. The final step is the process of **quality assessment**. This includes a high level analysis of the system functioning with details of decoder task mappings on the underlying MPSoC platform. The details of the MPSoC platform include the number of processing elements (PEs) used, PE frequencies, buffer capacities and how the tasks are mapped onto the PEs. These inputs are then used by a frame discard algorithm to detect the frames dropped. Once the frame drop indices are obtained, the PSNR denoted by $quality_{out}$ can be calculated.

The MPSoC platform architecture used for our case study of the MPEG-2 decoder application consists of multiple interconnected processing elements (PEs) as shown in Fig.2.2. The PEs communicate by passing data units or stream objects between them. PE_1 and PE_2 are the two programmable processors. It also consists of the input/network and output interface. In the base mapping strategy, for MPEG-2 decoder application on the MPSoC platform, PE_1 performs the VLD task, while PE_2 performs the IDCT and MC tasks. The stream objects on which the PEs operate are macroblocks (MBs). The input encoded MBs are stored in the input buffer (Buf_1) with size B_1. Partially decoded MBs are sent from PE_1 to PE_2 through buffer Buf_2 with size B_2 while fully processed MBs are sent

out of PE_2 to the output interface through buffer Buf_3 with size B_3. f_1 and f_2 are the frequencies of the PEs 1 and 2 respectively. We study the quality degradations due to MB drops in Buf_1 and Buf_2 when their respective sizes are exceeded for various combinations of f_1, f_2, B_1 and B_2. In the next section, we will discuss the task workload models for all the important tasks in MPEG-2 decoder and demonstrate their usefulness for some simulation heavy tasks like MC and IDCT.

5.2.4 Workload Models for Simulation Heavy Tasks

In this section, the two simulation heavy tasks (MC and IDCT) of MPEG-2 decoder and their corresponding workload models will be discussed. We use accurate workload models as estimation of quality degradation is the final goal. These task workload models compute the task execution requirements (at MB granularity). The task workload models were developed for Portable ISA (PISA) which is a MIPS like ISA. The MPEG-2 decoder source code used in the training phase for one time development of the workload models has been taken from [65].

It is difficult to get an accurate VLD task workload model, but as VLD task takes less simulation time, $exec_{VLD}$ was found using simplescalar simulations. Accuracy in workload estimation for a task is measured in terms of a **Frame Drop Deviation** (FDD) condition every time a buffer overflow condition occurs. FDD is defined as the difference in the frame indexes dropped (at each buffer overflow condition) between the scenario where the task workload values used are model-based and the real scenario where the workload values are Simplescalar simulation-based. The condition that needs to be satisfied is $FDD = 0$. The instantaneous composition of the buffer in both the scenarios are $b_{sim}(t) = r_{f1} + B\_bot_{sim}, \forall t$ and $b_{model}(t) = s_{f2} + B\_bot_{model}, \forall t$. where r_{f1} denotes that there are r number of MBs of $(f1)$-th frame in the top of the buffer and $0 \leq r < FSIZE$. A similar interpretation holds for s_{f2}. $FSIZE$ denotes the number of MBs in one frame. B_bot_{sim} and B_bot_{model} are the remaining number of MBs at the bottom of the buffer for the two scenarios. Hence, according to the FDD condition, the following expression should hold for an accurate estimation of frame drops using the workload model: $FDD = 0 \Rightarrow |f1 - f2| = 0 \Rightarrow f1 = f2$.

Algorithm 6 Computing Execution Requirement LUT for MC Workload model

Input: Training set video clips v_i where $1 \leq i \leq n$, and n is the cardinality of the training set, execution requirement values of all MBs in v_i for each call (indexed by j where $1 \leq j \leq 4$) to MC denoted by $exec_{ij}$ and parameter values for the same denoted by $parm_{ij}$;
Output: $exec_{MC}(parm)$ for all $1 \leq parm \leq 64$
1: $exec_{MC}(parm) \leftarrow 0$ for all $1 \leq parm \leq 64$;
2: **for** $i = 1$ to n **do**
3: **for** $j = 1$ to 4 **do**
4: $parm = parm_{ij}$
5: **if** $parm \neq 0$ and $exec_{MC}(parm) < exec_{ij}$ **then**
6: $exec_{MC}(parm) = exec_{ij}$
7: **end if**
8: **end for**
9: **end for**

5.2.4.1 MC Workload Model

There are three types of MBs in MPEG-2 bitstream namely I-type (do not require MC), P-type (require only forward MC) and B-type (require both forward and backward MC). The parameters we extract from the MPEG-2 bitstream for MC function workload model are (1) Y component's x-dimension is HALF-PIXEL, (2) Y component's y-dimension is HALF-PIXEL, (3) U or V component's x-dimension is HALF-PIXEL, (4) U or V component's y-dimension is HALF-PIXEL, (5) forward or backward motion compensation is required and (6) the motion compensation window size is 16x8 or 16x16. Based on the type of MB, the MC routine is called with different parameters, each requiring different workloads. A look-up table (LUT) was constructed with 64 values of processor workloads corresponding to 6 parameters. The MC routine may be called different number of times by each MB, which was also taken into consideration. In the case of MPEG-4, a similar idea is followed and we use 3 LUTs as there are more parameters to be considered. The algorithm to construct the LUT is shown as Algorithm 6.

Here, all the training video clips are simulated to find the execution requirements $exec_{ij}$ for each call to MC with the parameter values denoted by $parm_{ij}$. Finally, the worst case execution values from the training set are stored for each parameter value as $exec_{MC}$. For the test video clips, $exec_{MC}$ values are used.

5.2.4.2 IDCT Workload Model

The IDCT workload requirement is primarily related to the number and position of non zero IDCT coefficients in the 8x8 block structure of MB. In our case, the MPEG-2 streams had 6 such blocks as the chroma format was 4:1:1. The training set was used to simulate and find out the execution values for all possible combinations of strings consisting of non zero IDCT coefficients. However, this is not sufficient to make the IDCT estimation accurate adhering to the FDD condition. In fast IDCT implementations, certain shortcut conditions are checked where fewer instructions are traversed when any of the columns in the blocks are found to be entirely zero coefficients. In our IDCT workload model, we also take care of such fast implementations. Taking all these factors into consideration, the MB IDCT workload model that we have come up with for the portable ISA is

$$\begin{aligned} exec_i &= basis_i + \alpha \times (n\_add_i - n\_sub_i) - \beta \times scnt_i, \forall \text{i} \\ exec_{idct} &= \textstyle\sum_{i=1}^{6} exec_i \end{aligned} \tag{5.12}$$

Here, $exec_i$ is the execution requirement for a single block i in the MB. $basis_i$ is the base execution value for that block which takes on a value of 1965, 1852, 708 or 595 (from training set) based on the string of non-zero IDCT coefficients. From training set simulations, we found $\alpha = 113$ and $\beta = 143$. n_add_i and n_sub_i denote the number of coefficients added and subtracted respectively to determine the effective number of significant non-zero IDCT coefficients for one block. $scnt_i$ represents the number of fast bypasses encountered in a block for the fast IDCT implementations. We do not explain the IDCT workload model for MPEG-4 here, but it is exactly similar to the one discussed for MPEG-2 with changes in only the model parameter values. The final execution value is obtained by summing the values for all blocks. For test clips, n_add_i, n_sub_i, $scnt_i$ and $basis_i$ are found by a fast bitstream analysis.

5.2.5 Experimental Study

In the previous sections, we discussed the workload models for simulation heavy tasks and the hybrid simulation approach. Once the execution requirements are obtained for each of the tasks in the decoder, the MPSoC platform architecture shown in Fig.2.2 can be studied in terms of quality degradations of the input video. For this, we fix the system resources f_1, f_2, B_1 and B_2 with the

mapping of tasks as shown in Fig.2.2. In order to measure the quality degradation, we need a frame discard strategy which decides when the MBs and in turn frames are dropped and how the system starts accepting the subsequent frames.

5.2.5.1 Frame discard strategy

We use a simple frame discard strategy where an entire frame is dropped if one MB from that frame is dropped. In our experiments, we assume that only B_1 and B_2 are insufficient resources. The playout buffer is not considered here as it will essentially exhibit similar properties. At the display side, the dropped frames are substituted by the previous frame that was accepted. Some of the important aspects of the frame discard scheme are discussed below:

1. If an I-frame is dropped, then the entire set of frames following it in the group of pictures (GOP) is dropped as the decoder will not be able to interpret these frames which require the I-frame for decoding. Hence the number of frames dropped will be a minimum of GOP length. In our experiments, the GOP length was 14.

2. If a P-frame is dropped, then the number of frames dropped as a result of dependence on the P-frame is decided by the position of the P-frame in the GOP. In our experiments, where the sequence of frames was IPBBPBBPBBIBBP..., the minimum number of frames dropped after dropping a P-frame was 10, 7 or 4.

3. If a B-frame is dropped, there is no other frame dropped subsequently provided there is enough space in the buffer.

4. A frame is dropped if any of its MBs overflows the buffer (i.e. $Buf_1 > B_1$ for stage 1, the similar condition holds true for stage 2) or it is dependent on a previous dropped frame. The buffer starts accepting frames only when there is enough room for an entire frame in the buffer.

5.2.5.2 PSNR calculation

The PSNR of the output video is calculated using a difference based scheme as we do not have the original video. Noise is represented by the MSE between the actual pixel values of the dropped frame and the pixel value of the last accepted frame that substitutes the dropped frame. Let us denote the MSE in R, G and B domains as MSE_r, MSE_g and MSE_b. Then, the PSNR value of a video sequence with frame drops is expressed as

$$psnr = 10 \times log_{10} \frac{(255 \times 255)}{\frac{(MSE\_r + MSE\_g + MSE\_b)}{(3 \times N_{tot} \times W \times H)}} \tag{5.13}$$

$(MSE\_r)_n = \sum_{w=0}^{W-1} \sum_{h=0}^{H-1} (r_d(h,w,n) - r_c(h,w,n))^2$, $MSE\_r = \sum_{n=0}^{N_{drop}-1} (MSE\_r)_n$,$r_d$ is the red pixel intensity of the dropped frame and r_c is the red pixel intensity of the concealment frame. h, w and n are the height, width and frame drop number indices. Similar explanation holds true for MSE_g and MSE_b. W and H are the horizontal and vertical resolution of each frame in the video. N_{tot} is the total number of frames in the video sequence and N_{drop} is the number of frames dropped in the sequence. From [1], the training set clips were $100b\_080$, $bbc3\_080$, $pulb\_080$ and $susi\_080$ and the test set clips were $cact\_080$, $flwr\_080$, $mobl\_080$, $mulb\_080$, $tens\_080$, $time\_080$ and $v700\_080$.

5.2.5.3 Results and Discussion

We estimated the quality degradations for varying values of f_1, f_2, B_1 and B_2 taking the task execution values obtained from hybrid simulation, frame discard strategy and system resource values into consideration for system simulation. The piece of code to calculate this was plugged into the C code of the MPEG-2 decoder. Our quality assessment framework was used to understand various interesting properties of the MPSoC platform discussed earlier. Here, we show some interesting results for two task mappings *basemap* (as shown in Fig.2.2) and *newmap* (VLD and MC mapped to PE_1 and IDCT mapped to PE_2), highlighting few important trade-off aspects of the MPSoC platform for a decoder application.

Estimation Accuracy: Before we get into detailed trade-off aspects, we would like to emphasize that all the PSNR values obtained and plotted on the graph were also verified using real simulations. For every test clip, we estimated PSNR values with 1500 resource combinations. In the case of

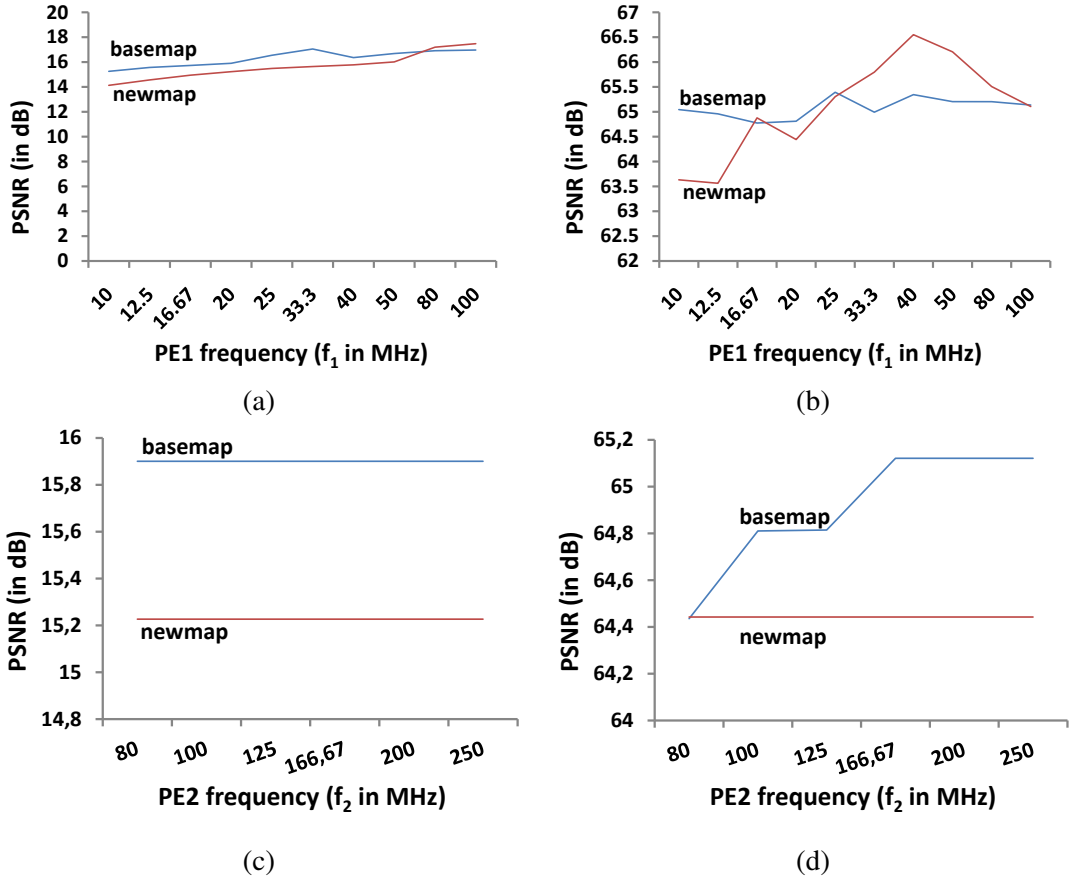


Figure 5.12: PSNR vs the system resource values f_1 and f_2 for two test videos (a) PSNR vs f_1 for *tens*_080, (b) PSNR vs f_1 for *v*700_080, (c) PSNR vs f_2 for *tens*_080, (d) PSNR vs f_2 for *v*700_080, (e) PSNR vs B_1 for *tens*_080, (f) PSNR vs B_1 for *v*700_080, (g) PSNR vs B_2 for *tens*_080 and (h) PSNR vs B_2 for *v*700_080.

motion video clips for *basemap*, the values were accurate for more than 98% of the resource combinations and the deviations in PSNR estimates for the other 2% cases were less than $\pm 0.3\%$. On the other hand, in the case of still video clips for *basemap*, the estimated PSNR values were 100% accurate. The small variation in motion video clips is due to their dynamic nature which results in a small drift from the hybrid simulation estimate. For *newmap*, we get nearly 100% PSNR estimation accuracy emphasizing that the method works irrespective of task mapping.

PSNR vs System Resource tradeoff: The detailed PSNR vs f_1 trade-off is shown in Fig.5.12(a) and Fig.5.12(b) for the clips *tens*_080 and *v*700_080 respectively. In this case $f_2 = 100$ MHz, $B_1 = 175000$ MBs and $B_2 = 100000$ MBs. It is interesting to note that irrespective of the task mappings,

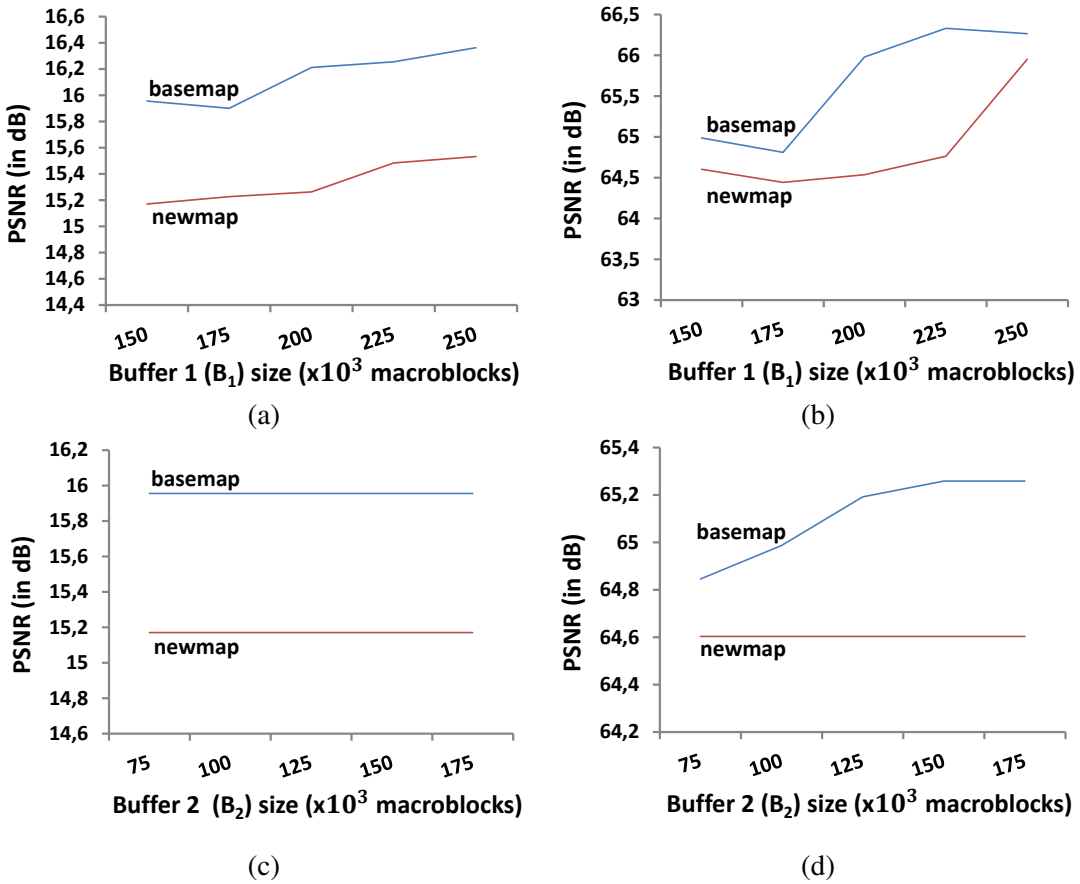


Figure 5.13: PSNR vs the system resource values B_1 and B_2 for two test videos (a) PSNR vs B_1 for *tens*_080, (b) PSNR vs B_1 for *v*700_080, (c) PSNR vs B_2 for *tens*_080 and (d) PSNR vs B_2 for *v*700_080.

there are some f_1 values for which the PSNR value is high after which it drops due to forwarding of more partially processed frames to Buf_2 and causing overflow. At low values of f_1, Buf_1 is the bottleneck and hence results in low PSNR. We therefore arrive at the intuition that there is a particular f_1 value for each set of other fixed resources when PSNR hits the highest value for both *basemap* and *newmap* task mapping strategies.

The PSNR vs f_2 trade-off for the two test videos is shown in Fig.5.12(c) and Fig.5.12(d). The other fixed resource values are $f_1 = 20$ MHz, $B_1 = 175000$ MBs and $B_2 = 100000$ MBs. As f_2 increases, the PSNR should also increase and eventually stabilize as the drops in Buf_2 decrease, which is seen in *basemap* for the clip *v*700_080. However, the PSNR for clip *tens*_080 is constant for both *basemap* and *newmap* because the increase of f_2 becomes redundant when the maximum number

of frame drops have already occurred. In this case, the lowest possible f_2 value can be chosen. This trend is also seen in the case of *newmap* for the clip *v*700_080.

It is observed from the PSNR vs B_1 curves in Figs.5.13(a)-(d) that the expected trend is followed whereby the PSNR value increases with increasing B_1 value for both mappings. However, in the *basemap* case for *v*700_080, this trend is not always true as the frames dropped due to increase of buffer sizes result in high MSE values which in turn lead to lower PSNR values. The fixed resource settings for Figs.5.13(a)-(d) are $f_1 = 20$ MHz, $f_2 = 100$ MHz and $B_2 = 100000$ MBs.

In the case of PSNR vs B_2 curve, the trend is similar to the PSNR vs f_2 curve. It is easily understood that the reasoning behind the behavior is also the same. The fixed resource values for curves in Figs.5.13(c)-(d) are $f_1 = 20$ MHz, $f_2 = 100$ MHz and $B_1 = 150000$ MBs. An interesting observation from Figs.5.12(a)-(d) and Figs.5.13(a)-(d) is that the effect of f_1 and B_1 on the PSNR values of the same clip have significant difference when compared to f_2 and B_2 for the various fixed resource settings we employed. Hence, optimum values must be selected from the trade-off curves. It is also observed that *basemap* is a better mapping strategy in most of the cases when compared to *newmap*.

Central focus: Although we have discussed many interesting PSNR vs [f_1,f_2,B_1,B_2] trade-off aspects in this section, it is essential to emphasize that the main focus of this work was to devise a hybrid simulation strategy in order to enable the system designers to rapidly arrive at quantitative estimates of quality degradations for video clips decoded by a multimedia decoder mapped onto a MPSoC platform with resource constraints. This framework will also be very useful in comprehensively understanding the existing non-trivial influences of system resources on quantitative quality degradations, which could be further exploited to estimate optimal points in the resource space for desired quality. However, we do not consider it extensively here.

5.3 Summary

In the first work, we have presented a fast and efficient model-based test case classification methodology for performance analysis of multimedia MPSoC platforms. Our method completely eliminates the time consuming simulations required to cluster the library of video clips. It also gives the system designer control over the selection of the number of representative video clips. We have

validated our method in the context of a MPEG-2 decoder application running on a MPSoC architecture with two PEs. The performance metrics analyzed to prove the validity of the method were worst case buffer backlog and worst case delay for one macroblock. The methodology discussed in this work is also envisaged to work well if the tasks are mapped separately to different processors.

In the second work, we discussed a fast hybrid simulation-based quantitative quality estimation framework for decoded video on a MPSoC platform with limited resources. It partitions the tasks of the decoder into simulation heavy and light tasks based on the ease of deriving task workload models. The execution cycle requirements of the simulation heavy tasks are derived using accurate workload models and the light tasks are simulated. This enables in accurate assessment of quality degradations. This framework is efficient and of immense use to a system designer in analyzing the various PSNR vs resource trade-off characteristics.

The model-based test case classification methodology can be extended to parameterized classification where the maximum number of classes or clusters required can be found by using tolerance constraints for buffer size for video clips within clusters. The model can also be extended to reflect the effects of microarchitectural features like cache etc on the workload of multimedia streams.

Chapter 6

Concluding Remarks

The summary of the thesis is first presented in this chapter and then some of the possible future works are discussed.

6.1 Summary

This thesis focused on system level performance analysis techniques for multimedia MPSoC platforms using a quality-aware approach. Real-Time Calculus (RTC) has been widely used for performance analysis of hard real-time systems. It is a deterministic performance analysis method that provides hard upper and lower bounds for the performance parameters. RTC uses interval-based representations to model both event streams (arrival curves) and resources (service curves). These curves are then used to evaluate the system performance using certain operations. The techniques presented here add on to the body of work on RTC based performance analysis techniques for embedded systems. It incorporates multimedia specific characteristics for resource dimensioning problems and workload estimations, which help in considerable reductions in resource requirements and simulation times for performance analysis.

On-chip buffer sizes are an important design goal in multimedia MPSoC architectures due to their contribution to the overall chip area. Therefore, two buffer dimensioning methodologies were presented first that reduce buffer requirements by trading off with quality of the video played at the output. This is a very important technique for multimedia streams because they can tolerate some

frame losses without affecting the video perception. In the first method, a mathematical framework was presented to study the trade-off between buffer size and objective quality in terms of PSNR. However, this framework did not take into consideration the priority among the frames that were dropped, which did not help to achieve more buffer reductions. In the second method, a simulation based framework was proposed which prioritizes the dropping of frames in order to design smaller buffer sizes for target output video quality in comparison to the mathematical framework mentioned above. However, the simulation framework requires more time for deriving the appropriate buffer size.

Processor bandwidth share is another important system parameter. A mathematical framework was presented in order to derive the processor cycle requirements for decoding video clips with bounded frame drops for MPSoC platforms with buffer constraints. The bounds on the processor cycle requirements obtained was used to schedule the processing multiple MPEG-2 videos such that both the decoded video clips satisfied a target quality constraint. This setup is useful for a PiP application.

Thermal capacity has become an important design concern lately. There are many works that try to achieve a reduction in the peak temperature subject to various design objectives. In this thesis, the concept of bounded frame drops was used to reduce the latency or end-to-end delay in video display while adhering to the peak temperature constraint. It was observed that for acceptable quality outputs, the latency can be reduced considerably.

Finally, two fast simulation based frameworks were used to utilize the multimedia stream characteristics to estimate the workload required for the various tasks in MPEG-2/MPEG-4 decoding. First, this was used to quickly classify the video clip library into representative sets, which allow the use of representative videos from each set in order to bring down the simulation time. In the second framework, the workload estimation was used to derive a hybrid simulation strategy, which was used to accurately compute the quality degradations in MPSoC platforms with resource constraints.

6.2 Future Work

The future works that are discussed here build upon the performance analysis techniques presented in this thesis.

6.2.1 Analytical framework for quality-driven buffer dimensioning with frame priority constraints

The mathematical framework presented in Chapter 2 to perform quality-driven buffer dimensioning for MPSoC platforms did not consider the inherent quality information in the frames. For example, within the B frames, dropping certain B frames results in larger distortion in comparison to certain other B frames. However, the current analytical framework drops the frames using the *drop oldest* frame scheme. The frame drop priority information can be used while computing the bounds on the number of frame drops. This is expected to reduce the buffer size estimations further.

The analytical framework proposed in Section 2.2 developed the interval based parameters of RTC such as delay, service bounds etc based on the assumption that the oldest frame in the buffer is dropped if the buffer overflow condition occurs. However, this strategy would drop the frames without taking into consideration the distortion caused by the dropping of that particular frame. The higher the distortion caused by the dropped frame, the lesser the number of frames that can be dropped further as the quality constraint has to be satisfied. Hence, it would be interesting to incorporate the priority based drop in the analytical framework and redefine the quantities like delay, service bounds etc.

6.2.2 Frame size considerations for buffer dimensioning along with motion vector

The simulation framework presented in Chapter 2 for quality-driven buffer dimensioning uses motion vectors only to drop maximum number of frames and thereby reduce buffer occupancy. However, in order to actually see buffer size reduction in bits, we also have to consider the frame sizes as there is a large variability in the sizes of frames. This would result in a knapsack like problem where the cumulative quality degradation by dropping frames cannot exceed a target value and the dropped frame sizes have to be maximized.

This problem can be defined as an optimization problem where the two objectives are to keep the quality losses below a prespecified quality constraint Q_{target} and the cumulative frame size of the dropped frames should be maximized. We intend to solve this problem using an ILP solution strategy. Here, we would select the frames to be dropped using motion vector based prioritization.

Let us call these frames as drop candidates. However, the final set of dropped frames will be decided from the drop candidates by searching for the appropriate set that maximizes the cumulative size of the dropped frames.

6.2.3 Joint design space exploration of buffer size and processor bandwidth

In this thesis, the mathematical frameworks that we present derive the resource requirements by keeping the other resource requirements at a constant value. We trade-off each resource with quality while keeping the other resources constant. However, it is an interesting problem to derive the pareto curve for buffer size and processor cycle given the target quality constraint. There are a huge number of configuration choices for these two system parameters. This framework will help the designer to choose the resource combination with the largest possible resource savings.

The two extremes of buffer size and processor bandwidth resource set satisfying a quality constraint are obtained using the two analytical frameworks described in Chapter 2 and Chapter 3. In between these two extreme configuration sets that greedily optimize only one of the two resources, there is a large design space that needs to be explored in order to obtain an optimized configuration set that satisfies some objective function like power consumption etc. Here the two extremes might not be the best candidate configuration set.

6.2.4 Lowest peak temperature estimation

For a system designer, given the available resources and the quality constraints that have to be met at the output (i.e. allowing some frame drops), it will be helpful to find the frame drop patterns that will lead to lowest peak temperature. It will be challenging to explore this problem in a multiple PEs scenario because the frame drops on one PE have to take into account that the temperature reductions are also optimized on the succeeding PEs.

This problem is quite challenging due to the inherent variability in the multimedia stream processing. In order to gain maximum advantage from frame drops, it is required to find the critical section of the frame sequence that would lead to the an overall maximum rise in temperature across both the PEs.

6.2.5 Parameterized test case classification for fast performance analysis

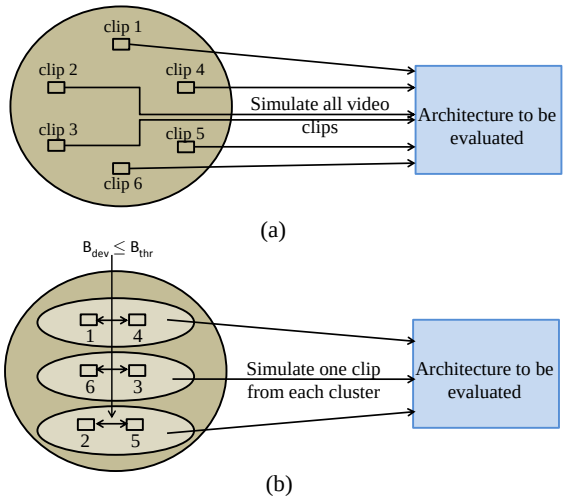


Figure 6.1: Cluster formation based on condition that buffer occupancy deviation B_{dev} is less than a threshold B_{thr}

In the completed work on test case classification [99], we do not have a systematic method of choosing the number of clusters into which the library of video clips must be classified for a target multimedia MPSoC platform. It was left to the system designer to choose the appropriate number of clusters based on his/her understanding of the target system. However, it is a better approach to classify test video clips based on some parameters set apriori by the system designer. Here, specifically we would like to explore test video classification based on the maximum tolerance in deviation of performance parameters such as buffer/end-to-end delay within a cluster as shown in Fig.6.1. This would automatically help the system designer to find out how many clusters will be required and hence the number of representative test clips.

This work will also involve in performing a fine-grained test case classification where the test video clips will be fragmented and the fragments of the video clips will be classified based on the technique described earlier. The fragments can be a single video frame or a group of pictures (GOP).

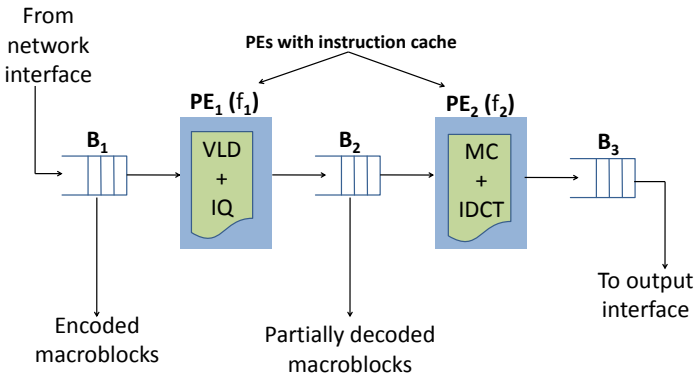


Figure 6.2: Workload model for tasks on PEs taking instruction cache in PE into consideration

6.2.6 Workload model derivation in the context of microarchitectural features like cache

The MPSoC platform that was used in [99] to evaluate our model-based performance analysis method consisted of a simple architecture consisting of two PEs. However, the state-of-the-art in MPSoCs include microarchitectural features like cache Fig.6.2. The usage of instruction cache brings down the execution cycle requirements of a PE if temporal locality is present in the sequence of instructions executed. This will affect the workload model that we currently use for a MPEG-2 decoder as we will get more tighter execution cycle requirements. Hence, there arises a need to develop a model to integrate the differences in the architecture experienced due to the introduction of these microarchitectural features. Moreover, it will be interesting to see if decoding video clips in a cluster require similar instruction cache sizes for a particular cache hit ratio as we have proved for various buffer sizes in the architecture that hold data.

Bibliography

[1] "Mpeg-2 benchmark videos," ftp://ftp.tek.com/tv/test/streams/Element/MPEG-Video/625/.

[2] D. Isovic and G. Fohler, "Quality aware mpeg-2 stream adaptation in resource constrained systems," in *16th Euromicro Conference on Real-Time Systems (ECRTS)*, 2004, pp. 23–32.

[3] W. Tu, W. Kellerer, and E. Steinbach, "Rate-distortion optimized video frame dropping on active network nodes," in *Packet Video Workshop*, 2004.

[4] M. Jersak, R. Henia, and R. Ernst, "Context-aware performance analysis for efficient embedded system design," in *7th Design, Automation and Test in Europe (DATE)*, 2004, pp. 1046–1051.

[5] S. Dutta, R. Jensen, and A. Rieckmann, "Viper: A multiprocessor soc for advanced set-top box and digital tv systems," *IEEE Design & Test of Computers*, vol. 18, no. 5, pp. 21–31, 2001.

[6] M. J. Rutten, J. T. J. van Eijndhoven, E. G. T. Jaspers, P. van der Wolf, O. P. Gangwal, A. Timmer, and E. J. D. Pol, "A heterogeneous multiprocessor architecture for flexible media processing," *IEEE Design & Test of Computers*, vol. 19, no. 4, pp. 39–50, 2002.

[7] C. Lee, M. Potkonjak, and W. H. Mangione-Smith, "Mediabench: A tool for evaluating and synthesizing multimedia and communicatons systems," in *30th ACM/IEEE International Symposium on Microarchitecture*, 1997, pp. 330–335.

[8] T. M. Austin, E. Larson, and D. Ernst, "Simplescalar: An infrastructure for computer system modeling," *IEEE Computer*, vol. 35, no. 2, pp. 59–67, 2002.

[9] A. A. Omar and F. A. Mohammed, "A survey of software functional testing methods," *ACM SIGSOFT Software Engineering Notes*, vol. 16, no. 2, pp. 75–82, 1991.

[10] G. Varatkar and R. Marculescu, "On-chip traffic modeling and synthesis for mpeg-2 video applications," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 12, no. 1, pp. 108–119, 2004.

[11] K. Richter, M. Jersak, and R. Ernst, "A formal approach to mpsoc performance verification," *IEEE Computer*, vol. 36, no. 4, pp. 60–67, 2003.

[12] R. Alur and D. L. Dill, "A theory of timed automata," *Theoretical Computer Science*, vol. 126, no. 2, pp. 183–235, 1994.

[13] F. E. B. Ophelders, S. Chakraborty, and H. Corporaal, "Intra-and inter-processor hybrid performance modeling for mpsoc architectures," in *International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2008.

[14] T. Wild, A. Herkersdorf, and R. Ohlendorf, "Performance evaluation for system-on-chip architectures using trace-based transaction level simulation," in *9th Design, Automation and Test in Europe (DATE)*, 2006, pp. 248–253.

[15] L. Gao, K. Karuri, S. Kraemer, R. Leupers, G. Ascheid, and H. Meyr, "Multiprocessor performance estimation using hybrid simulation," in *45th Design Automation Conference*, 2008, pp. 325–330.

[16] L. Thiele, S. Chakraborty, and M. Naedele, "Real-time calculus for scheduling hard real-time systems," in *IEEE International Symposium on Circuits and Systems*, 2000, pp. 101–104.

[17] S. Chakraborty, S. Kunzli, and L. Thiele, "A general framework for analysing system properties in platform-based embedded system designs," in *6th Design, Automation and Test in Europe (DATE)*, 2003, pp. 190–195.

[18] E. Wandeler and L. Thiele, "Workload correlations in multi-processor hard real-time systems," *Journal of Computer and System Sciences*, vol. 73, no. 2, pp. 207–224, 2007.

[19] ——, "Characterizing workload correlations in multi processor hard real-time systems," in *11th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2005, pp. 46–55.

[20] E. Wandeler, L. Thiele, M. Verhoef, and P. Lieverse, "System architecture evaluation using modular performance analysis: a case study," *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 8, no. 6, pp. 649–667, 2006.

[21] E. Wandeler and L. Thiele, "Abstracting functionality for modular performance analysis of hard real-time systems," in *Asia South Pacific Design Automation Conference (ASP-DAC)*, 2005, pp. 697–702.

[22] S. V. Gheorghita, T. Basten, and H. Corporaal, "An overview of application scenario usage in streaming-oriented embedded system design," *Eindhoven University of Technology, Tech Report*, no. esr-2006-03, 2006.

[23] G. Raghavan, A. Salomaki, and R. Lencevicius.

[24] A. D. Popescu, "Media streaming in peer-to-peer overlay networks."

[25] M. M. Hefeeda, B. K. Bhargava, and D. K. Y. Yau, "A hybrid architecture for cost-effective on-demand media streaming," *Computer Networks*, vol. 44, no. 3, pp. 353–382, 2004.

[26] H. Yin, C. Lin, F. Qiu, X. Liu, and D. Wu, "Truststream: a novel secure and scalable media streaming architecture," in *13th ACM International Conference on Multimedia*, 2005, pp. 295–298.

[27] H. Jenkac, T. Stockhammer, and G. Kuhn, "Streaming media in variable bit-rate environments," in *Packet Video Workshop*, 2003.

[28] L. Ying, R. Srikant, and S. Shakkottai, "The asymptotic behavior of minimum buffer size requirements in large p2p streaming networks," in *Information Theory and Applications Workshop*, 2010, pp. 1–6.

[29] G. Liang and B. Liang, "Effect of delay and buffering on jitter-free streaming over random vbr channels," *IEEE Transactions on Multimedia*, vol. 10, no. 6, pp. 1128–1141, 2008.

[30] M. Narbutt and L. Murphy, "Voip playout buffer adjustment using adaptive estimation of network delays."

[31] K. Fujimoto, S. Ata, and M. Murata, "Adaptive playout buffer algorithm for enhancing perceived quality of streaming applications," in *IEEE Global Telecommunications Conference (GLOBECOM)*, vol. 3, 2002, pp. 2451–2457.

[32] N. Sarshar and X. Wu, "Buffer size reduction through buffer sharing for streaming applications," in *IEEE International Conference on Multimedia and Expo (ICME)*, vol. 3, 2004, pp. 1635–1638.

[33] S. Stuijk, M. Geilen, and T. Basten, "Exploring trade-offs in buffer requirements and throughput constraints for synchronous dataflow graphs," *Eindhoven University of Technology, Tech Report*, no. esr-2006-01, 2006.

[34] S. Rampal, D. P. Agrawal, and D. S. Reeves, "Processor scheduling algorithms for minimizing bu er requirements in multimedia applications," 1994.

[35] J. Nieh and M. S. Lam, "Integrated processor scheduling for multimedia," in *Network and Operating Systems Support for Digital Audio and Video*, 1995, pp. 202–205.

[36] P. Goyal, X. Guo, and H. M. Vin, "A hierarchical cpu scheduler for multimedia operating systems," *ACM SIGOPS Operating Systems Review*, vol. 30, pp. 107–121, 1996.

[37] W. Yuan and K. Nahrstedt, "Energy-efficient cpu scheduling for multimedia applications," *ACM Transactions on Computer Systems (TOCS)*, vol. 24, no. 3, pp. 292–331, 2006.

[38] B. Lee, E. Nurvitadhi, R. Dixit, C. Yu, and M. Kim, "Dynamic voltage scaling techniques for power efficient video decoding," *Journal of Systems Architecture*, vol. 51, no. 10, pp. 633–652, 2005.

[39] D. Son, C. Yu, and H. N. Kim, "Dynamic voltage scaling on mpeg decoding," in *8th International Conference on Parallel and Distributed Systems (ICPADS)*, 2001, pp. 633–640.

[40] I. Yeo and E. J. Kim, "Hybrid dynamic thermal management based on statistical characteristics

of multimedia applications," in *13th International Symposium on Low Power Electronics and Design*, 2008, pp. 321–326.

[41] S. Mohapatra, N. Dutt, A. Nicolau, and N. Venkatasubramanian, "Dynamo: A cross-layer framework for end-to-end qos and energy optimization in mobile handheld devices," *IEEE Journal on Selected Areas in Communications*, vol. 25, no. 4, pp. 722–737, 2007.

[42] R. T. Apteker, J. A. Fisher, V. S. Kisimov, and H. Neishlos, "Video acceptability and frame rate," *IEEE Multimedia*, vol. 2, no. 3, pp. 32–40, 1995.

[43] D. Wijesekera, J. Srivastava, A. Nerode, and M. Foresti, "Experimental evaluation of loss perception in continuous media," *Multimedia systems*, vol. 7, no. 6, pp. 486–499, 1999.

[44] I. Recommendation, "500-11, methodology for the subjective assessment of the quality of television pictures," *International Telecommunication Union, Geneva, Switzerland*, 2002.

[45] S. Chikkerur, V. Sundaram, M. Reisslein, and L. J. Karam, "Objective video quality assessment methods: A classification, review, and performance comparison," *IEEE Transactions on Broadcasting*, vol. 57, no. 99, pp. 165–182, 2011.

[46] P. ITU-T RECOMMENDATION, "Subjective video quality assessment methods for multimedia applications," 1995.

[47] "Subjective video quality assessment [online]," http://www.acceptv.com.

[48] J. Y. L. Boudec and P. Thiran, *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Springer, 2001, vol. LNCS 2050.

[49] J. Hu, U. Y. Ogras, and R. Marculescu, "System-level buffer allocation for application-specific networks-on-chip router design," *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 25, no. 12, pp. 2919–2933, 2006.

[50] G. Varatkar and R. Marculescu, "Traffic analysis for on-chip networks design of multimedia applications," in *39th Design Automation Conference (DAC)*, 2002, pp. 795–800.

[51] P. Jamieson, W. Luk, S. J. E. Wilton, and G. Constantinides, "An energy and power consumption analysis of fpga routing architectures," in *International Conference on Field-Programmable Technology*, 2009, pp. 324–327.

[52] H. Wang, L.-S. Peh, and S. Malik, "Power-driven design of router microarchitectures in on-chip networks," in *36th Annual IEEE/ACM International Symposium on Microarchitecture*, 2003, pp. 105–116.

[53] A. Maxiaguine, S. Kunzli, L. Thiele, and S. Chakraborty, "Evaluating schedulers for multimedia processing on buffer-constrained soc platforms," *IEEE Design & Test of Computers*, vol. 21, no. 5, pp. 368–377, 2004.

[54] B. Raman, S. Chakraborty, O. W. Tsang, and S. Dutta, "Reducing data-memory footprint of multimedia applications by delay redistribution," in *44th Design Automation Conference (DAC)*, 2007, pp. 738–743.

[55] M. Coenen, S. Murali, A. Radulescu, K. Goossens, and G. D. Micheli, "A buffer-sizing algorithm for networks on chip using tdma and credit-based end-to-end flow control," in *4th International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2006, pp. 130–135.

[56] A. Nandi and R. Marculescu, "System-level power-performance analysis for embedded systems design," in *38th Design Automation Conference (DAC)*, 2001, pp. 599–604.

[57] M. Kalman, E. G. Steinbach, and B. Girod, "System-level buffer allocation for application-specific networks-on-chip router design," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 14, no. 6, pp. 841–851, 2004.

[58] A. Dua and N. Bambos, "Buffer management for wireless media streaming," in *GLOBECOM*, 2007, pp. 5226–5230.

[59] L. Zhang and H. Fu, "Dynamic bandwidth allocation and buffer dimensioning for supporting video-on-demand services in virtual private networks," *Computer Communications*, vol. 23, no. 14-14, pp. 1410–1424, 2000.

[60] A. Maxiaguine, S. Kunzli, S. Chakraborty, and L. Thiele, "Rate analysis for streaming applications with on-chip buffer constraints," in *9th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2004, pp. 131–136.

[61] A. Maxiaguine, S. Chakraborty, and L. Thiele, "Dvs for buffer-constrained architectures with predictable qos-energy tradeoffs," in *3rd International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2005, pp. 111–116.

[62] J. Ray and P. Koopman, "Data management mechanisms for embedded system gateways," in *DSN*, 2009, pp. 175–184.

[63] A. Vishwanath, P. Dutta, M. Chetlu, P. Gupta, S. Kalyanaraman, and A. Ghosh, "Perspectives on quality of experience for video streaming over wimax," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 13, no. 4, pp. 15–25, 2010.

[64] "Hubblesource mpeg benchmark videos," http://hubblesource.stsci.edu/sources/video/clips/index_2.php.

[65] "Mpeg-2 decoder source code," http://www.mpeg.org/MPEG/video/mssg-free-mpeg-software.html.

[66] "Samsung brings pip to mobile phones," http://gizmodo.com/252919/samsung-brings-pip-to-mobile-phones.

[67] K. Skadron, M. R. Stan, K. Sankaranarayanan, W. Huang, S. Velusamy, and D. Tarjan, "Temperature-aware microarchitecture: Modeling and implementation," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 1, no. 1, pp. 94–125, 2004.

[68] R. Jejurikar, C. Pereira, and R. K. Gupta, "Leakage aware dynamic voltage scaling for real-time embedded systems," in *Design Automation Conference (DAC)*, 2004, pp. 275–280.

[69] S. M. Martin, K. Flautner, T. N. Mudge, and D. Blaauw, "Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads," in *IEEE/ACM International Conference on Computer-aided Design (ICCAD)*, 2002, pp. 721–725.

[70] K. Skadron, "Hybrid architectural dynamic thermal management," in *Design Automation & Test in Europe (DATE)*, 2004, pp. 10–15.

[71] Y. Liu, H. Yang, R. P. Dick, H. Wang, and L. Shang, "Thermal vs energy optimization for dvfs-enabled processors in embedded systems," in *8th International Symposium on Quality of Electronic Design (ISQED)*, 2007, pp. 204–209.

[72] M. Bao, A. Andrei, P. Eles, and Z. Peng, "Temperature-aware idle time distribution for energy optimization with dynamic voltage scaling," in *Design Automation & Test in Europe (DATE)*, 2010, pp. 21–26.

[73] ——, "On-line thermal aware dynamic voltage scaling for energy optimization with frequency/temperature dependency consideration," in *Design Automation Conference (DAC)*, 2009, pp. 490–495.

[74] P. Kumar and L. Thiele, "End-to-end delay minimization in thermally constrained distributed systems," in *23rd Euromicro Conference on Real-Time Systems (ECRTS)*, 2011, pp. 81–91.

[75] J. Srinivasan and S. V. Adve, "Predictive dynamic thermal management for multimedia applications," in *17th Annual International Conference on Supercomputing (ICS)*, 2003, pp. 109–120.

[76] I. Yeo, H. K. Lee, E. J. Kim, and K. H. Yum, "Effective dynamic thermal management for mpeg-4 decoding," in *25th International Conference on Computer Design (ICCD)*, 2007, pp. 623–628.

[77] W. Lee, K. Patel, and M. Pedram, "Dynamic thermal management for mpeg-2 decoding," in *11th International Symposium on Low Power Electronics and Design (ISLPED)*, 2006, pp. 316–321.

[78] ——, "Gop-level dynamic thermal management in mpeg-2 decoding," *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, vol. 16, no. 6, pp. 662–672, 2008.

[79] M. A. Baker, V. Parameswaran, K. S. Chatha, and B. Li, "Power reduction via macroblock prioritization for power aware h. 264 video applications," in *6th IEEE/ACM/IFIP International*

Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2008, pp. 261–266.

[80] D. Gangadharan, H. Ma, S. Chakraborty, and R. Zimmermann, "Video quality-driven buffer dimensioning in mpsoc platforms via prioritized frame drops," in *29th International Conference on Computer Design (ICCD)*, 2011, pp. 247–252.

[81] R. Jayaseelan and T. Mitra, "Temperature aware task sequencing and voltage scaling," in *IEEE/ACM International Conference on Computer-aided Design (ICCAD)*, 2008.

[82] L. Eeckhout, H. Vandierendonck, and K. D. Bosschere, "Workload design: Selecting representative program-input pairs," in *International Conference on Parallel Architectures and Compilation Techniques*, 2002, pp. 83–94.

[83] K. Hoste and L. Eeckhout, "Microarchitecture-independent workload characterization," *IEEE Micro*, vol. 27, no. 3, pp. 63–72, 2007.

[84] L. K. John, P. Vasudevan, and J. Sabarinathan, "Workload characterization: Motivation, goals and methodology," in *International Workshop on Workload Characterization: Methodology and Case Studies*, 1999, pp. 3–14.

[85] A. Maxiaguine, L. Yanhong, S. Chakraborty, and O. W. Tsang, "Identifying representative workloads in designing mpsoc platforms for media processing," in *2nd Workshop on Embedded Systems for Real-Time Multimedia (ESTImedia)*, 2004, pp. 41–46.

[86] S. V. Gheorghita, T. Basten, and H. Corporaal, "Scenario selection and prediction for dvs-aware scheduling of multimedia applications," *Journal of Signal Processing Systems*, vol. 50, no. 2, pp. 137–161, 2008.

[87] H. Yicheng, S. Chakraborty, and W. Ye, "Using offline bitstream analysis for power-aware video decoding in portable devices," in *13th ACM International Conference on Multimedia*, 2005, pp. 299–302.

[88] I. T. Jolliffe, *Principal component analysis*. Springer New York, 2002.

[89] A. Joshi, A. Phansalkar, L. Eeckhout, and L. K. John, "Measuring benchmark similarity using inherent program characteristics," *IEEE Transactions on Computers*, vol. 55, no. 6, pp. 769–782, 2006.

[90] J. Hamers and L. Eeckhout, "Resource prediction for media stream decoding," in *10th Design, Automation and Test in Europe (DATE)*, 2007, pp. 594–599.

[91] http://www.tns.lcs.mit.edu/manuals/mpeg2/.

[92] H. Yicheng, V. A. Tran, and W. Ye, "A workload prediction model for decoding mpeg video and its application to workload-scalable transcoding," in *15th International Conference on Multimedia (MM)*, 2007, pp. 952–961.

[93] W. Pan and A. Ortega, "Complexity-scalable transform coding using variable complexity algorithms," in *Data Compression Conference*, 2000.

[94] A. D. Gordon, *Classification*. Chapman & Hall/CRC, 1999.

[95] L. Yanhong, S. Chakraborty, O. W. Tsang, A. Gupta, and S. Mohan, "Workload characterization and cost-quality tradeoffs in mpeg-4 decoding on resource-constrained devices," in *3rd IEEE Workshop on Embedded Systems for Real-Time Multimedia (ESTIMedia)*, 2005.

[96] H. Koumaras, C. H. Lin, C. K. Shieh, and A. Kourtis, "A framework for end-to-end video quality prediction of mpeg video," *Journal of Visual Communication and Image Representation*, vol. 21, no. 2, pp. 139–154, 2010.

[97] M. Roitzsch and M. Pohlack, "Principles for the prediction of video decoding times applied to mpeg-1/2 and mpeg-4 part 2 video," in *27th IEEE International Real-Time Systems Symposium (RTSS)*, 2006.

[98] D. Isovic, G. Fohler, and L. Steffens, "Timing constraints of mpeg-2 decoding for high quality video: misconceptions and realistic assumptions," in *15th Euromicro Conference on Real-Time Systems (ECRTS)*, 2003, pp. 73–82.

[99] D. Gangadharan, S. Chakraborty, and R. Zimmermann, "Fast model-based test case classification for performance analysis of multimedia mpsoc platforms," in *7th IEEE/ACM International*

Conference on Hardware/software codesign and System Synthesis (CODES+ISSS), 2009, pp. 413–422.