Performance and QoS-aware MPEG-4 video-object coding for multiprocessor architecture

Performance and QoS-aware MPEG-4 video-object coding for multiprocessor architecture

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van de
Rector Magnificus, prof.dr.ir. C.J. van Duijn, voor een
commissie aangewezen door het College voor
Promoties in het openbaar te verdedigen op
donderdag 24 januari 2008 om 16.00 uur

door

Milan Paštrnák

geboren te Čadca, Slowakije

Dit proefschrift is goedgekeurd door de promotoren:

prof.dr.ir. P.H.N. de With
en
prof.dr.ir. J.L. van Meerbergen

CIP-DATA LIBRARY TECHNISCHE UNIVERSITEIT EINDHOVEN

Paštrnák, Milan

Performance and QoS-aware MPEG-4 video-object coding for multiprocessor architecture / by Milan Paštrnák. - Eindhoven : Technische Universiteit Eindhoven, 2008.
Proefschrift. - ISBN 978-90-386-1744-2
NUR 959
Trefw.: beeldcodering / multiprocessoren / elektronische beeldtechniek ; beeldkwaliteit / digitale televisietechniek.
Subject headings: video coding / multiprocessing systems / quality of service / digital signal processing chips.

Performance and QoS-aware MPEG-4 video-object coding for multiprocessor architecture

Milan Paštrnák

Committee members:

prof.dr.ir. P.H.N. de With (TU Eindhoven, prommoter)
prof.dr.ir. J.L. van Meerbergen (TU Eindhoven, prommoter)
prof.dr.ir. H. Corporaal (TU Eindhoven)
prof.dr.ir. R.L. Lagendijk (TU Delft)
prof.dr.ing. P. Pirsch (Leibniz University Hannover)
prof.dr.ir. H.J. Sips (TU Delft)
prof.dr.ir. A.J Vinck (University of Essen)

Marie Curie Fellowships

The research work reported in this dissertation was supported by European Union via the Marie Curie Fellowship program under the project number HPMI-CT-2001-00150.

Cover design: Bregje Schoffelen
Printing: Printservice Technische Universiteit Eindhoven

Acknowledgements

In 2002, around the time I was finalizing my studies at OOTI Program, I was certain about knowing that my academic life was not yet over; and the blurry ideas I had during my OOTI studies about becoming a PhD student happened to be more genuine after talking to, nowadays my promoter, Prof.dr. Peter de With.

This research and all the work conducted within this thesis would not have been possible without Peter arranging the Marry Curie Fellowship program and, together with Dr. Gert-Jan van Dijk and Ir. Peter Hupperetz, creating a PhD position for me within LogicaCMG Nederland. Also, the transition from student life into a working one would not have been as smooth without the mentorship of Gert-Jan, who helped me to understand the working style in a large, Dutch company.

Therefore, I would like to express my gratitude to Dr. Gert-Jan van Dijk and Ir. Peter Huppertz, but above all, I would like to thank to my promoter, Prof.dr. Peter de With, for giving me the opportunity to conduct my PhD research under his supervision at Eindhoven University of Technology. Throughout all these years, Peter, with his great amount of energy, was playing the key role in both, the scientific and managerial supervision of my research work. As my promoter, Peter had given me much advice as well as encouragement throughout all these years.

Also, I would like to express my sincere appreciation to my second promoter, Prof.dr. Jef van Meerbergen, for helping me throughout my transition period from computer science domain into the world of design of multimedia embedded systems, as well as for his guidance during my stay at Philips Research, where I have spent half of my research time in the cooperation with members of the Hijdra project group.

Within the Hijdra group, my special thanks go to Ir. Peter Poplavko, who helped me in my understanding of clock-cycle-true simulations, the debugging of software, and the mapping of the decoder on the target platform simulator. I am also appreciative to Dr. Marco Bekooij, Dr. Bart Mesman, Dr. Sander Stuijk, Ir. Calin Ciordas and all members of this group for having the opportunity to participate in their delighted and motivating discussions.

Additionally, I would like thank Dr. Dirk Farin for his help at the beginning stage of decoder development and for the time we had spent on interesting discussions about Linux and music. Also, many thanks to all my colleagues from Video Coding and Architectures group, with whom I had spent many enjoyable moments.

My further regards are expressed to my current colleagues at Philips Research, especially for their forbearance while I was finishing my thesis.

I would like to thank the promotion committee members for reviewing this thesis; especially to Prof.dr. Peter de With, for a very thorough review, and Prof.dr. Inald Lagendijk, for providing me with useful comments on the draft version.

Also, I would like to faithfully admit that without the love and care of my parents and other members of my family, this work would be never where it is now. Last, but definitely not least, I would like to thank my wife Eva for her love, patience, sacrifice, and endless support.

Contents

Chapter 1

Introduction

This chapter provides an outline of the thesis and briefly introduces the research scope and contributions. The chapter commences with outlining the trends in multimedia coding. Afterwards, a similar discussion follows on computing platforms. The third part presents the research scope and background: i.e. predictable mapping and Quality-of-Service management. The fourth section summarizes the research contributions. This introductory chapter concludes with an overview of the individual succeeding chapters, indicating the relevant publications and contributions of the author to this thesis.

1.1 Pervasive multimedia coding

The concept of Digital TeleVision (DTV) was first introduced in the early 1990-ties. On one hand, the broadcasting in digital form improves the robustness of the transmission against noise, while on the other hand, the digital representation of three color components forming the full-color image signal results in a huge data expansion. Unfortunately, the transmission of three digital baseband signal components sampled at a decent video frequency requires a large bandwidth. For this reason, *compression* is one of the most important technology components of DTV and other forms of digital video communication. The nowadays proven concept of DTV would be impossible without a compression algorithm such as MPEG-2 and the related DVB standard. The compression is also of great value for storage systems as it increases the effective capacities of magnetic hard disk storage and optical disc media, such as Digital Versatile Disk (DVD) and its latest successors Blue-ray disk and HD-DVD. The compression also plays the dominant role in portable devices, like mobile phones

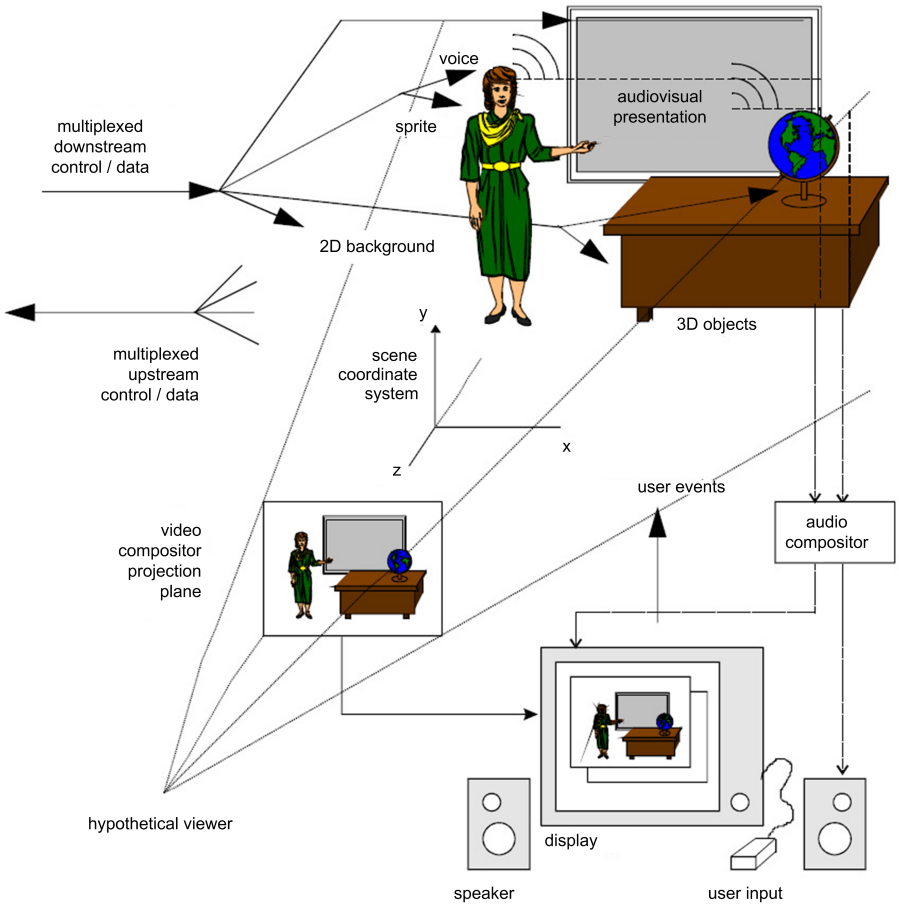


Figure 1.1: *MPEG-4 compositional view [60].*

and PDAs. One of the latest multimedia devices is the iPhone[1] that is based on the H.264 compression standard for video content. The bandwidth limitation of the Internet motivated the development of the MPEG-4 Simple Profile and the related DivX standard. Video playback over the Internet is nowadays based on a multitude of commercial and freeware players, e.g. Flash, VLC, Real, Quick Time, K-Lite, which support a variety of compression standards and resolutions, up to high-definition H.264 and AAC coding for audio.

In contrast with the traditional video processing using rectangular frames with video information, this thesis studies the implementation of *object-based coding* of video signals. The novelty of the object-based video processing is that it

[1] iPhone is a registered trademark of Apple Inc.

considers the video signal to be a collection of individual objects in front of a scene background. For compression, those objects are individually coded. At the receiving side, the resulting video scene is reconstructed by parallel processing of the independent video-object (VO) decoders. Besides this parallel computing, the decoding results have to be buffered and combined with the reconstructed scene background (rendering). Figure 1.1 outlines a typical composition of a video scene with synthetic and natural visual and audio objects. For more details, the reader is referred to Chapter 2.

We have decided to focus on the Core Profile of the MPEG-4 standard that specifies the syntax and usage of several audio-visual decoding tools [101], like arbitrary-shaped decoding and other tools for support of non-rectangular video data[2]. Our motivation for using object-based coding is that it poses interesting new requirements on e.g. buffering and dynamism within the system design. Besides this, it simply was one of the latest standards available at the start of the research work.

Object-based coding itself provides a broad potential for future applications, due to to the build-in *interactivity* and *compositionality*. However, the dynamism and the buffering complicates the system design of full object-based video processing. Let us now briefly discuss the involved major aspects.

1. Complexity
The complexity of object-based MPEG-4 video processing occurs at several levels. First, the block-based Motion-Compensated (MC) Discrete Cosine Transform (DCT) decoder comprises both temporal and spatial decoding techniques. The decoding complexity is growing with the number of video object decoders. Second, for real-time applications, those decoders have to produce decoded data within a specified time interval. Third, object-based coding combines the traditional texture compression techniques with shape coding that is addressing the compression of the arbitrary-shaped contour information of the moving video objects. The experiments in this thesis have resulted in a quantification about the relative complexity of the individual decoding tasks and tools. This complexity comparison is provided in Chapter 3, where all decoding algorithms have been explained to the reader and first experiments on complexity are conducted. From that explanation, it will become clear that decoding of shape information with motion compensation consumes a comparable amount of clock cycles as MC-DCT decoding for the contents of the object. This makes AS VO MPEG-4 decoding clearly more complex than MPEG-2 decoding and comparable to H.264 decoding in the order of magnitude.

[2]The Core Profile of MPEG-4 should not be mixed with MPEG-4 AVC / H.264 coding for HDTV applications and next generation DVD recording.

2. Dynamism
Object-based video is more dynamic in behavior than conventional video processing. This is because, the number of video objects per scene is varying as new objects appear and others vanish. Moreover, the size of objects is variable over time, e.g. as they move closer to the camera or move away from it. Last but not least, each video object consists of different block types such as boundary blocks, texture blocks and transparent blocks.

3. Scalability and associated QoS
Irrespective of video objects, multimedia compression can be made scalable in quality and/or computing, because in many applications it is desired to use the most inexpensive platforms for this kind of processing. If the computing requirements are variable, they sometimes may exceed the capabilities of the chosen platform. In such cases, the computing could be downscaled at the expense of some quality to continue processing without interruptions. The subject of scalable video algorithms was addressed in earlier research [47, 112]. This work concentrated on the scalability for transmission, or closed encoder-decoder chains. In this thesis, we pursue a scalability concept that enables various levels of processing in the decoding terminal without effecting the encoding and transmission of the video. Furthermore, for the multiprocessor architecture, the scalability can be considered for all three primary resources: computing, communication and memory.

The sequel of this chapter is as follows. In Section 1.2, we summarize various examples of platforms for multimedia processing. Section 1.3 formulates the problem statement and research questions. In Section 1.4, we outline the conducted research and list the contributions. Section 1.5 presents the structure of the thesis content and the relation to scientific publications of the author.

1.2 Platforms and trends

System-on-Chip (SoC) represents an evolving paradigm for the design of Integrated Circuits (ICs). The evolution of ICs enables that integration grows from simple electronic systems to the nowadays complex, multi-functional devices as illustrated in Figure 1.2. The simple SoCs from the past were based on Application-Specific Integrated Circuits (ASICs), addressing specific application domains, e.g. the sensor system or A/D converters. With the growing complexity of multimedia applications, it has become logical that the I/O subsystems, hierarchical memory, coprocessor(s) and main computational unit(s) have to be integrated also. Besides the aforementioned computing units, also a communication infrastructure is needed. In the first SoCs, this issue was

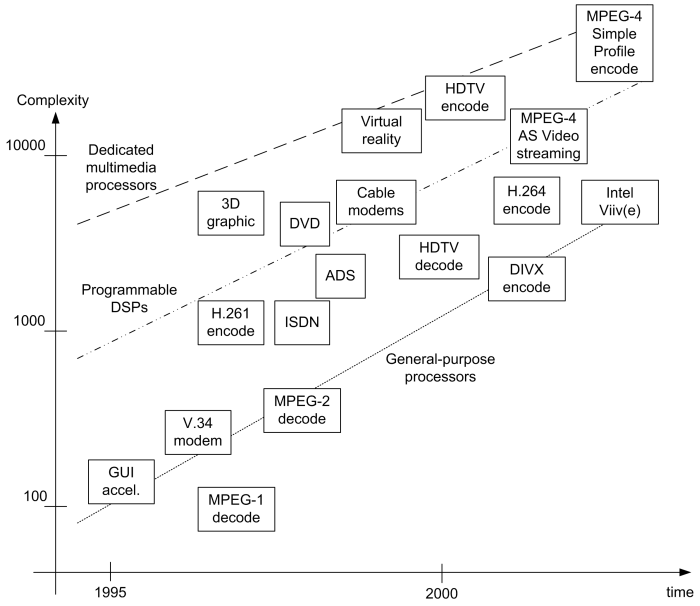


Figure 1.2: *Various multimedia functions and the current trends in computing cores.*

addressed by several interconnect mechanisms with busses as the preferred communication system.

At the high integration densities of modern process technologies, the interconnection design has become the most critical step in the whole design process. Consequently, the design of on-chip communication for complex SoCs is a key to the overall system performance [27]. To avoid lengthy and complex trajectories for on-chip communication, a popular approach is to integrate a set of processors in a *networked* fashion on a single chip. Therefore, the Network-on-Chip (NoC) is an evolutionary step from the bus-based architecture to a more concurrent communicating processor system. Let us discuss some examples of multiprocessor systems from the past decade.

The distributed computing on multiprocessor platforms was observed as one of the most energy-saving approaches for extensive computing. This problem was already studied in the mid-nineties in a project targeting an experimental Television Processor(TVP). The project has resulted in a Coprocessor Array (CPA) and a Telecommunication and Control Processor (TCP)[3] [56]. Other researchers worked on similar concepts [12].

[3]This IC was commercially produced by Philips Electronics under the name SAA7430.

The OMAP platform [25] aims at real-time processing and low-power consumption on a heterogenous multiprocessor architecture. The OMAP design separates the execution of general-purpose tasks on an ARM9 core and a TI c55x DSP for high efficiency of real-time signal processing tasks. The interest for multi-core systems becomes apparent in the gaming market as well. The latest products, such as the XBox 360 with a triple-core, PlayStation 2 with the Emotion Engine or PlayStation 3 based on the CELL processor clearly show the trend for multiprocessor architectures. The integration of dedicated DSPs together with a RISC core on one chip is available as e.g. TMS DaVinci processors with TMS320C4 and ARM926. Similarly for multimedia, an example architecture is the Viper [34], which is based on a dedicated TriMedia processor together with a 32-bits RISC core. The Cake platform [113] is an an example of an architecture focusing on low-cost consumer video and audio. It combines VLIW cores, CPUs, and accelerators, which are connected to a memory.

NEXPERIA [28] forms a more general heterogeneous programmable system offering an integrated, programmable system-on-chip (SoC) and companion ICs, and it includes reference designs, system software, and development tools. A similar, but homogeneous concept of multiple cores integrated on a single chip for personal computers, was shown in the form of a working processor with 80 identical cores, delivering more than 1 trillion floating-point operations per second (teraflops). These examples prove the attractiveness of the multiprocessor concept for future systems. Even general-purpose computers such as PCs have all kinds of integrated specialized processors to perform dedicated tasks on communication, memory control, disk control, graphics operation, simply because it is more efficient.

1.3 Research scope and background

Object-based video processing requires a simultaneous execution of the individual video-object decoders. It is difficult to provide more functions and create efficient architectures without several Application-Specific Integrated Circuits (ASICs). The efficient mapping of the advanced video applications on a plurality of individual processing cores, communicating with each other and integrated on one chip, is a complicated problem. The designer wants to have an estimate of the system behavior and resource usage at the time of executing all involved tasks. This points to the desire for a *predictable mapping* of the multiple multimedia applications onto such a chip. Furthermore, the predicability is highly preferred for processing multiple video objects with real-time requirements, because of the dynamical behavior of these objects.

The simultaneous execution of multiple video applications in a system, while aiming at real-time performance of that system needs a resource management system. This management can ensure that each application meets its deadline. Critical situations occur when not all tasks can meet their deadlines, because sufficient computing power is not available. The only remaining solution is to reduce the computing effort or other resource requirements to speedup the execution. This reduction of computations will inevitably lead to a degradation of the output quality of applications. When optimizing the resource usage in combination with the best quality, applications should be controllable with respect to their computing requirements (and thus quality). This type of control is known as *Quality-of-Service* management, which selects the presented quality output based on the availability of system resources (computations, memory, bandwidth). The research scope of this thesis is on the intersection of these domains: object-based video coding and its predictable mapping onto and Quality-of-Service for a multiprocessor SoC. In the sequel, a more detailed view on these issues is provided.

1.3.1 Predictable mapping and timing models

The problem definition of a predictable design of MP-NoC systems is split in three aspects. First, the design of an MP-NoC for a set of parallel applications with dynamic resource usage is so complicated that the construction of an accurate processor model coping with those tasks is indispensable. Second, we see a trend towards more dynamism in advanced multimedia applications. This trend originates from the dependency of the processing on the input data and the type of processing that is applied. As a consequence, the conventional way of a worst-case design of an MP-NoC is becoming increasingly overdimensioned and thus inefficient. Instead, we have to pursue flexibility in on/off switching of the processing tasks. Third, to handle the dynamic properties of tasks and facilitate their predictability, we aim at making an accurate model of the task execution capturing the dynamic behavior as closely as possible. Let us discuss these aspects in some more detail.

1. Covering dynamism in an application
The dynamism in recent applications comes from the algorithm and input data. Therefore, the model of an application should be able to cover and emulate the possible dynamic behavior. In our work, we also consider the input data characteristics and take them into account in the execution and communication model. In this thesis, we have adopted MPEG-4 object-based coding as an example of an application with a high level of dynamism. Certainly, video objects change more rapidly over time than rectangular video frames. The use of video objects is common in the computing and Internet environment al-

though some of those objects are created in different fashion. With the advent of 3D TV and the integration of synthetic video objects into natural scenes, we expect that the usage of objects will also grow in the multimedia domain. As a conclusion, we will develop application models that depend on the input data (or other key features), like object size, so that the models will closely represent the actual execution.

2. Application model development
A commonly used model for real-time audio and video applications is the Synchronous Data Flow (SDF) graph. The SDF graph is well studied for mapping DSP applications onto multiprocessor architectures (e.g. see the TV application in [29]). Many techniques to analyze and map such applications have been proposed [48][106]. The SDF graph allows the analysis of a complex application partitioned into a set of communicating tasks. The buffering between the tasks and the strict communication rules enable mapping of tasks on individual processors without loss of generality. Furthermore, exploring the task granularity of an application directly leads to quickly generating various mappings onto an MP-NoC, which gives a speedup of the exploration of the design space. A natural split of the communication and computation parts allows a more accurate analysis of the overall system timing behavior. Hence, the approach in our research is to model applications in the form of SDF graphs and partition them into communicating tasks.

3. Processor model development
The MP-NoC processor model has to accurately correspond with the execution of the application on the target system. In our approach we deploy a tile-based architecture that separates the processing elements from the communication features. The predictable mapping paradigm requires that the execution on the system would match with the prediction made at design time. Therefore, we aim at building application models based on the execution on clock-cycle-true simulators of target processor tiles, and we assume that task scheduling has predictable characteristics, e.g. Time Division Multiple Access (TDMA). Furthermore, the communication model is based on guaranteed task throughput: once a data connection is established, the data delivery is always guaranteed. In the next subsection, the processor model will be extended with monitoring features for a Quality-of-Service control.

1.3.2 QoS on multiprocessor platforms

Quality-of-Service (QoS) management is a very popular technique in mainly two domains: computer networks and multimedia systems. In the area of networks, QoS is applied in the transport layer of the communication hierarchy. Examples are video conferencing, video on demand and similar multimedia

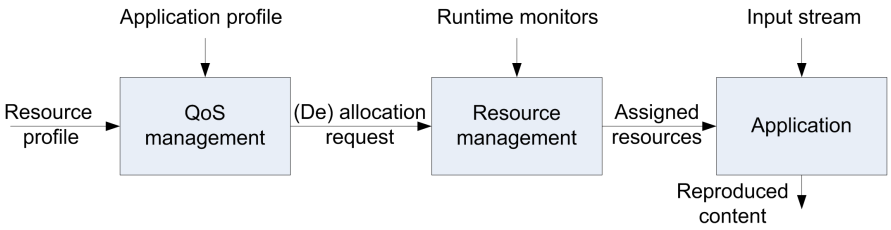


Figure 1.3: *General QoS framework for a multimedia system with runtime control.*

applications that have to deliver their services using non-predictable media. The non-predictable nature requires the persistent monitoring of the network performance and the prioritization of a subset of network data tokens in the transport mechanism [71, 123, 21]. In contrast with networked QoS, multimedia runtime systems focus more on the optimal allocation of the system resources among a set of applications active in the system. Due to the complexity of resource management in a multitasking system, most of the existing approaches concentrate on one, mostly computational resource of the platform [18, 63]. We concentrate on the second QoS domain, i.e. QoS control in multimedia systems.

A general concept of a Quality-of-Service management is illustrated in Figure 1.3. First, a detailed model of computation of a scalable application is used for effective management of the QoS. Second, in the combination with the profile of available resources, the QoS requests the resources from the platform management. If the resources can be allocated to the application, a fixed reservation is performed. Third, to improve efficiency, we explore the addition of best-effort computing on top of fixed-reservation services.

1. Scalable application model
The AS VO MPEG-4 decoding is defined as a sequential process. This process can be controlled by a QoS system when the application becomes scalable in computation and output quality. Since it is modeled as an SDF graph with various communicating tasks, we will explore the scalability at level of task switching. In this way, we do not have to rewrite the whole application for designing a scalable coding algorithm. The objective is to implement scalability without the obligation to use the scalable video-coding profile of MPEG-4 to operate with standard non-scalable streams.

2. Monitoring of resources
The platform with integrated QoS should provide means for the runtime allocation of resources and the resource-usage monitoring. The monitoring of

computation resources requires an extension of the computation-scheduling algorithms at individual processing tiles. For the monitoring of communication resources, the platform-integrated NoC probes initially used for collecting debug information will be tuned for the runtime monitoring of communication-link utilization.

3. Best-effort adaptation
As a fixed reservation-based approach ensures a predictable mapping due to its strict resource-usage rules, the consequence is a low utilization of the reserved resources resulting from over-allocation. An improvement in the utilization of resources can potentially increase the video output quality. This is achieved by activating some of the idle processing tasks via best-effort services.

1.4 Conducted research and contributions

1.4.1 Research objectives

The research topic of this thesis is to study the mapping of MPEG-4 video-object coding on multiprocessor architectures while keeping track of the resource usage and enabling QoS control. Given the results of the scope discussion in the previous section, we formulate the following research objectives.

1. AS VO MPEG-4 decoding model
This involves the design of an MPEG-4 application model for arbitrary-shaped objects, that incorporates the previously discussed dynamic behavior with sufficient accuracy. The currently used Synchronous Data Flow models cannot capture the application dynamism and therefore an extension of the SDF models is required.

2. Quality-of-Service
This requires the design of a QoS framework that handles the decoding of multiple dynamic video objects as a parallel application. This involves a hierarchical control of the system and applications themselves. Scalability is achieved by task switching within the MPEG-4 decoding applications.

3. Best-effort computing
To improve efficiency, best-effort computing is deployed for idle tasks that are not covered by the reservation-based approach.

The above-listed objectives are addressed independently and inter-dependently: as a stand-alone problem and as a joint problem in which all aspects play a role simultaneously. The remaining part of this section summarizes our contributions for each objective presented above.

1.4.2 Research contributions

1. Parametrical timing models
Our major contribution towards the first objective is in introducing *parametrical models* describing the execution of MPEG-4 video coding in high detail. Instead of a single-valued performance metric, we propose a *linear parametrical function* that is added to the Synchronous DataFlow (SDF) model that describes the dynamism at the task level. Two key elements play a role for accurate modeling: input data dependency and the characteristics of the target processor of the multiprocessor system.

- A set of linear equations describing the required computational resources for AS VO MPEG-4 decoding with an accuracy above 90%.
- The second contribution is a similar linear parametrical model for the usage of communication resources involved in the MPEG-4 coding application.
- A concept for combining the previous individual models per resource into one multidimensional model to cover the usage of different resource types in the complete MP-NoC.

2. Hierarchical QoS system
Given the multiple applications running in parallel, we have adopted a layered hierarchical QoS system that controls the average quality of all applications executed at the system while controlling the individual applications simultaneously. The new hierarchical QoS management employs the estimation of resource requirements derived from the above-discussed timing models. Besides the control aspects, the hierarchy supports the modularity and compositionality of the system.

- A hierarchical QoS system architecture for a multiprocessor system interpreting the actual resource usage measured inside the NoC.
- A scalable model of computation based on activation/deactivation of non-essential tasks within the AS VO MPEG-4 application.
- A heuristic algorithm for negotiating the assignment of available resources between Global and Local QoS managers.

3. Best-effort computing
We address a combined solution for the control of a computation by extending the guaranteed-throughput services with best-effort computing. The strategy of this combination serves in cases when the highest quality-level cannot be assigned to all tasks in parallel.

- An algorithm in which the most important tasks are assigned to guaranteed throughput to ensure their quality, whereas the remainder of the tasks are assigned to the best-effort computing so that the overall system efficiency is optimized.

Besides the above principal categories, the research has resulted in additional contributions.

- In order to conduct the research in this thesis, we developed a fully compliant AS VO MPEG-4 decoder in such a way that it could be used for time-controlled pipelined execution of application tasks and explorations of *task-level parallelism, data-level parallelism.*
- An alternative algorithm for extended padding and post-processing tasks in the MPEG-4 decoder to perform processing at macroblock level in order to minimize the required buffering and enable pipelined processing.
- An alternative algorithm for MPEG-4 sprite decoding featuring random-access to coded macroblock data that enables distributed data-level processing over various processors.

1.5 Thesis organization and scientific background

The core part of the thesis is structured into two major parts based on the primary contributing steps to the final architecture, and an overall system contribution. Chapters 2, 3, and 4 contain a detailed view on the video-coding mapping issues required by a predictable mapping and our parametrical performance modeling. Chapters 5 and 6 detail our concept for Quality-of-Service management and a connection with the multimedia applications. Figure 1.4 depicts the above-mentioned chapters that are individually outlined below.

Chapter 1 addresses the background on the desired realization of multiprocessor systems for executing recent multimedia standards and motivate the presented research. We introduce a predictable mapping paradigm for multiprocessor NoC systems. Afterwards, we present a framework for a hierarchical Quality-of-Service resource management. The chapter discusses the research issues and our contributions and concludes with an outline of the thesis.

Chapter 2 presents an overview of the MPEG-4 video coding standard, in particular the details of the arbitrary-shaped video object decoding tools. In the second half, a multiprocessor platform is defined as a target architecture. The system concept was first presented at the 4th Symp. On Embedded Systems

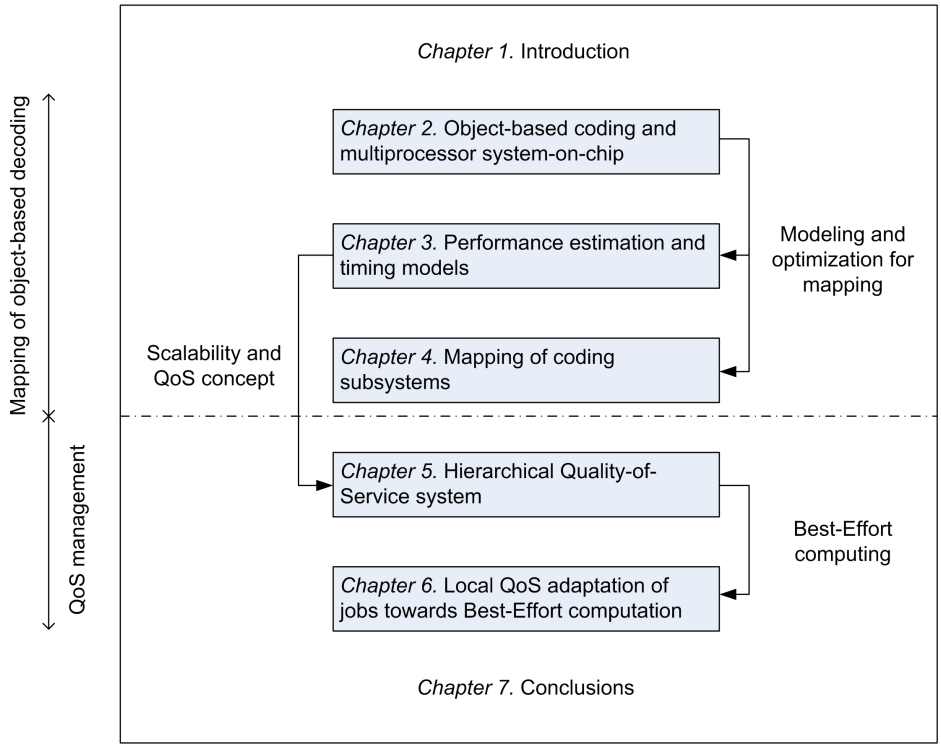


Figure 1.4: *Thesis structure indicating also chapter dependencies.*

(PROGRESS 2003) [90] and at the Workshop On the Design of Multimedia Architectures (MMA 2003) [97].

Chapter 3 introduces a solution for performance estimation and its importance for predictable mapping. The predictable design of embedded systems were reported at the 8th Int. Workshop on Software and Compilers for Embedded Systems (SCOPES2004) [8]. The extended version focusing on dataflow analysis was published in a book chapter of the book *Dynamic and Robust Streaming in and between Connected Consumer-Electronic Devices* [7]. The dynamic characteristics of the computational requirements motivate the usage of linear parametrical functions in data-flow models as presented at the 14th Workshop on Circuits, Integrated Systems and Signal Processing (ProRisc) [98]. Details obtained by further exploration of our application and MP-NoC were presented at the 4th IEEE Int. Workshop on System-on-Chip for Real-Time Applications (SoCRT) [89]. The parametrical modeling for communication resources was presented at the 25th Symp. on Information Theory in the Benelux [82] and further elaborated for both intra and inter-coded video frames at the 9th IEEE Int. Symp. on Consumer Electronics (ISCE) [81]. Multidimensional

parametrical models of the partitioned AS VO MPEG-4 decoder and MP-NoC platform resource modeling were presented at the 27th Symp. on Information Theory in the Benelux [83].

Chapter 4 discusses the specific mappings on architectures that are constrained in communication resources, granularity or the internal memory size. First part focuses on the optimal utilization and balanced traffic between cores can be achieved by partitioning the application to tasks with the same granularity as the processing granularity. The second part explores the mapping of background sprite decoding algorithm on the CELL processor. The new MPEG-4 background sprite decoding was presented at the 26th Symp. on Information Theory in the Benelux [88]. The adaptation of the AS VO MPEG-4 decoder towards the same granularity level of the communication was given at the SPIE Visual Communications and Image Processing 2006 (VCIP 2006) [85].

Chapter 5 deals with the scalability of the AS VO MPEG-4 decoder and the hierarchical Quality-of-Service (QoS) management framework. Our concept of two management layers which are communicating with each other and finding a balance in the negotiation on resources was presented at the Workshop On Resource Management for Media Processing in Networked Embedded Systems [91]. The task-level scalability was published in the proceedings of the 9th IEEE Int. Symp. on Consumer Electronics (ISCE). The high-layer QoS control was presented at the IEEE Int. Symp. on Circuits and Systems in 2006 (ISCAS) [87].

Chapter 6 concentrates on combining the reservation-based technique for guaranteed throughput with runtime monitoring to handle a non-optimal resource allocation by best-effort computing. This explored concept results in a performance-scalable MPEG-4 decoding application. It is shown that in about 80% of frame-based tasks, the activation of higher quality-level processing was obtained compared to the pure reservation-based approach. The results were published at the 10th IEEE Int. Symp. on Consumer Electronics (ISCE) [84] and obtained a best paper award. The extended version of this work was published in the IEEE Transactions on Consumer Electronics 2006 [86].

Chapter 7 concludes the research presented in this thesis and provides the suggestions for future work. It also briefly presents the results of the decoding application on a commercial mobile device.

Chapter 2

Object-based coding and multiprocessor system-on-chip

The objective of this chapter is to first give a brief overview of the arbitrary-shaped MPEG-4 decoding algorithm (Sections 2.2 - 2.5) . Individual compression steps are discussed, with the emphasis on the object-oriented nature of the processing, since this is different from MPEG-2 and MPEG-4 AVC processing. The second half (Sections 2.6 - 2.8) of this chapter is devoted to architecture details of the target multiprocessor platform with a focus on on-chip communication and the tile-based processing of the system.

2.1 Introduction

The mapping of a streaming multimedia application to a multiprocessor Network-on-Chip (NoC) requires several design steps that lead to an MP-SoC configuration, satisfying the performance and throughput constraint of the application. In general, Figure 2.1 illustrates the arbitrary-shaped MPEG-4 decoder as an intrinsically interesting and typical multimedia application to be mapped onto a multiprocessor Network-on-Chip as the target platform. The arbitrary-shaped MPEG-4 decoding was chosen as a representative recent sample out of a multimedia standard that has shown continuous growth in complexity with an ever increasing number of profiles and levels and it involves variable amount of resource requirements. The platform choice for a multiprocessor Network-on-Chip follows the natural evolution in the design of System-on-Chips. This trend to go for multiprocessor solutions occurs because a single core system

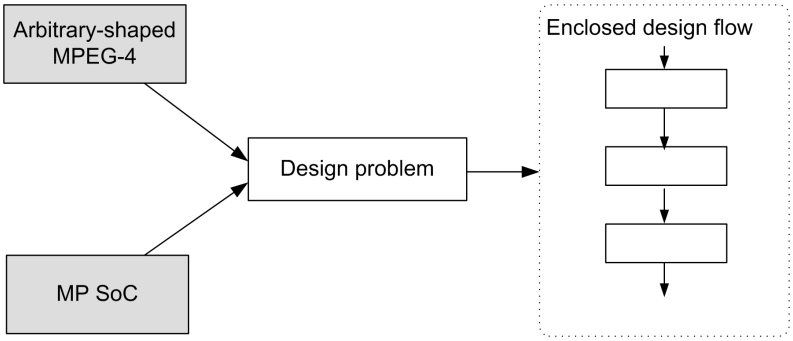


Figure 2.1: *General structure of the target system design.*

cannot easily increase the CPU clock frequency, while the above-mentioned type of application still requires a growing amount of computational resources.

This chapter provides details on both the selected application and the platform as it is essential to understand the aspects that motivate the research presented in succeeding chapters. The details of the decoding process are presented in (Sections 2.2 - 2.5). Readers familiar with the decoding of the arbitrary-shaped video objects can skip this part[1]. The individual decoding steps of arbitrary-shaped MPEG-4 decoding are presented in very detailed form, because they are the fundamentals of the work presented through the whole thesis[2]. The background sprite reconstruction is presented briefly, because details and work based on this technique are discussed only in Chapter 4. The second part of the current chapter (Sections 2.6 - 2.8) introduces the multiprocessor Network-on-Chip as a target platform for the execution of the arbitrary-shaped MPEG-4 decoder application. The design flow involved with the mapping and as visualized in Figure 2.1, is discussed in Section 2.9 in more detail.

2.2 Principles of object-based video

The concept of object-based video processing and the system aspects of a corresponding video codec based on such a concept are an essential feature of the presented research. Compared to traditional video coding, MPEG-4 defines the new concept of object-based video coding, in which foreground objects are segmented from the scene and coded as individual image objects. This means

[1] Even if we adopt a profile that overlaps with MPEG-2, the arbitrary-shaped MPEG-4 decoder contains many new functions.

[2] At the project start, there was no software available. Therefore, the author had to design the software of the arbitrary-shaped MPEG-4 decoder himself. The development of a standard-compliant and validated arbitrary-shaped video object MPEG-4 decoder is a notable contribution already.

(a) Segmented VO. (b) Original stream.

(c) Artificial rendering. (d) Rendering of several VOs.

Figure 2.2: *An example of video objects and scene composition in MPEG-4.*

that not only the texture and motion of the object is coded, but also the *shape* of the object.

The MPEG-4 Core Profile[3] provides the coding techniques for video object-based coding[4] to provide an interactive usage of individual objects originated by different sources and the combination of natural and synthetic video signals in one scene. This means that objects can be taken from different sources in the network.

The main differentiation with the previously successful standards like H.263 or

[3]Since this thesis is fully focused on arbitrary-shaped coding, we mean with MPEG-4 always this profile or related profiles that support arbitrary-shaped objects and the coding of them.

[4]In this thesis, with *objects* we mean video objects unless otherwise stated. There is no relation to object-oriented software or programming.

MPEG-1/2 is in aiming at *content-based coding* [94]. The initial thought was that it would be better to code each object within a scene with its own coding tools and settings so that the optimal efficiency would be achieved. The establishment of the standard was time-aligned with the Internet hype at the end of the previous millennium, where Internet was seen as the major driver and content provider for video material and video objects (VO). Probably, this will come true, but later than expected and in a different form from which it was initially designed for [99]. Despite the success of the MPEG-4 AVC substandard for the next generation of DVD, content- and object-based processing is recently again emerging [61] for exploring lower bitrate coding. This is visualized in the following example.

Figure 2.2 portrays the example usage of video objects in realistic and artificial scenes. The news reader Akiyo, shown at the top left was extracted from the original sequence at the top right. At the bottom left, the news reader is inserted in a laboratory environment, which seems a quite realistic scenario. The bottom-right picture shows an example of a more complex scene composition with the same object and inserted graphics. The user can disable the foreground object in order to see the part of the video sequence that was originally hidden behind the foreground object. Such a feature is useful if the foreground object covers a substantial part of the scene.

2.3 Object-based data reception in MPEG-4

For MPEG-4 decoding, each video object can be found in the elementary stream containing the compressed video-object data. The correct positioning and scaling of the object is done by the *composition editor* (see Figure 2.4) at receiver side. There is a special stream for describing the scene composition, called Binary Format for Scenes data (BIFS). BIFS controls the position of individual video objects, and background usage and any control parameter that is needed for the scene composition.

The scene graph within the BIFS and object-oriented stream-based setup are visualized in Figure 2.3. All decoded scenes are described in a scene-graph, which is a hierarchical representation of audio, video and graphical objects, each represented by a node abstracting the interfaces to those objects [51]). The concept of separating foreground objects from a background image and individual object manipulation requires that each object is individually coded and transmitted. The background of the scene is considered as a separate video object and the coding of it is called *sprite coding*.

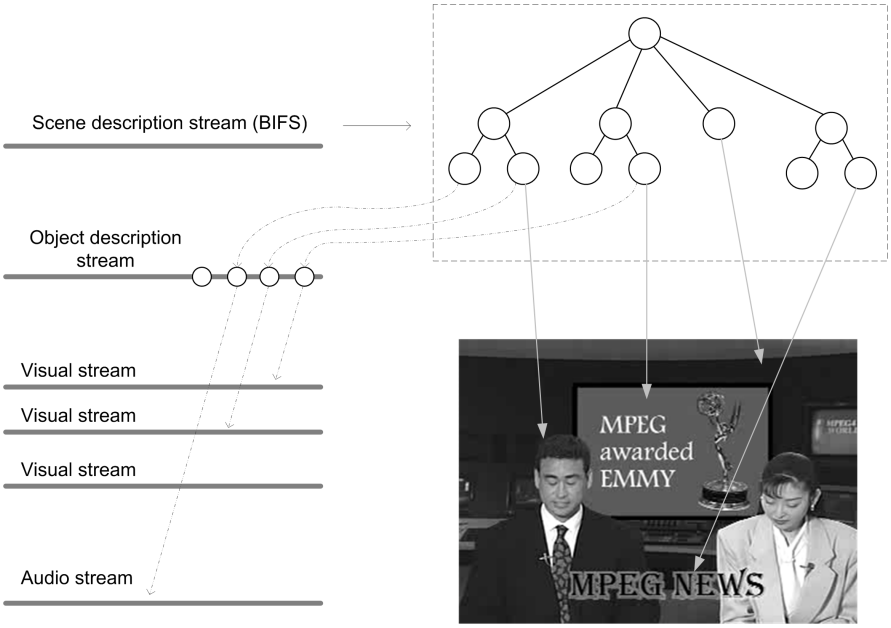


Figure 2.3: *MPEG-4 scene graph and composition of different AV elementary streams.*

An object stream starts with a visual object sequence header, which is then split in subheaders for the individual objects and followed by the elementary bitstreams. This together forms a hierarchical data structure with headers and layers. The MPEG-4 standard allows both: the separate coding of configuration data and combined configuration with elementary bitstream. In Appendix A further clarification of details and the transmission structure is provided.

The compression of object data at the encoder and decompression of those objects at the decoder is achieved by employing the coding system as depicted in Figure 2.4. The extraction of the VO definition, like segmentation of shape, texture data, the frame type and corresponding tools are selected before defining the scene and encoding individual objects into elementary streams. These streams are combined into a multiplexed stream that is delivered to the receiver. In our work, we focus on the decoding of arbitrary-shaped video objects and the decoding of the background image sprite (see the bottom half of Figure 2.4).

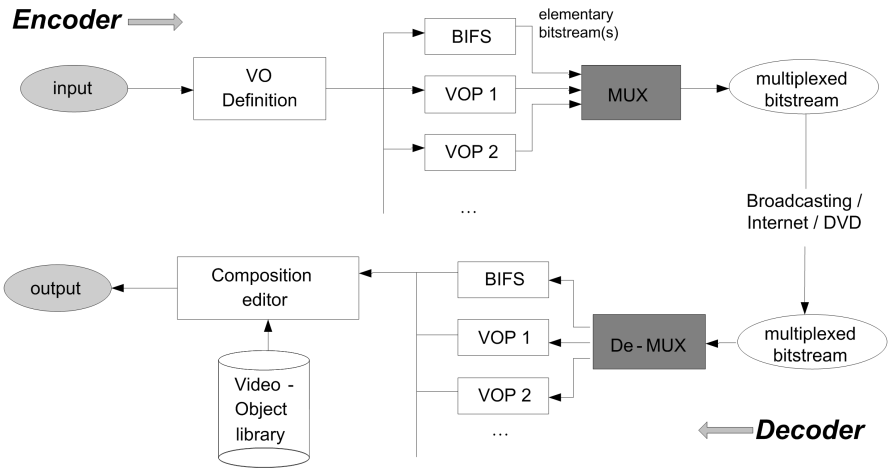


Figure 2.4: *Structure of the object-based MPEG-4 codec.*

2.4 Arbitrary-shaped objects decoding in MPEG-4

2.4.1 Video objects and VOP planes

For the MPEG-4 decoding of Arbitrary-Shaped (AS) video objects, we focus on the tools in the standard that support this feature. Every Video Object (VO) is represented in several information layers, with the Video Object Plane (VOP) at the base layer. The moving behavior of video objects is captured with Video object Sequences (VS). A video object sequence is time-sampled series of consecutive VOPs.

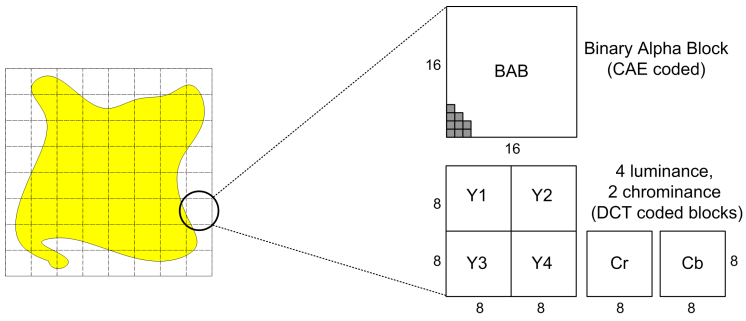


Figure 2.5: *Macroblock representation of a video object within a VOP.*

The position and texture data of a video object is captured by a rectangular area, called VOP bounding box. This box is composed of a grid of a 16×16 sample blocks. The texture information is enclosed by the four 8×8 blocks

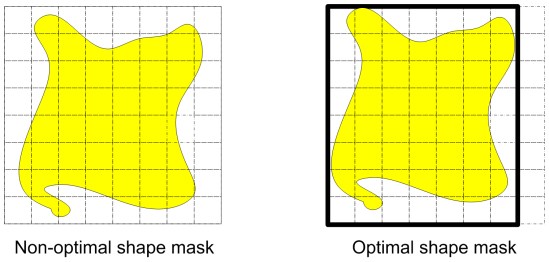


Figure 2.6: *In MPEG-4 coding, the VOP bounding box is always positioned such that the amount of macroblocks is minimal. The left image requires 64 MBs, the right image, which is preferred, requires only 56 MBs.*

of luminance data and the two corresponding 8×8 blocks of chrominance data (for 4:2:0 chrominance format). The six 8×8 blocks forming the colored texture is called a MacroBlock (MB). The shape information is stored in a collocated Binary Alpha Block (BAB) of 16×16 binary values indicating opacity or transparency of the texture pixels. Figure 2.5 depicts an artificial video object, which is described as a rectangular area of MBs and BABs.

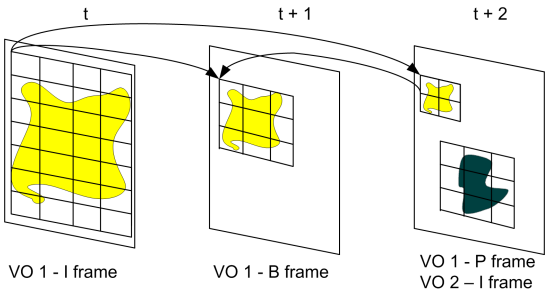


Figure 2.7: *Moving video object(s) in an MPEG-4 scene.*

Prior to the encoding of a video object, the bounding box has to be optimally positioned (see Figure 2.6), in order to minimize the block overhead. Due to the fact that video objects change in size and position over time, the bounding box sizes and positions change accordingly.

In the MPEG-4 standard, each scene is composed and rendered from independent VOs. An intra-coded VOP can coexist with an inter-coded VOP as illustrated in the most right picture from Figure 2.7. MPEG-4 supports predictive and bidirectional predictive coding of VOPs (P-VOP and B-VOP frames).

2.4.2 Decoding process of AS VOP

The decoding process of an arbitrary-shaped VO object is portrayed in Figure 2.8. The shaded box represents the texture decoding that is inherited from the MPEG-2 standard. The new elements in the decoder are shape decoder, inverse coefficient prediction and a backward loop based on reconstructed VOPs.

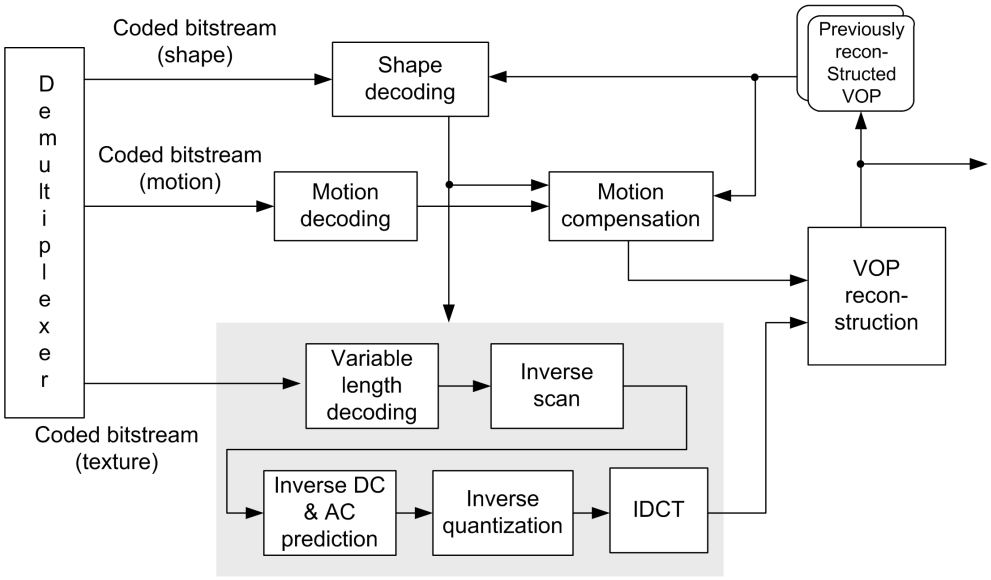


Figure 2.8: *Block diagram of an arbitrary-shaped object MPEG-4 video decoder.*

The MPEG-4 standard includes two algorithms for encoding the shape information. The first algorithm is Shape-Adaptive DCT (SA-DCT), that provides the same number of transform coefficients as the number of pixels enclosed by the VO part in the pixel block. However, it has several drawbacks [58], such as the non-orthogonality and the mean weighting defect, so that it requires the modification of the DCT transformation, thereby limiting reuse of existing DCT hardware solutions. In our work, we focus on the second type of shape encoding, which is based on Context Arithmetic Encoding (CAE) of shape information. The texture part of the object is encoded with the conventional Motion-Compensated DCT coding. This type of arbitrary-shaped VO coding is relying on the traditional block-based processing of coded images.

As is indicated in Figure 2.8, the macroblock bitstream is encoded from three individual parts in this respective order: shape information, motion vectors, texture information. The order of encoding has strong influence on the decod-

ing process because the coded information are inserted to the final bitstream without any markers and therefore without decoding the preceding part, the next one cannot start. The processing of macroblock starts with decoding the BAB type (see Table 2.1). Based on the BAB type, the following functions are performed.

- *Shape reconstruction* - is based on the decoding of BAB-type, the motion vectors of the referenced BAB, the sampling ratio, the scan order and the processing of intra- or inter-Context Arithmetic Encoded (CAE) shape is executed,
- *Motion processing* - the motion vectors are decoded and used for fetching the reconstructed macroblock from the previously reconstructed VOPs,
- *Texture decoding* - is composed of Variable Length Decoding (VLD), the inverse scan, the inverse DC & AC prediction, the inverse quantization (IQ) and the inverse DCT transformation.

The reconstructed macroblock contains four signal channels (see Figure 2.5), the BAB is available after the CAE operation, the Y, Cr, Cb components are fully reconstructed after performing motion compensation on the texture decoding output and the referenced macroblock. The individual decoding operations are discussed in the remainder of this section.

A. BAB-type decoding

A video object plane contains three different MB types for coding of the arbitrary-shaped video objects. The standard distinguishes three types of macroblocks: boundary, opaque, and transparent. The transparent MB does not contain any visible pixels. The opaque macroblock has all pixel visible. Therefore, the BAB is not encoded but is clear from the macroblock type, i.e. all BAB pixels have value 0 for transparent, or 1 for opaque macroblock. The boundary macroblock is defined as a macroblock that contains both transparent pixels and opaque pixels and therefore it contains both the object shape and texture information.

The predictive coding of the BAB introduces another four BAB types: two with changes in shape information and two without changes of the shape. These pairs are further split in cases with shape motion vectors equal to zero or non-zero. Based on the required Context Arithmetic Encoding (CAE) method and motion compensation of the shape, the coded BAB can be classified into seven different types as listed in Table 2.1.

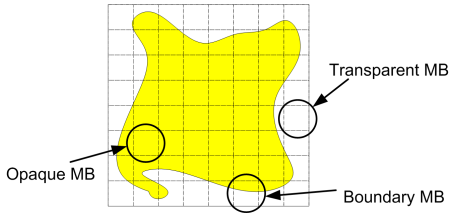


Figure 2.9: *Different macroblocks within VOP.*

BAB type	Semantic
0	BAB motion vector is 0, BAB does not require update of shape.
1	BAB motion vector is non 0, BAB shape does not require update of shape.
2	BAB is transparent.
3	BAB is opaque.
4	BAB requires intraCAE decoding of shape.
5	BAB motion vector is 0; BAB requires interCAE decoding of shape.
6	BAB motion vector is non 0; BAB requires interCAE decoding of shape.

Table 2.1: *BAB types and the required operations for the shape decoding.*

For each macroblock, the BAB type is context-based coded. The context for I-VOP is calculated from the BAB types of surrounded BABs. Let be $b(y,x)$ the BAB type of the BAB where y is the BAB row index and x is the BAB column index. If the index is outside the BAB, the BAB type is assumed to be transparent. A context C is based on previously decoded BAB types and it determines the coding output of the VLC. The computed context number is used as an input index to extract the code bits in a VLC table.

$$C = 27 \cdot (b(y-1,x-1)-2) + 9 \cdot (b(y-1,x)-2) + 3 \cdot (b(y-1,x+1)-2) + \\ + (b(y,x-1)-2) \tag{2.1}$$

The above definition of context C refers to intra-coded VOPs. For B-VOPs, P-VOPs, S(GMC)-VOPs, the context number for decoding the BAB type is based on the collocated previously decoded BAB types in the reference VOP, resulting in seven different context values.

B. Shape motion vector(s) decoding

If the VOP is predictively coded (P- or B-VOP type), the shape can reuse the previously reconstructed shape of the VOP that is used as the prediction. Based on the content, the BAB can be directly reused, is just repositioned, or the content is different and intraCAE or interCAE has to be performed. The

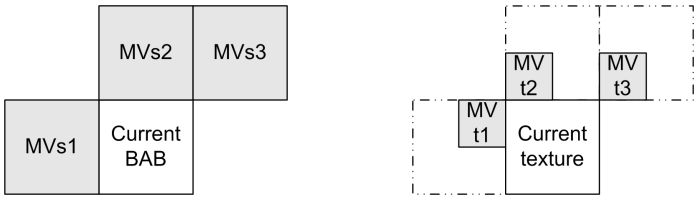
correct position of the referenced BAB is indicated with vertical and horizontal Motion Vectors (MVs).

If the BAB type indicates that motion vectors are not zero (BAB type has value 1 or 6), the motion vectors are reconstructed from the encoded motion vector differences. The Shape Motion Vector Differences (SMVDs) are decoded with a VLC of running 1s and ending with bit 0, followed by a sign bit(1=positive, 0=negative). If the first (horizontal) vector is non-zero, the second (vertical) motion vector is encoded with the same VLC as the horizontal one, otherwise one bit is saved by removing the "0" codeword from the table and shifting all codewords one position up with respect to their index. The reconstructed integer-valued shape motion vector is found by adding the Shape Motion Vector Prediction (SMVP) to the SMVDs as defined above. Figure 2.10 depicts the motion vectors of the surrounded BABs and texture blocks used for the prediction. The candidates are checked in this order: MVs1, MVs2, MVs3, MVt1, MVt2, MVt3. The shape SMVP is determined by taking the first candidate that is defined. From the above, it can be noticed that shape and texture motion are both needed for the decoding of the next macroblock.

C. Sampling ratio and scan order decoding

The individual BABs can be encoded at smaller block sizes than the original 16×16 to save bitrate for shape coding. The standard allows two downsampling factors, i.e. a factor 2 or 4, resulting in a BAB size of 8×8 and 4×4 samples, respectively. The appropriate CAE decoding is performed on the downsampled BAB size and after CAE, the BAB upsampling is invoked in order to obtain the 16×16 sample resolution of the BAB. The sampling ratio is coded with a VLC of 1-2 bits, yielding values 1,2,4.

After the sampling ratio, one bit is inserted in the bitstream, which indicates the horizontal or vertical scanning order of the BAB image for coding. If



(a) Candidates from BABs.

(b) Candidates from texture blocks.

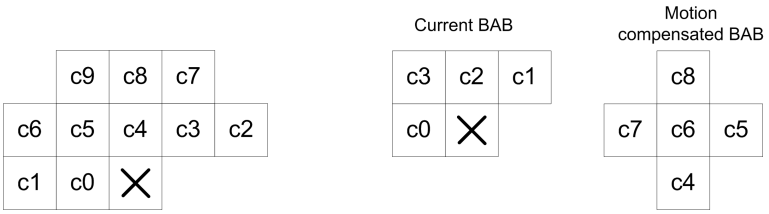
Figure 2.10: *Shape motion vector prediction candidates (characters "s" and "t" refer to shape and texture, respectively).*

the scan order bit has value 0, the original BAB block was transposed before encoding. After the CAE decoding, if required, the BAB block should be inversely transposed.

D. Decoding of Context Arithmetic Encoded shape information

The processing of Context Arithmetic Encoded (CAE) shape information is based on arithmetic decoding. The pixels of the shape representation are decoded in raster order. After the arithmetic decoder is initialized, the following steps are applied to each sample.

Step 1: A context number is computed based on the context samples, as illustrated in Figure 2.11. The context for each pixel is calculated as $C = \sum_k c_k \cdot 2^k$. The two possible types of the context are shown in Figure 2.11.
Step 2: Using the calculated context number, the probability value at the position with index C is used by the arithmetic decoder.
Step 3: The bit sequence of the corresponding binary arithmetic code is decoded, to retrieve the decoded shape sample value.



(a) Context for Intra CAE.

(b) Context for Inter CAE.

Figure 2.11: *Reconstructed BAB samples used for the context calculation.*

Inter-coded VOP can contain both intraCAE- and interCAE-coded BABs, depending on which type of CAE is more efficient with respect to the size of the coded stream. When all samples in a 16×16 Binary Alpha Block (BAB) have been decoded, the arithmetic decoding process is stopped[5].

[5] In special cases, the BAB image can turn from a binary mask into a grayscale block that serves the blending of the video object within another signal. For grayscale shape coding, the shape is composed of two information channels. First, a support region is CAE coded for binary shape coding. Second, the 8-bit alpha values representing the transparency are coded in the same way as luminance texture signals. The data of the grayscale channel are inserted after the data corresponding to texture channels of macroblock.

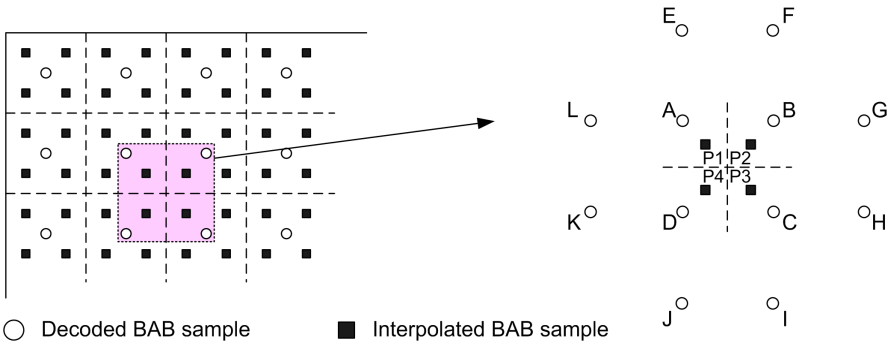


Figure 2.12: *Position of decoded and upsampled BAB values.*

E. Upsampling of the BAB

If the decoded sampling ratio indicates that the BAB was downsampled prior to the encoding, the upsampling processes has to be performed after the CAE decoding. If the BAB was downsampled to 4×4 size, the upsampling is performed two times. Figure 2.12 illustrates the positions of individual pixel samples for the upsampling. First, the context number is calculated based on the surrounding pixels as described in Equation (2.2)

$$\begin{aligned} Cf1 &= F \cdot 2^0 + E \cdot 2^1 + L \cdot 2^2 + K \cdot 2^3 + J \cdot 2^4 + I \cdot 2^5 + H \cdot 2^6 + G \cdot 2^7, \\ Cf2 &= H \cdot 2^0 + G \cdot 2^1 + F \cdot 2^2 + E \cdot 2^3 + L \cdot 2^4 + K \cdot 2^5 + J \cdot 2^6 + I \cdot 2^7, \\ Cf3 &= J \cdot 2^0 + I \cdot 2^1 + H \cdot 2^2 + G \cdot 2^3 + F \cdot 2^4 + E \cdot 2^5 + L \cdot 2^6 + K \cdot 2^7, \\ Cf4 &= L \cdot 2^0 + K \cdot 2^1 + J \cdot 2^2 + I \cdot 2^3 + H \cdot 2^4 + G \cdot 2^5 + F \cdot 2^6 + E \cdot 2^7. \end{aligned} \tag{2.2}$$

$$\begin{aligned} P1: \ & if\ (4 \cdot A + 2 \cdot (B{+}C{+}D) + (E{+}F{+}G{+}H{+}I{+}J{+}K{+}L) > Th[Cf1]) \\ & \text{then 1 else 0}, \\ P2: \ & if\ (4 \cdot B + 2 \cdot (A{+}C{+}D) + (E{+}F{+}G{+}H{+}I{+}J{+}K{+}L) > Th[Cf2]) \\ & \text{then 1 else 0}, \\ P3: \ & if\ (4 \cdot C + 2 \cdot (B{+}A{+}D) + (E{+}F{+}G{+}H{+}I{+}J{+}K{+}L) > Th[Cf3]) \\ & \text{then 1 else 0}, \\ P4: \ & if\ (4 \cdot D + 2 \cdot (B{+}C{+}A) + (E{+}F{+}G{+}H{+}I{+}J{+}K{+}L) > Th[Cf4]) \\ & \text{then 1 else 0}. \end{aligned} \tag{2.3}$$

The context numbers $Cf1, Cf2, Cf3, Cf4$ are used to access the probability table that determines the threshold value based on which the opacity/transparency

is set, depending on a number of conditions. These conditions are based on reconstructing a number of samples, denoted in Figure 2.12 by $P1, P2, P3, P4$ as described in Equation (2.3).

It can be concluded that the reduction in CAE decoding of downsampled BABs is leading to extra processing for interpolation.

F. Decoding of luminance CBP

The Coded Block Pattern (CBP) represents the pattern of non-transparent 8×8 luminance blocks with at least one non-intra DC coefficient. The pattern is important to distinguish between the luminance and chrominance blocks. Since they are encoded without any extra marker information and a boundary macroblock can have a variable number of luminance blocks, this decoding pattern is required. Further, in the case of less than four luminance blocks, the CBP positions luminance blocks within the macroblock.

G. Texture motion-vector decoding

In interframe coding, the reconstruction of the texture blocks relies on the reference block that is similar to the currently reconstructed block. This block is used as a prediction that is subtracted from the pixel values of the current block and only the differences are encoded. The position of the reference block is determined by a Motion Vector (MV). The bounding box of the VOP is positioned by means of the frame coordinates in the VOP header parameter, called $VOP\_spatial\_reference$. The motion vector of the current block (x_{curr}, y_{curr}) is calculated based on the absolute position of the current block within the frame and the absolute position of the reference block (x_{reff}, y_{reff}). An example of the texture motion compensation is given in Figure 2.13.

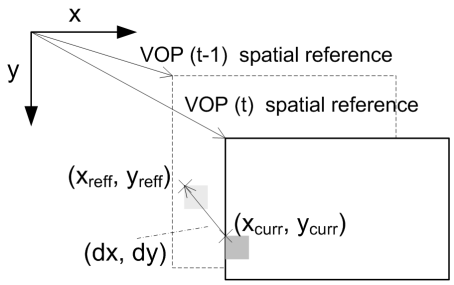
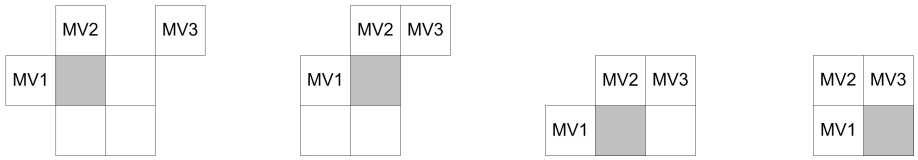


Figure 2.13: *VOP motion compensation.*

$$\begin{aligned} P_x &= Median(MV1_x, MV2_x, MV3_x) \\ P_y &= Median(MV1_y, MV2_y, MV3_y) \end{aligned} \tag{2.4}$$

The motion vector is decoded differentially by using a prediction that is formed by a median filtering of motion vector predictors $(MV1, MV2, MV3)$. The spatial position of candidates is depicted in Figure 2.14. In the case a candidate is not valid, its value is set to 0. The value of the texture MV is obtained as sum of the VLC-coded motion vector differences (MVD_x, MVD_y), and the prediction (P_x, P_y). The prediction calculation is given in Equation (2.4).



(a) Top left. (b) Top right. (c) Bottom left. (d) Bottom right.

Figure 2.14: *Candidate motion-vector predictors for individual* 8×8 *blocks of macroblock.*

H. Variable length decoding

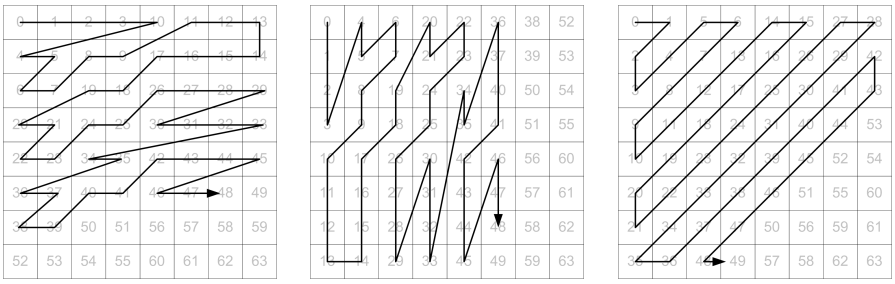
The decoding of texture information of a VOP starts with Variable Length Decoding (VLD) of coded data. Two different approaches of VLD are used: the VLD for a differential DC coefficient ($n = 0$ in the scan order) of intra-coded macroblocks and the VLD for differential AC and DC coefficients of inter-coded macroblocks.

The decoding of a *differential DC coefficient* of intra-coded macroblocks starts with decoding the dct_dc_size that is a VLC code that categorizes the differential DC coefficient according to its "size". For each category, the $dct\_dc\_differential$ is added to identify which difference in that category actually occurred. The final value is the sum of the differential DC values and the predicted values that are discussed later in this subsection.

Other differential coefficients are decoded with variable length codes to produce EVENTs. The standard defines an EVENT as a combination of an indication of a last non-zero coefficient in the current block (LAST), the number of running zeros preceding the coded coefficient (RUN) and the non-zero value of the coded coefficient (LEVEL). The MPEG-4 standard contains the VLC table for

a subset of possible combinations (Table B-16 of the standard). However, many of the possible EVENTs have no variable length code to represent them. In order to code them, the MPEG-4 standard defines an Escape Coding method to encode these statistically rare combinations. The complexity of VLD relies in the iterative procedure of updating probability tables and using those tables for decoding the bitstream.

I. Inverse coefficient scanning



(a) Alternative-horizontal scanning. (b) Alternative-vertical scanning. (c) Zigzag scanning.

Figure 2.15: *MPEG-4 scanning patterns for converting to an* 8×8 *block.*

Prior to the inverse quantization, a one-dimensional array of differential coefficients decoded by VLD are converted to two-dimensional 8×8 array. The MPEG-4 standard specifies three types of scanning orders for differential coefficients decoding as shown in Figure 2.15. For intra-coded blocks where the $acpred\_flag$ parameter is 0, the zigzag scanning order is selected for all blocks in the macroblock. Otherwise, the DC prediction direction is used to select a scanning pattern on block basis. If the DC prediction refers to the horizontally adjacent block, the alternate-vertical scanning is used, otherwise the alternate-horizontal scanning is applied.

J. Inverse DC/AC prediction

The adaptive DC prediction determines the prediction value based on previously decoded blocks. Let $F[0,0]$ denote the inverse quantized DC value of a block. The prediction for block X in Figure 2.16 is taken from the block C or block A as follows

$$\begin{aligned} &if && (\mid F_A[0,0] - F_B[0,0] \mid < \mid F_B[0,0] - F_C[0,0] \mid) \\ &then && \text{predict from } C, \\ &else && \\ & && \text{predict from } A. \end{aligned} \tag{2.5}$$

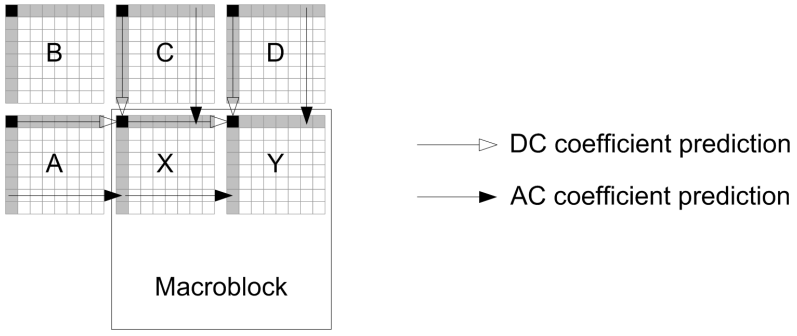


Figure 2.16: *Previously decoded blocks are used for DC and AC prediction.*

The quantized value of the DCT coefficient QF_X of the block X is reconstructed from the inverse scanned differential coefficient PQF_X. The quantization scaling factors of the surrounding blocks can be different so that the prediction of the actual block should be compensated for those different scaling factors (denoted as dc_scaler). For this reason, we incorporate a scaling operation in the reconstruction, hence

$$\begin{aligned} &if && \text{predict from } C \\ &then && QF_X[0,0] = PQF_X[0,0] + F_C[0,0]/dc\_scaler \qquad (2.6) \\ &else && \\ & && QF_X[0,0] = PQF_X[0,0] + F_A[0,0]/dc\_scaler. \qquad (2.7) \end{aligned}$$

When AC prediction is enabled by the parameter ac_pred_flag, the coefficients of the first row or the first column of adjacent blocks are used as prediction. The difference in quantization has to be taken into account by scaling the prediction coefficient with the ratio between the current quantization step and the one used for the quantization of the predictor block. The resulting AC coefficients are obtained as follows

$$\begin{aligned} &if && \text{predict from } C \\ &then && QF_X[0,u] = PQF_X[0,u] + (QF_C[0,u] \times QP_C)/QP_X, \quad u = 1,..,7, \\ & && \qquad (2.8) \\ &else && \\ & && QF_X[v,0] = PQF_X[v,0] + (QF_A[v,0] \times QP_A)/QP_X, \quad v = 1,..,7. \\ & && \qquad (2.9) \end{aligned}$$

The quantized coefficients resulting from DC and AC prediction have to be in the amplitude range [-2048, +2047]. Therefore the, following saturation is

used prior to the inverse quantization.

$$QF[v,u] = \begin{cases} +2047, & \text{if } QF[v,u] > 2047, \\ QF[v,u], & \text{if } -2048 \leq QF[v,u] \leq 2047, \\ -2048, & \text{if } QF[v,u] < -2048. \end{cases} \tag{2.10}$$

J. Inverse quantization

The quantized coefficients $QF[v,u]$ have to be inversely quantized to reconstruct DCT coefficients. The inverse quantization of DC coefficient differs from the inverse quantization of other coefficients. The reconstructed DC values are obtained as follows:

$$F[0,0] = dc\_scaler \times QF[0,0]. \tag{2.11}$$

Depending on the *quant_type* parameter, one of the specified two inverse quantization methods is used for obtaining de-quantized AC coefficients. If the *quant_type* parameter is non-zero the first inverse quantization depends on two weighting matrices, where the first is defined for intra- and the second is defined for inter-coded macroblocks[6]. If the weighting matrix is denoted as W, the values $F[v,u]$ are reconstructed by

$$F[v,u] = \begin{cases} 0, & \text{if } QF[v,u] = 0, \\ ((2QF[v,u] + k) \times W[v,u] \times q\_s)/16, & \text{if } QF[v,u] \neq 0. \end{cases} \tag{2.12}$$

where q_s is the $quantiser_scale$ parameter and

$$k = \begin{cases} 0, & \text{for intra blocks}, \\ Sign(QF[v,u]), & \text{for inter blocks}. \end{cases}$$

The second inverse quantization is used when *quant_type* parameter is equal to zero. The AC coefficients depend on the quantization parameter *quantiser_scale* (denoted as q_s in Equation (2.13)) that may take integer values from 1 to $2^{quant\_precision-1}$.

$$F[v,u] = \begin{cases} 0, & \text{if } QF[v,u] = 0, \\ (2|QF[v,u]| + 1) \times q\_s, & \text{if } QF[v,u] \neq 0, q\_s \text{ is odd}, \\ (2|QF[v,u]| + 1) \times (q\_s - 1), & \text{if } QF[v,u] \neq 0, q\_s \text{ is even}. \end{cases} \tag{2.13}$$

All coefficients after inverse quantization have to be saturated to lie in the range $[-2^{bits\_per\_pixel+3}, 2^{bits\_per\_pixel+3} - 1]$.

[6] The weighting matrices can be overwritten by a user-defined values

K. IDCT

The inverse DCT transformation is performed on inversely quantized coefficients as discussed above. The IDCT for obtaining reconstructed samples is based on the following equation

$$f[x,y] = \frac{2}{N}\sum_{u=0}^{N-1}\sum_{v=0}^{N-1} C(u)C(v)F[u,v]cos\frac{(2x+1)u\pi}{2N}cos\frac{(2y+1)v\pi}{2N}, \quad (2.14)$$

where $N = 8$ and

$$C(u), C(v) = \begin{cases} \frac{1}{\sqrt{2}}, & \text{for } u, v = 0, \\ 1, & \text{otherwise.} \end{cases}$$

The accuracy of the $N \times N$ IDCT should conform to the IEEE standard specification for the implementations of the 8×8 inverse discrete cosine transformation [52]. For the implementation of the IDCT algorithm, we have employed a fast algorithm similar to the one discussed in [74]. Our implementation also satisfies the above-mentioned accuracy requirement.

L. Repetitive padding

The reference that is used for texture prediction can be any type of texture block, i.e. opaque, boundary, or transparent. For the prediction, the transparent pixels of boundary blocks are replaced by data copies of object-boundary pixels, in order to minimize the amount of DCT coefficients after the transformation. The so-called *repetitive padding steps* should be performed to obtain the same reference data as in the encoder for a proper motion compensation.

The boundary blocks are first padded using *Horizontal Repetitive Padding*, i.e. each sample at the boundary of the VO is replicated horizontally to the left and/or right to fill the row of transparent pixels outside the VO of the boundary block. The remaining unassigned transparent horizontal samples are padded using *Vertical Repetitive Padding*, which works in a similar way, but column-based as illustrated by Figure 2.17.

M. Extended padding

Similar to repetitive padding, the *Extended Padding* task should fill fully transparent blocks of a VO plane with padded values. After performing the repetitive padding at the macroblock level, the VOP contains only fully defined MBs (padded boundary and opaque) or undefined (transparent) MBs. The motion compensation of subsequent VOPs can require blocks from the reference VOPs that partially or fully refer to a block that has undefined texture information.

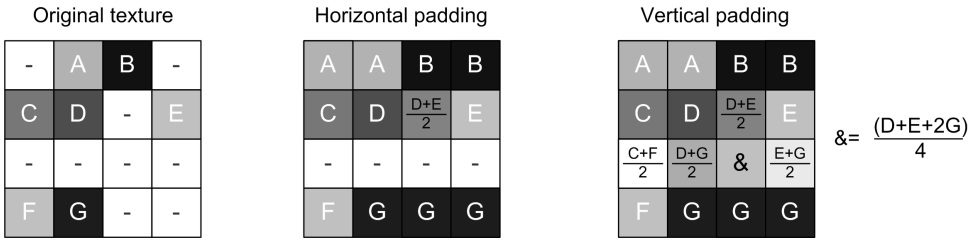


Figure 2.17: *Principles of horizontal and vertical repetitive-padding algorithm. The characters represent different pixel values and the hyphens indicate transparent pixels.*

Therefore, the extended and boundary padding should define the complete texture information for the whole image and should be accomplished prior to starting the next motion-compensated VOP decoding.

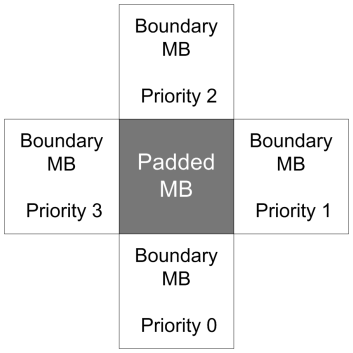


Figure 2.18: *Priorities for border copying of extended padding, where priority 0 is defined as the highest priority.*

The extended padding algorithm works as follows. Exterior macroblocks adjacent to boundary macroblocks are filled completely by replicating the samples at the border of the boundary macroblocks into the whole block. If the padded macroblock has several neighboring macroblocks, then the macroblock with the highest priority for padding is selected. The priority is assigned according to Figure 2.18, and the exterior macroblock is padded by replicating the horizontal or vertical border. The exterior macroblocks not having any neighboring boundary-macroblock are filled with an integer value of $2^{bits/pixel-1}$ (for 8-bit luminance coding, this means filing with 128).

N. Deblocking

The standard defines two post-processing filters, deblocking and deringing filters. The deblocking filter removes the coding artifacts at the block edges. The horizontal and vertical operations are performed along the 8×8 block edges of luminance and chrominance data.

The first, default mode of filtering is applied in normal cases. However, if there is a small difference in dc values on the block boundaries, the so-called DC offset mode of deblocking is used. This mode is switched on when the following criterion in Equations (2.15) and (2.16) holds. Figure 2.19 denotes the pixel indexing used for the filtering operations. The following condition is evaluated in order to select the correct filtering mode.

$$\begin{aligned} eq\_cnt = \quad & \phi(v_0 - v_1) + \phi(v_1 - v_2) + \phi(v_2 - v_3) + \phi(v_3 - v_4) + \phi(v_4 - v_5) + \\ & \phi(v_5 - v_6) + \phi(v_6 - v_7) + \phi(v_7 - v_8) + \phi(v_8 - v_9), \end{aligned} \tag{2.15}$$

where $THR1 = 2$ and

$$\begin{aligned} \phi(y) &= 1, \quad \text{if } |y| \leq THR1, \\ \phi(y) &= 0, \quad \text{otherwise.} \end{aligned} \tag{2.16}$$

The filtering in default mode is applied when eq_cnt calculated from Equation (2.15) is smaller than the threshold value THR2=6. In the default mode, the boundary pixel values v_4 and v_5 are modified by removing or adding the delta d that is calculated as follows:

$$\begin{aligned} v_{4,filtered} &= v_4 - d, \\ v_{5,filtered} &= v_5 + d, \\ d = MINMAX(5 \cdot (a'_{3,0} - a_{3,0})/8, p, q) \cdot & \quad \delta(|a_{3,0}| < QP), \end{aligned} \tag{2.17}$$

where $p = 0$ and $q = (v_4 - v_5)/2$ and furthermore,

$a'_{3,0} =$	$SIGN(a_{3,0}) \cdot MIN(\lvert a_{3,0}\rvert, \lvert a_{3,1}\rvert, \lvert a_{3,2}\rvert)$,
$\delta(cond)$	returns 1 if *cond* is true and 0 otherwise,
$MINMAX(x, p, q)$	limits value x within range p to q,
QP	denotes the quantization parameter of the MB where v_5 belongs.

The frequency components $a_{3,0}, a_{3,1}, a_{3,2}$ are evaluated as the inner product of the approximated DCT vector [2 -5 +5 2] with the pixel vector $\mathbf{v}_i$ as below

$$\begin{aligned} a_{3,0} &= ([2 \ \ -5 \ 5 \ 2] * [v_3 \ v_4 \ v_5 \ v_6]^T)/8, \\ a_{3,1} &= ([2 \ \ -5 \ 5 \ 2] * [v_1 \ v_2 \ v_3 \ v_4]^T)/8, \\ a_{3,2} &= ([2 \ \ -5 \ 5 \ 2] * [v_5 \ v_6 \ v_7 \ v_8]^T)/8. \end{aligned} \tag{2.18}$$

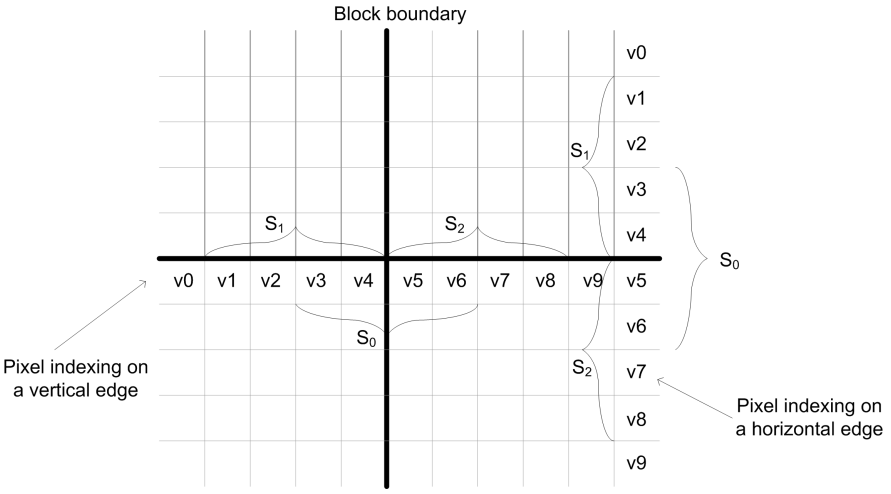


Figure 2.19: *Indexing of pixels for deblocking filtering.*

The second mode of the deblocking filter, called DC offset mode, is applied when eq_cnt calculated from Equation (2.15) is equal or larger than the threshold THR2. Exceeding the threshold value indicates that the default mode will not be good enough to reduce the blocking artifacts at block grid positions. The second mode is described in pseudo code as follows

$$
\begin{aligned}
&if(|max - min| \quad < 2 \cdot QP) \\
&\qquad \{ \\
&\qquad\qquad v'_n = \sum_{k=-4}^{4} b_k \cdot p_{n+k}, \text{ for } 1 \le n \le 8 \\
&\qquad\qquad p_m = \begin{cases} (|v_1 - v_0| < QP) \ ? \ v_0 : v_1, & \text{if } m < 1, \\ v_m, & \text{if } 1 \le m \le 8, \\ (|v_8 - v_9| < QP) \ ? \ v_9 : v_8, & \text{if } m > 8, \end{cases} \\
&\qquad\qquad \{b_k : -4 \le k \le 4\} = \{1, 1, 2, 2, 4, 2, 2, 1, 1\}/16 \\
&\qquad \} \\
&else \text{ no change},
\end{aligned}
$$

where

$$
\begin{aligned}
max &= MAX(v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8), \\
min &= MIN(v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8).
\end{aligned}
\tag{2.19}
$$

O. Deringing filter

The deringing is a second, optional post-processing step after deblocking that can be switched on independently. This function removes ringing effects in the video signal within the 8×8 blocks. The deringing filter architecture is based on three steps: threshold determination, index acquisition and adaptive smoothing. The filtering is applied on all luminance and chrominance blocks.

The threshold determination starts with calculating the minimal and the maximal gray value within an 8×8 block, denoted as $min[k]$ and $max[k]$, respectively. The threshold and range for the block k are obtained as follows

$$thr[k] = (max[k] + min[k] + 1)/2, \tag{2.20}$$

$$range[k] = max[k] - min[k]. \tag{2.21}$$

An additional process is applied for the four luminance blocks of a macroblock. If max_range is the maximum value of the dynamic range among four luminance blocks, the following rearrangement of the thresholds is made, specified by

$$\begin{aligned} for(\quad & k = 1; k < 5; k++)\{ \\ & if((range[k] < 32) \;\&\&\; max\_range \geq 64) \\ & \quad thr[k] = thr[kmax]; \\ & if(range[k] < 16) \\ & \quad thr[k] = 0; \\ \}. & \end{aligned}$$

Once the threshold value is determined, the second deringing step involves an index acquisition that can be obtained as follows. Let $rec[h, v]$ and $bin[h, v]$ be the gray value at coordinates $[h, v]$ where $h, v = 0, 1, 2, ..., 7$, and the corresponding binary index (opaque or transparent), respectively. Then $bin[h, v]$ is obtained by

$$bin(h, v) = \begin{cases} 1 & \text{if } rec[h, v] > thr, \\ 0 & \text{otherwise.} \end{cases} \tag{2.22}$$

1	2	1
2	4	2
1	2	1

Figure 2.20: *Filter mask for adaptive smoothing.*

The array of binary indices obtained in the previous step are used for applying the adaptive filtering. The filter is applied only if the binary indices in a 3x3 window are all the same, i.e. they consist of only "0" indices or only "1" indices. The filter coefficients used for both intra and inter blocks are denoted by $coef[i,j]$, where $i,j=-1,0,1$, as depicted in Figure 2.20.

The coefficient at the center pixel, i.e., $coef[0,0]$, corresponds to the pixel to be filtered. The filter output is obtained by computing

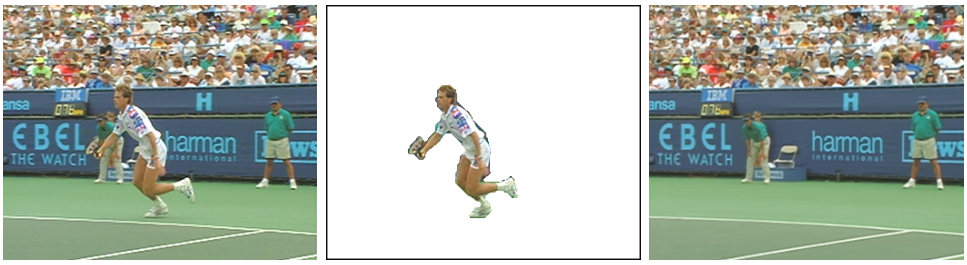
$$filt[h,v] = \left(8 + \sum_{i=-1}^{1}\sum_{j=-1}^{1} coef[i,j] \cdot rec[h+i, v+j]\right)/16. \qquad (2.23)$$

2.5 Background sprite coding

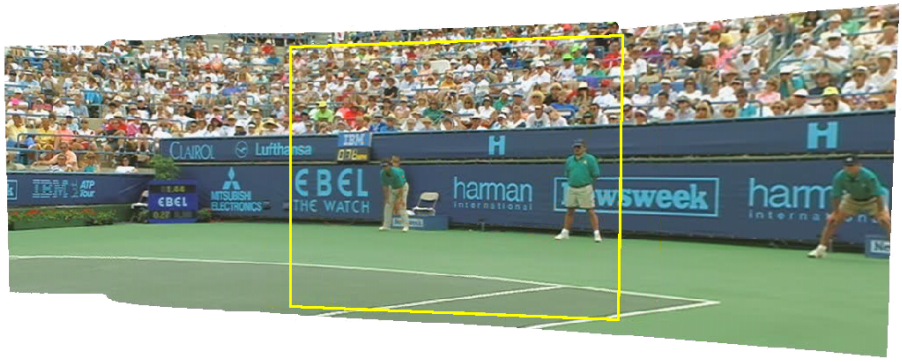
In contrast with the previous steps that are always used in MPEG-4 AS VO decoding, the background sprite coding in this section is only used in specific cases. In natural video sequences, a background is always defined, but this does not apply to synthetic sequences or cases where there is only a still picture as a scene background. The details of the sprite coding are not presented in this chapter, but only in the involved chapter where it is actually used. All other chapters do not employ this feature, so that we only present the concept.
The background sprite image is used for the reconstruction of the scene background for a set of consecutive frames. The sprite is constructed by merging several views into one large image that is further encoded and transmitted. The MPEG-4 standard requires the transmission of warping parameters so that the decoder knows which part to extract from the sprite and how it should be warped. At the time of the background reconstruction at the decoder, the sprite warping generates the current camera view that is used in the renderer for the scene recomposition (see Figure 2.21(d)). The sprite transformation is modeled as a mapping between the decoded sprite plane and a current view plane. This transformation involves a so-called affine transformation containing rotation, translation and perspective scaling. It is described by the following specification:

$$x' = \frac{m_{00}\cdot x + m_{01}\cdot y + m_{02}}{m_{20}\cdot x + m_{21}\cdot y + 1}, \quad y' = \frac{m_{10}\cdot x + m_{11}\cdot y + m_{12}}{m_{20}\cdot x + m_{21}\cdot y + 1}, \qquad (2.24)$$

where x', y' are the transformed display coordinates and x, y are the original coordinates for the video texture in the sprite buffer. The global motion parameters m_{ij} in the above equations are calculated at the encoder.



(a) Original image. (b) Segmented object. (c) Background without object.



(d) Composed sprite image.

Figure 2.21: *Visualization of the reconstruction of an MPEG-4 background from a sprite image. The wire frame indicated inside the background at the bottom figure specifies the extraction area used for reconstruction in the actual frame. Pictures are taken from Chapter 1 in [37].*

This completes the detailed description of the implemented MPEG-4 AS VO decoder and the sprite coding concept. The following sections will be addressing the platform on which the MPEG-4 decoder will be executed.

2.6 Network-on-Chip (NoC)

As already discussed in the previous chapter, the focus of our research is on using a multiprocessor system for executing the MPEG-4 AS VO decoder. The parallelism of the platform enables the parallel execution of a multitude of video object decoders, where the objects can have variable characteristics (size, motion, shape, etc.). Such a mapping of a multitude of object decoders onto a multiprocessor platform poses an interesting challenge.

In the remaining part of this chapter, we will discuss the experimental setup of the multiprocessor platform that we studied and used for the above mapping. First, we will outline the architecture cornerstones of a networked multiprocessor system. Afterwards, in a following section, we discuss the topology of the network. Subsequently, a section on a tile-based processing presents more details on the individual processors. After having presented the individual processor layout, we provide two architecture instantiations of a networked multiprocessor, i.e. Æthereal NoC and CELL processor, which were used in the experiments in the succeeding chapters. The last section of this chapter summarizes the design flow.

2.6.1 NoC computation units

Computation units are one of the dominating elements of an NoC architecture. The research on mapping of MPEG-4 processing onto platforms concentrates on efficient implementations [78, 3, 118], or on the encoder for the MPEG-4 Simple Profile (SP) [31, 30]. The results of this research shows that a heterogeneous architecture is attractive for high efficiency in computing. However, for a networked multiprocessor, first the backbone employing a network and a set of programmable processors has to be developed. This is also our starting point in this thesis.

Regardless of implementation details, a general approach for designing a dedicated multimedia system is based on using a DSP or RISC core processor (or both) for control and flexible processing, or special accelerators for pixel-based or streaming-oriented video functions, e.g. DCT, VLD, CAE, ME, MC, etc. The following elements are commonly used processing units for a multiprocessor NoC.

- *General-Purpose (GP) processing cores* providing high computational performance and level of programmability. The advantage of GP cores is that any kind of application can be mapped, but processor utilization and the communication usage is highly unpredictable at the design time.
- *Dedicated processors* support a specific class of applications only. This is the most efficient solution, but also most expensive and it requires a long design time. Dedicated processors are typically used for algorithms where computing is very demanding.
- *Field-Programmable Gate Arrays* (FPGAs) sub-systems contain programmable logic components and programmable interconnects. Components differ in size and functionality from basic operations as AND, OR up to complex video functions. The major advantage of FPGA technology is

the re-programmability and therefore it fills the gap between dedicated and general-purpose processors.

Concluding, the above-listed processing can be combined in any kind of fashion into a heterogeneous NoC platform. The optimal mixture of those elements depends on the application and its requirements. Despite the fact that we would prefer a heterogeneous solution, there was not sufficient manpower to develop dedicated processors for specific tasks. Therefore, we focus on a homogeneous network-on-chip in this thesis. In a succeeding project, FPGA subsystems containing dedicated accelerating processors will be added (this was started while writing this thesis; a first application study on finding of objects was reported in [115]).

2.6.2 NoC topologies

The NoC chips contain the processing and storage units, and the *switches* and *physical links* to facilitate the connections between them. The switches act as routers and repeaters in a network. Network architectures are generally categorized with respect to their topology as follows.

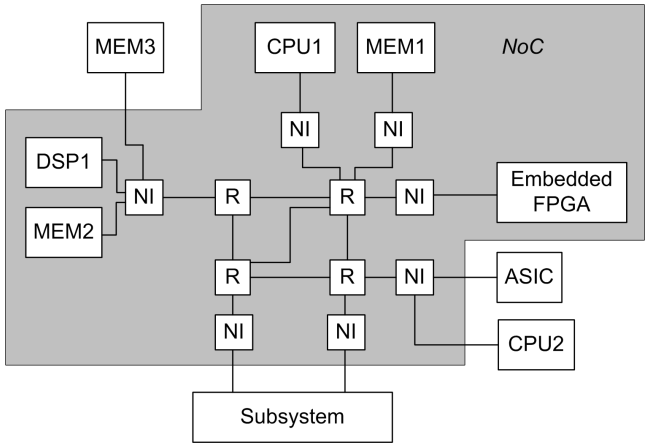


Figure 2.22: *An example of an NoC with indirect connections, because every unit is connected to a Network Interface (NI).*

- *Shared medium networks*: the link between the nodes is shared and only one node can use it at a time (e.g. a bus).
- *Direct networks*: the topology allows direct point-to-point communication (e.g. a switch matrix).

- *Indirect networks*: the nodes have a network port that is connected to a switch acting as a router, providing point-to-point communication (this acts as a mixture of the two previously mentioned networks).
- *Hybrid networks*: the combination of the above-mentioned approaches, mostly providing direct communication as an extension of an indirect or shared medium network.

Our target architecture employs an indirect network, where individual processing units are connected via a so-called network interface to the routers. This choice was motivated by several aspects.

First, the network interface separates the computing cores from the network. This enables that estimation of computational resources can be handled separately from the estimation of communication resources. Further, the integration of accelerators is performed more easily due to the well-defined interfaces.

Second, our first results on resource usage estimations (similar to the estimation techniques used in [55]) of the resource requirements revealed that one single MPEG-4 AS VO, would require up to eight ARM7 [1] cores[7] for decoding a single object at CIF resolution at 25 frames/s. If a decoder should support MPEG-4 Level 1 decoding, a scene may have up to four AS video objects. Then the NoC should have at least 32 ARM7 cores. This number is too large for an efficient design of a complex switch matrix. This relatively simple analysis leads to the conclusion that direct networks cannot be used in our NoC. Therefore, as indicated above, we have adopted indirect networks. Besides this argumentation, indirect networks offer a self-configuring property that can be employed in the Quality-of-Service approach from Chapter 5.

2.7 Tile-based NoC and application modeling

For the NoC platform, we employ a *tile-based* architecture with distributed memory, as depicted in Figure 2.23. Each *processing tile* represents a small self-contained embedded computer, consisting of one embedded CPU core (e.g. RISC), local memory for data (DMEM) and instructions (IMEM), and possibly, application-specific accelerators (not depicted in the figure). Furthermore, each tile contains a so-called Communication Assist (CA), which is a gateway from the tile's local memory subsystem to the standard network services. A *storage tile* provides access to large memories in the background, e.g. external off-chip memory. The storage tile contains a memory controller that performs

[7] This calculation was based on multiplying an averaged amount of clock cycles needed for one macroblock, the number of macroblocks per frame and frame frequency in order to obtain a first impression on the complexity of computations on an embedded processor.

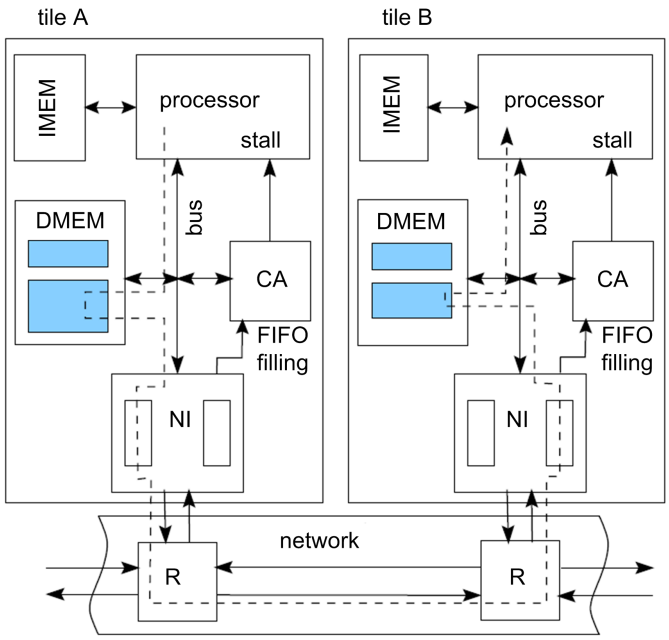


Figure 2.23: *MP-NoC tile-based architecture which is the corner stone for this thesis. A key feature is that every tile has its own local memory without caches to support predictable behavior.*

data transfers (possibly with some pre/post-processing) between the network and the off-chip memory. The NoC transports data packets between the tiles.

In general, the applications mapped onto the NoC are modeled with dataflow graphs, provided that the following constraints on this architecture are satisfied.

- The tasks run concurrently in different tiles and use only the local memory from the same tile (Note that the execution of tiles is based on predictable real-time scheduling, e.g. TDMA). The instructions and data structures used by tasks are mapped to the local memories and caching is not used. This enables a prediction of the task execution times. The accesses to remote memory[8] through the storage tiles may be performed using explicit message passing, involving the communication assist (CA) and the switch network.
- The local memory arbitration between the processor and the communi-

[8] A-priori knowledge about the memory access patterns can be used for an efficient organization of the memory traffic[64].

cation assist should have well-defined access delays to have predictable local behavior.

- As already stated using indirect networks, the NoC provides point-to-point connections with tightly bounded packet propagation delays. These bounded delays can be achieved in the NoC at a reasonable cost [102]. Similar to data edges in dataflow graphs, the connections must be independent from each other and they must carry multiple tokens in FIFO order.

In conclusion, from the above tile-based MP-NoC description, we extract the following NoC key characteristics. First, data is passed from one tile to the other in a robust way, free from access conflicts, due to the independent connections. Second, both flexible homogeneous and more efficient heterogeneous platforms are supported. Third, any kind of communication link can be set up due to the well-defined communication structure.

2.8 Applied NoCs for experiments

2.8.1 Æthereal NoC

Several NoCs [9, 26, 44, 75] have been proposed, using different topologies and routing strategies. In our research we aim at mapping two types of communication services onto the previously defined tile-based architecture. These service types are *guaranteed services* and *best-effort services*. We have adopted the Æthereal interconnect [42] as the NoC system offering both services, as it was shared by the project partners in our research domain (PreMaDona, Hijdra).

In our work on Quality-of-Service, an NoC should offer runtime monitoring features. Up till now, the monitoring was used mainly for debugging purposes. The role of monitoring becomes more valuable when it is coupled to advanced QoS management that can explore the monitoring information for better distribution of resources. The NoC monitoring service, as illustrated in Figure 2.24, consists of configurable monitoring probes (P) attached to a selection of the NoC components, i.e. Routers (R) or Network Interfaces (NI). Processing nodes in our architecture are ARM cores, as depicted in the figure. On top of the probes, an associated programming model is deployed, and a monitoring traffic management strategy has to be considered. In the sequel, we discuss briefly the monitoring in the NoC.

The *monitoring probes* collect the required information from the NoC components in the form of events. The collected information involves link utilization and point connection information. Although in our experiments all

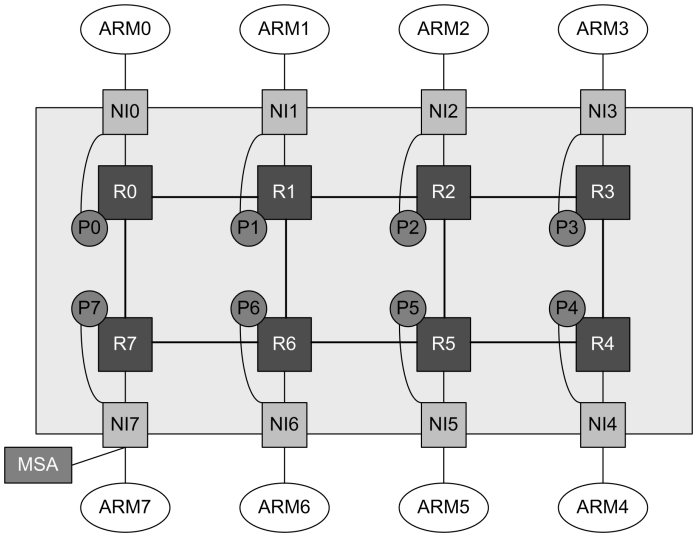


Figure 2.24: *NoC architecture with a monitoring service (MSA) and monitoring Probes (Pn) to support QoS control.*

routers have activated probes, generally monitoring probes are not necessarily attached to all NoC components.

The *traffic management* is implemented with a Monitoring Service Access point (MSA), which is primarily used to collect network traffic information from the probes. The MSA provides monitoring information to the system control part (in our case QoS management) and the MSA can also configure the probes. In our experiments, the information exchange was implemented via Guaranteed Throughput (GT) connections embedded within the existing network topology.

In our implementation, we employ RISC processor tiles, using an instruction-set simulator of the ARM7 RISC processor model [1]. Local memory is assumed to have a uniform address space with a single-cycle access delay.

2.8.2 CELL processor

An alternative already available multiprocessor NoC is comprising heterogenous cores. This NoC implementation will be used in Chapter 4 and is depicted in Figure 2.25. The architecture consists of a 64-bit *Power Processor Element* (PPE) and its L2 cache, multiple *Synergistic Processor Elements* (SPE) [38] that all have their own local memory (LS) [32], a high-bandwidth internal *Element Interconnect Bus* (EIB), two configurable non-coherent I/O interfaces, a Memory Interface Controller (MIC), and a pervasive unit that supports extensive test, monitoring, and debug functions.

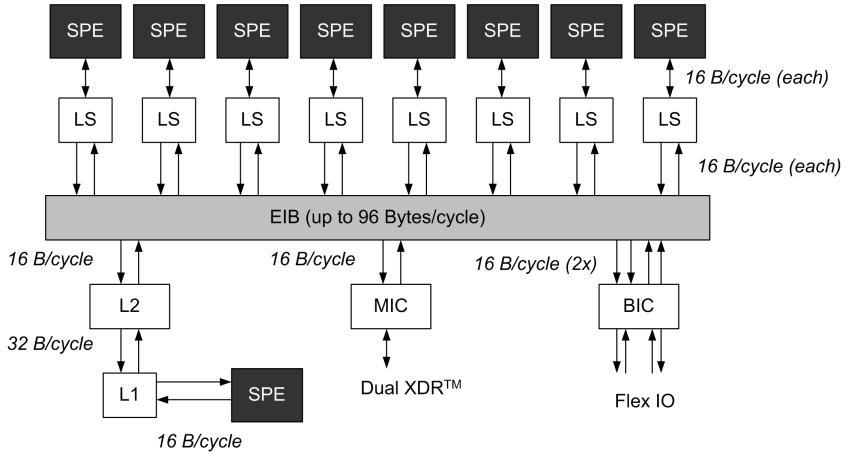


Figure 2.25: *High-level Cell processor diagram, from [95].*

The (EIB) connects to one PPE, the eight SPEs and the memory interface controller. The commands and data communication are separated. The EIB has a structure that is based on four "rings", each representing a chain connecting all data ports. Data moves through a ring in one direction. Two rings push data clockwise and two other rings communicate counterclockwise. Each ring can move 16 Bytes at a time from any position on the ring to any other position. The suitability of this architecture is further evaluated in Chapter 4.

2.9 Design flow

For our experiments, we have adopted the design flow as presented in [107], but we have superimposed two additional steps on top of it. The design flow is schematically shown in Figure 2.26. The two steps at the top were added in our work to explore the video processing algorithm with respect to its mapping onto a multiprocessor platform and to properly partition an algorithm into Synchronous Data Flow (SDF) graphs, which are the starting point for the platform-specific design steps (covered by the existing design flow).

At the time of starting the research, the required SDF graphs were not available so that the author was forced to explore the top steps in depth. An SDF graph model is basically a set of communicating tasks and the dependencies among individual processing functions called *actors*. This model of computation is used in the later stages of the design flow (see Chapter 3 for more details on data-flow graphs). During the design, the partitioned actors were also ported to

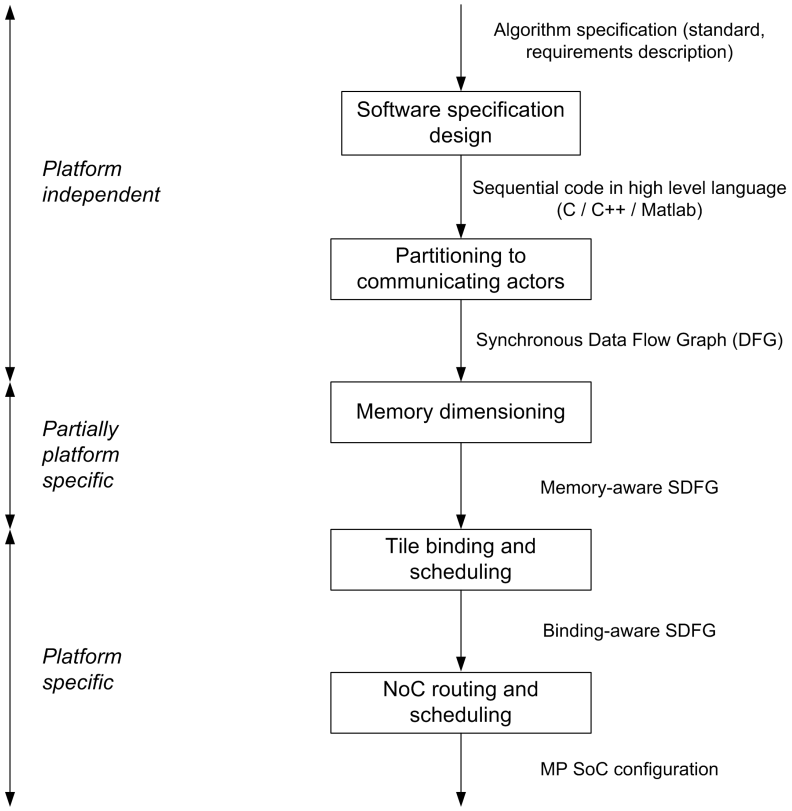


Figure 2.26: *Multiprocessor SoC design flow.*

the target processing core. The compilation and execution of functions on the target processor revealed directly realistic data on the computation resource requirements per actor.

2.10 Mapping assumptions

The mapping of the chosen multimedia application on the multiprocessor NoC is influenced by several assumptions and boundaries to the research work. Based on the application characteristics and the target architecture, we specify the following assumptions and constraints for our work.

- An application is split into a set of communicating tasks executed in pipelined way, which is modeled by a data-flow graph or by a Kahn Process Network.
- The initialization time of an application is excluded from the application models.

- A tile-based architecture having one processor per tile is considered as a target platform.
- Processing tiles have programmable embedded processors as computation units.
- The task scheduling at individual tiles is *a-priori* known and can be modified to meet the task/application real-time requirements.
- An application uses only local memory of a tile without caching.
- The physical allocation of resources is provided by platfrom/OS specific services that are controlled by a QoS system.
- The study is performed on homogeneous processor platforms only, without further exploring accelerators or FPGA components.

2.11 Conclusions

The first part of this chapter introduced object-based video coding in detail based on the MPEG-4 Core Profile and using arbitrary-shaped video objects. The complexity of object-based video processing results from the combination of high-level control-driven operations and streaming-oriented processing at video-data level. The most important observation of this type of computing is that processing of video objects that vary in shape and behavior over time, leads to dynamic usage of platform resources. This property poses new requirements on managing of the resources. In a general application case, we can instantiate an arbitrary number of objects with arbitrary behavior.

In the second part of this chapter, we introduced a network-based multiprocessor system. An essential architectural property is the tile-based Network-on-Chip (NoC) approach, where processors are connected to a local network with a special network interface. The NoC contains routers and traffic probes providing runtime monitoring facilities to implement Quality-of-Service (QoS) management for the whole system and individual tasks. For the experiments, we employed two models of recent NoCs: Æthereal NoC and CELL processors. We have used their clock-cycle-true simulators for evaluating accurate results on performance modeling and QoS management.

The combination of the general application case with multiple dynamic video objects and its mapping onto a multiprocessor NoC provides an important generic mapping problem that gives highly relevant information for the design of many upcoming systems.

Chapter 3

Performance estimation and timing models

Since video objects vary in shape and behavior over time, the usage of platform resources is characterized by a dynamic consumption of computation and other resources. In order to facilitate faster and robust system design, an estimation of resource consumption is of vital importance. The chapter starts with presenting a computation model based on the actual coding parameters in order to enhance the accuracy. This is followed by a parametrical model for describing the usage of communication resources, which is equally important as computation modeling. The chapter ends with the fusion of various types of resources in one model, leading to a multidimensional resource-usage model. It is evident that such a multidimensional model is an attractive starting point for advanced Quality-of-Service control of executed applications, discussed later in the thesis.

3.1 Introduction

The parallel execution and processing of moving video objects in MPEG-4 coding forms a complicated design and optimization problem that cannot be solved analytically at compile time. A possible solution to this problem is to characterize the application execution using model descriptions of the resource usage. This chapter concentrates on achieving sufficient accuracy in those descriptions with the following two possibilities for applications of these models. The primary objective is to facilitate a faster design and more robustness of

a multiprocessor NoC. For example, various scenarios for splitting and distributing decoding tasks over the processor network can be explored with the model instead of executing rather lengthy clock-cycle-true simulations of such distributions. The secondary objective is to reuse them for the runtime estimation of the resource usage. In this way, the model descriptions can be used as a prediction for resource planning and possibly the corresponding quality control, thereby avoiding deadline misses or even system breakdowns.

A commonly used model for the processing of real-time audio and video applications is the Synchronous Data Flow (SDF) graph. The SDF graph is well studied for mapping DSP applications onto multiprocessor architectures (see e.g. the TV application in [29]). Many techniques to analyze and map such applications have been proposed [116, 106]. However, general DSP applications have *static* characteristics, whereas object-based video applications are more *dynamic* in behavior. Therefore, in this work we extend the SDF formalism to enable the expression of the dynamism of such applications.

One recent approach for addressing dynamics of applications in embedded system design was published in [41]. The approach explores the several operational modes of an application using a static SDF. Each operational point is called an application scenario, which are runtime selected after periodic intervals. Depending on the estimated WCET within an interval, the best scenario is selected. We support the adaptivity of this approach, but we want to be more accurate and avoid switching of SDF graphs. Instead, we pursue an SDF in which the dynamism is an integral part of it. Moreover, we strive for a finer granularity of decision making, so that the processor network can be planned for nearly full utilization (the utilization issue will be further explored in Chapter 6). The granularity will be chosen in the order of a VOP which has typically a size of 50-150 MBs. The finer granularity ensures that the system will have only small efficiency losses.

The performance prediction model that we introduce potentially serves multiple purposes with different forms of timing analysis, depending on the design and implementation approach. If it is implemented as a hard real-time system, the Worst-Case Execution Time (WCET) is required for the runtime scheduler to ensure that the job can meet its timing constraints [62]. The WCET approach guarantees the real-time behavior, however, it is too pessimistic for an application with dynamic resource requirements. This pessimism results in a continuous under-usage of the available resources, which increases the system costs. For this reason, we have adopted a *soft real-time* approach, which allows to exploit the system capabilities to the maximum.

The presented performance prediction models can be seen as an extension of the WCET analysis models proposed in [69]. For a soft real-time design approach, similar models have been used to analyze an alternative AS-profile implementation of MPEG-4 [11]. In our case, we concentrate mostly on the second type of real-time behavior, i.e. soft real-time, because it is more efficient and we observed that the hard real-time problems can be studied as a special case of the soft real-time domain.

The approach for the model construction is as follows. By a careful examination of the decoding algorithms for AS VO MPEG-4 decoding, we have found the performance-critical coding parameters directly affecting the execution time. Afterwards, these parameters are weighted and combined into the resource-usage model. The weighting coefficients are obtained from regression of clock-cycle traces of the actual execution of the decoder.

The sequel of this chapter is divided as follows. Section 3.2 defines in a formal way Synchronous Data Flow graph as a model of computation used for performance analysis. Section 3.3 discusses different techniques, such as WCET, Queuing networks, and statistical techniques, which are used for modeling of the system behavior and their applicability to our research. The details of our performance prediction model and positioning of arbitrary-shaped MPEG-4 decoding in the model abstraction is given in Section 3.4. Section 3.5 shows various aspects of the dynamism involved in the AS VO MPEG-4 decoding. Section 3.6 provides an overview of and discusses the individual task complexities. Section 3.7 addresses a parametrical model for a more optimal allocation of bandwidth, which is similar to the model for computations presented earlier in the chapter. Section 3.8 ends this chapter with the fusion of various types of resources usage in a single multidimensional resource-usage model.

3.2 Synchronous Data Flow Graph

This section introduces Synchronous Data Flow (SDF) graphs in more detail and formalizes the notations and definitions. In addition we provide an insight on a more restricted version of SDF, called Homogeneous SDF, which is used in the remainder of this chapter as a model of computation.

In the past, Kahn process networks [57] were extensively used for describing streaming applications. However, it is unsuitable for static analysis of bounded buffering. Synchronous Data Flow graphs (SDF), which were initially proposed in [66], are used for expressing the task-level parallelism in our model, which is suited for the exploration of multi-processor system design. The SDF graph is defined as follows.

Definition 3.1 Synchronous Data Flow graph (SDF)

The tuple (V, E, d, P, O, I) *defines a Synchronous Data Flow (SDF) graph, where*

- V *is the set of nodes (actors),*
- $E \subseteq V \times V$ *is the set of directed edges,*
- $d : E \rightarrow \boldsymbol{N}$ *is a function describing the number of initial tokens on an edge* $(u, v) \in E$*,*
- $P : V \rightarrow \boldsymbol{R}^+$ *is a function describing the worst-case response time of actor* $v \in V$*,*
- $O : E \rightarrow \boldsymbol{N}$ *is a function describing the number of tokens produced on edge* $(u, v) \in E$ *by actor* u *for each execution,*
- $I : E \rightarrow \boldsymbol{N}$ *is a function describing the number of tokens consumed from edge* $(u, v) \in E$ *by actor* v *for each execution.*

It is common to distinguish *data edges* and *sequence edges*, although their behavior is the same. Passing a token through a data edge represents the transfer of a block of data (in the following called *data token*) from the producer to the consumer. On a sequence edge, the tokens represent the enforced ordering between actors, e.g. the release of space in memory or a sequence of calls to a subroutine by the processor.

The above-defined SDF graph is characterized by a multi-rate dataflow description; consumption and production of data tokens are modeled by the amount of data tokens involved, thus they are constant integers known at the design phase. From the nature of our streaming application, we use a more restricted version of SDF, called *Homogeneous SDF*, abbreviated as HSDF. HSDF is a special case of an SDF, in which the execution of an actor consumes one token from each incoming edge and produces one token to each outgoing edge.

Figure 3.1 depicts an example of an HSDF graph in SDF notation. In the figure, we follow generally accepted SDF notation, however, since in HSDF always one token is produced and consumed, the numbers in this figure can be left out, thereby leading to an HSDF graph. The computations in an HSDF are represented by the nodes of the HSDF graph (labeled ellipses in the diagram), called *actors*. For example, an actor is a decoding task such as Context Arithmetic Decoding of the shape of a video object. The *edges* of the graph (arrows

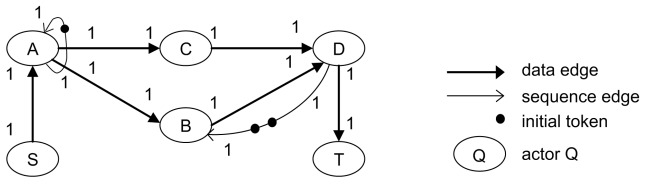


Figure 3.1: *Example of a Homogeneous Synchronous Data Flow graph. The data flow starts in source actor 'S' and writes output to target 'T'.*

in the diagram) represent dependencies between actors and carry tokens that are produced and consumed by the actors. Referring to the previous example, an outgoing data token would be the Binary Alpha Block (BAB) of the video object. Each edge indicates the direction of its token flow and may contain a few initial tokens, which are placed on an edge at the start of the execution.

The execution of an HSDF can be defined as follows. Each actor waits until there is at least one data token at each incoming edge. In this case, the so-called actor firing-procedure starts. In one firing, the actor performs computations on the contents of the first data block that is available at each data input. It takes a well-defined time interval to perform the computation of the involved algorithms. This interval, after compilation, depends only on the contents of the input data. Such a time interval is called the *firing delay* of the actor. It can be fixed or variable in different iterations. When the computations have finished, one token is consumed from each incoming edge and one token is sent to each outgoing edge. Conditional branches are not possible in this data-flow model.

The edges of the graph carry multiple tokens and take them from the producer, then deliver them in FIFO order to the consumer without any delay. A communication delay can be introduced by splitting an edge into two parts and introducing an extra delay actor in between. It is important to mention here that the SDF model has predictable timing behavior, which means that if the timing of the inputs is known, the timing of the outputs can be derived. Both aspects allow us to create more complete models, especially for a networked multiprocessor system. For more details on SDF graphs, the reader is referred to [66].

3.3 Performance analysis

The performance analysis of applications has to be carried out at different levels of abstraction during the whole trajectory of mapping multimedia algo-

rithms onto a desired architecture. The methodology for predicting actor-level resource usage (estimation) has been applied in video coding [110]. In our case, the performance prediction has the two objectives mentioned in the introduction of this chapter: the mapping analysis at design-time and the runtime modeling of the application resource usage for realizing an effective Quality-of-Service control.

Prior to presenting our approach, we first classify different performance techniques to position our model.

- *Worst-Case Execution Time (WCET) estimation* - The WCET estimation techniques are based on three major steps: (1) the extraction of control flow, (2) the low-level analysis of the target architecture, (3) calculation of the WCET, based on the explored signal path and extracted low-level details. The advantage of such a technique is the high accuracy of the construction of the model. However, it requires a complete execution graph [45], covering the full application. This requirement hampers the use of the WCET at the early phase of the system design, when this specification is not yet available, or is continuously subject to changes.

- *Analytical Methods based on Queuing Networks* - A platform is described by *servers* and *jobs*. Jobs are first inserted into *queues* and waiting until the server can handle their requests. A job is characterized with an arrival rate, a queue by an average number of jobs in the queue, and a server with the mean service time. The platform can be described and analyzed as an M/M/1 queueing problem [59]. This type of estimation based on queueing analysis omits the behavior aspects of jobs, like job dependencies and interaction.

- *Statistical techniques* - These techniques are data-driven approaches based on input data characteristics. This type of performance modeling is applied to the set of input data and produced output. The mostly used models are linear models [54]. The major advantages of such models are the abstraction from irrelevant issues at different stages of the design, and very fast estimation execution behavior using non-considered input data.

- *Simulations* - This is one of the mostly employed techniques, which is based on the construction of a simulation model that is typically executed on a host computer system [39]. It is used when the analytical methods do not allow the use of previous methods, because of the complexity of the exploration space. The results are highly dependent on a proper selection of the input data.

Considering the above classification, the problem of performance analysis of arbitrary-shaped video-object processing is not suited or too complex for WCET and Queueing Theory approaches. The WCET is a worst-case model and will lead to underused resources (e.g. following WCET, resources should be allocated for the full video resolution instead of a few macroblocks). The Queueing Networks approach will give a model that is probably not accurate enough for the behavior prediction of multimedia tasks, when the objects are relatively small, leading to unpredictable statistics.

Our technique presented in the next section is sequentially applying a simulation and statistical approach. The traces of actor execution times are obtained with a clock-cycle-true simulator of a target processor. Then a statistical approach is applied to obtain the correlation between input data coding parameters and the required computation.

3.4 Prediction model of execution time

In this section we will setup an accurate linear performance-prediction model for each actor that has to be executed on the multiprocessor platform. This performance-prediction model should be able to follow the dynamism in the processing over time with sufficient accuracy. For this reason, we base our model on the actual setting of coding parameters and the input data.

The main resources of the MP-NoC are the computing power of the processing tiles, the size of the memories and the network communication bandwidth. The computing power is modeled as the required processing time. At this stage, we abstract from details of the use of heterogeneous tiles that may exploit internal parallelism, which would influence the computation time. The abstraction can be justified by having an additional mapping step in which heterogeneous features are taken into account. Consequently, we model the processing tiles as general-purpose computing cores (e.g. RISC), operating at a certain clock frequency.

Figure 3.2 outlines the resource-estimation process. The resource estimation accepts inputs from the application and from the architecture. The input from the application is in the form of an executable specification. The specification distinguishes individual processing jobs and each job is divided into actors. The hardware architecture is represented by accurate simulation models, e.g. the instruction-set simulator for a processing tile, thereby enabling the measurement of the actor firing delay.

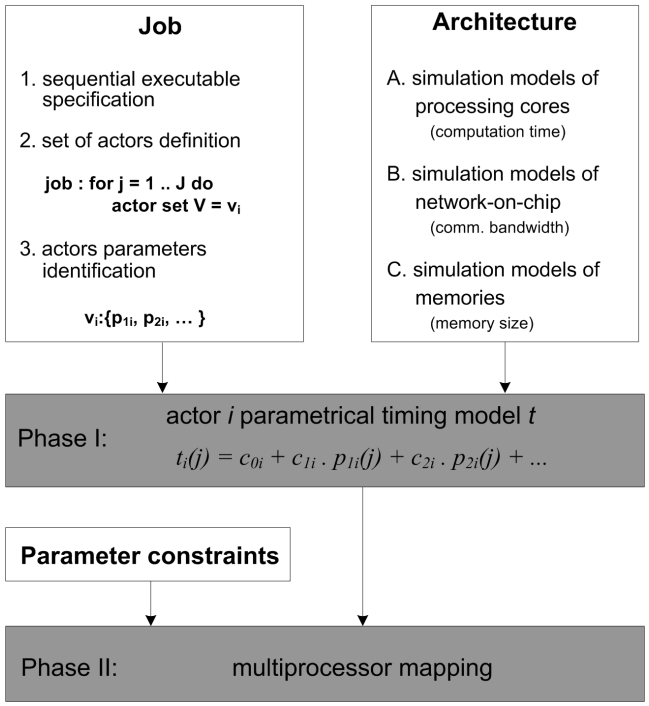


Figure 3.2: *Detailed view on the design flow. Algorithms in the form of jobs are executed on the architecture to find timing models used for multiprocessor mapping.*

At the timing-analysis stage, every job is characterized by application-specific performance constraints on the throughput of the job. In MPEG-4 applications, these constraints are strongly influenced by the chosen profile of the bitstream. For example, the number of shape macroblocks per VOP is limited by the MPEG-4 profile variable *video_ complexity_ verifier_ maximum_ buffer_ size*. The complete video decoding (application) activates a sequence of VOP-decoding jobs. Each job can use multiple processing tiles in parallel. We model a video-decoding job as an iterative "for" loop, taking J iterations to produce J data tokens of a video building block (like a macroblock).

The body of the "for" loop in Figure 3.2 is described by actors and the dependencies between them. We denote the set of actors as $V = \{v_i\}$. An actor can be described as a function in a programming language like C/C++, representing an atomic unit of computation that can be assigned to a processing tile. We assume that the firing delay of an C/C++ actor v_i can be expressed as a linear timing function t_i. The function t_i uses a set of variables, called input-data parameters $\{p_{1i}, p_{2i}, ...\}$, with fixed coefficients $\{c_{0i}, c_{1i}, c_{2i}, ...\}$ that depend on the architecture of the tile to which the actor is assigned (see Figure 3.2). The

linear timing function is based on a sum of various terms. Each term refers to a specific block of the software, and a special parameter can be introduced that is sometimes incorporated in the linear function indicating the number of instantiations of that software block. The individual timing functions t_i are combined in an overall timing-function vector $\boldsymbol{t}$, expressing all contributions to execution times. This timing vector represents the performance prediction-model adopted in our approach. Note that both the values of the individual timing functions and the parameters $p_i(j)$ may change in each iteration of the "for" loop.

In this chapter, we consider only Phase I in Figure 3.2 of the resource estimation, namely, the derivation of the parametrical timing model $\boldsymbol{t}$. We will use a job example to introduce a structure for the HSDF graph. Subsequently, we derive the performance prediction model for the arbitrary-shaped MPEG-4 decoding. Finally, the calculated firing delays of the obtained performance prediction model are compared with the corresponding firing delays on the clock-cycle-true simulator of the RISC processor.

3.4.1 HSDF Graph for Shaped Video-Object Decoding

To express parallelism between actors in the loop body of the job, we use a computation graph which is an instance of an Homogeneous SDF graph as discussed in Section 3.2. In the graph G, the parameters V and t are as defined in the previous section, while E is the set of edges that models the dependencies between the actors, and m (set of markings) is a function that assigns a non-negative integer number of initial tokens to every edge.

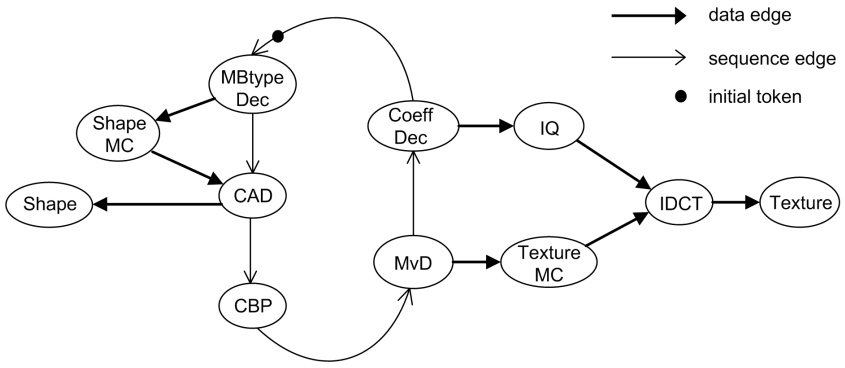


Figure 3.3: *HSDF graph of a Motion-Compensated AS VO MPEG-4 decoder.*

Figure 3.3 shows a computation graph for arbitrary-shaped video-object decoding. For simplicity, the de-multiplexing and other video tasks prior to the central part of the stream processing are omitted from the graph. Similar to MPEG-2, the processing involves a block-based coding technique. The depicted graph represents the processing required for one macroblock.

The motivation for choosing the computation granularity at the macroblock level is based on the following arguments.

- The processing nature of the involved video functions is rather different from task to task. This leads to the decision to go to a granularity level that is lower than the functional level. The functions are applied to the VOP layer or lower.

- The required computing power results leads to the distribution of the object decoding tasks over several cores. This means that the object decoding has to be distributed at task level. The granularity should be fine enough to efficiently load each processing core.

When comparing the computation graph with a pure texture decoding of rectangular video frames such as in MPEG-2, we easily identify the extension for the decoding of shape information. In the input bitstream, each macroblock starts with the shape information, followed by the texture data. Within the shape-decoding process we have identified the following actors: macroblock-type decoding (MBtype Dec), shape motion-compensation (ShapeMC), context arithmetic decoding (CAD), and decoding of the coded block pattern (CBP).

The complete VOP of an object-decoding job involves the processing of a grid of J macroblocks. Eeach j-th macroblock iteration ($1 \leq j \leq J$) of the job loop starts with the macroblock type decoding. Based on the macroblock type [52], it continues with the shape decoding or directly proceeds with the texture decoding. The ShapeMC actor computes the motion compensation for the shape part and provides the referenced MB shape for the CAD. The shape information for an VO macroblock is represented by a 16×16 Binary Alpha Block (BAB). As is depicted in Figure 3.3, the BAB should be sent to the output of the job (actor "Shape") for later texture processing (e.g. padding) and the scene composition. The CBP extracts information about which of the DCT blocks in the macroblock actually contain texture data. The texture decoding comprises five conventional and well-known steps: motion-vector decoding (MvD), coefficient decoding (Coeff Dec), texture motion compensation (TextureMC), Inverse Quantization (IQ) and Inverse DCT (IDCT).

We discuss now the splitting of actors and surrounding loops in the graph of Figure 3.3. At some occasions, we have split specific actors into parts based on their functionality. To correctly model the data dependencies between actors, we should include a number of sequence edges, due to the data dependency between the bitstream reading and the sharing of data structures. The sequence edge ensures that the connected actors are executed in a predefined order. For example, the texture part depends on the decoded shape information, so that we cannot use pipelining between them. The actors that have to read sequentially from the bitstream are: MBtype Dec, CAD, CBP, MvD and Coeff Dec. For this reason, we have introduced a loop surrounding these actors using sequence edges. We have observed possible parallelism with other actors, namely ShapeMC, TextureMC, IQ and IDCT. The output data consumers in Figure 3.3 are modeled with actors "Shape" and "Texture", covering the buffering and consumption of decoded data.

3.4.2 Construction of timing models

This subsection presents the construction of the timing models and their coefficients. Prior to giving the details of the model construction, we first discuss the approach and the applied conservatism in the estimation technique.

We have taken a conservative approach for determining the parametric function of the HSDF graph[1]. This means that the parametric function will provide an *upper bound* on the duration of any processing where the vector of parameters uses input-data-based settings. We expect that our estimation model for WCET analysis (like in [69]) can be adjusted to derive the upper bounds of the coefficient values. We obtain the coefficient values in two steps. First, we run a limited series of test sequences to obtain the initial set of coefficient values. Second, for refinement of those values, we perform optimization techniques, such as e.g. linear regression. This approach does not require the WCET tools.

If an actor exhibits a high variation of processing times, then the application designer should define a set of parameters that makes an adequate parametric function. In such a case, it is possible to estimate the coefficients using a linear regression technique. The linear regression employs coefficient values from the traces and obtains the final coefficient values, and it also provides a *confidence interval* $[\{c_{min}\}_i, \{c_{max}\}_i]$. The conservative estimate from the confidence interval is c_{max}, and therefore we use this estimate in the parametric function to ensure the conservatism of the HSDF. We use confidence intervals to obtain a measure of certainty about the value of the coefficients.[2]

[1] The consideration of this section also applies to SDF graphs as well.

[2] Sometimes it is justified to measure the maximum value of the processing time and use it as a linear term with the maximum value for the corresponding regression coefficient.

Experiments with multimedia decoding show a significant variation of the required processing time. In order to cope with this large variation, we introduce specific coding parameters into our estimation model. These decoding parameters vary in accordance with the execution time, so that our model follows the actual execution. Therefore, we define for each actor of the computation graph a set of coding parameters denoted by p_i, which are based on an algorithm exploration and the experimental evaluations.

After a number of design iterations, we have found that the computation time can be estimated accurately using a linear parametrical model, given by

$$t_i(j) = c_{0,1} + c_{1,i}.p_{1,i}(j) + c_{2,i}.p_{2,i}(j)... \quad . \tag{3.1}$$

As mentioned, we apply linear regression to derive the coefficients $c_{k,i}$. The parametrical model can function in several ways, e.g. as an accurate estimation of an actor firing time based on input data parameters. It can also function as WCET analysis, if the worst-case parameters are derived (e.g. the upper bound for the number of boundary macroblocks is specified in the standard). The model can be used to derive an average requirement on resource usage, based on an average number of parameter values, such as the average amount of DCT coefficients per block.

The choice of a linear parametrical model is motivated as follows. The usage of linear equations for performance modeling has been explored in the past. For example, in [6] it is already shown that MPEG-2 decoding can be well modeled with linear equations. Since a significant part of our decoding functions are similar to MPEG-2, the usage of such equations seems a valid assumption. A second argument is found in the complexity. It is generally known that computing linear functions on a computer is one of the most simple tasks. This will ensure that the computational costs involved with the model calculation will be limited. This will allow us to periodically use the model for consistency checking or prediction.

3.4.3 Derived timing models for AS VO MPEG-4 decoding

Prior to presenting the derived parametrical model, we provide more details on how we have derived it.

- The equations of the model are partly based on key parameters for MPEG-4 decoding. Some of the parameters are transmitted to the decoder (e.g. macroblock type) in the transmission format and others are not transmitted (e.g. number of AC coefficients in a DCT block), but

used in the algorithm as a local parameter having a significant importance. Both types of parameters are denoted as coding parameters in this section. The choice for using these coding parameters in our model, resulted from an iterative design procedure. The most likely coding parameter having an influence on the computing time was chosen and evaluated with respect to its influence. If the actual timing model was not accurate enough, i.e. less than 95%, another coding parameter was adopted and included in the equation. The coding-parameter choice was defined by algorithm analysis.

- To support the derivation of the timing model, we have added the generation of traces to measure the coding parameter values inside the MPEG-4 decoding algorithm, executed on a general-purpose platform. Next, we have measured the processing time of the actors on the target processor simulator (in our case the ARM7 armulator [1]). By the code inspection, the coding parameters and their importance was weighted to the actor execution. Because of using the clock-cycle-true ARM7TDMI simulator for processing tiles, the basic unit of the timing-function components is the ARM clock cycle. This means that our equations are based on the actual execution on the target processor, which gives realistic model with a potentially high accuracy.

- The third point is about usage of the test data set. The iterative design procedure for deriving the equations is a form of learning. The derivation of equations is based on a different data set than the experimental results on accuracy presented later in this section. For learning we have used the relatively long "Dancer" sequence with many different macroblocks, whereas for validation the sequences "Stefan" and "Singer" were used. The applied sequences originate from the MoMuSys project [93], which was a leading European project for AS VO MPEG-4 standardization.

The extraction of the timing model for AS VO MPEG-4 decoding was performed at the actor level, as described in the previous subsection. The result of using Equation (3.1) for each actor, together with measurements of coefficients as described in the above aspects, leads to a set of parametrical timing equations, which are modeling the complete execution of the job including all macroblock iterations. Following all these steps and performing the linear regression, the final result for the execution of the individual video-decoding actors leads to the following set of equations. The applied regression was based on the MATLAB 6.3 algorithm for linear regression.

$$
\begin{aligned}
t_{MBtypeDec} &= 913 + 1.34k\tau + 1.9k\tau^{'} \\
t_{ShapeMC} &= 53 + 4.17k\tau \\
t_{CAD} &= 14.13k + 135.12k\mu + 190N_{bits1} + 400N_{bits2} + 390N_{bits3} \\
t_{CBP} &= 13 + 111\xi + 97\omega + 26N_{empty} \\
t_{MvD} &= 413 + 980\tau + 2.9k\tau^{'} \\
t_{TextureMC} &= 210 + 3.11k\mu + 5.81k\tau \\
t_{CoeffDec} &= 790\varphi + 83N_{VLD-Bytes} + 105N_{DC} + 87N_{DC+} \\
&\quad +291N_{AC-NE} + 534N_{AC-E1} + 618N_{AC-E1} + 934N_{AC-E1} \\
t_{IQ} &= 1.42k\varphi + 14.9kN_{AC} \\
t_{IDCT} &= 74 + 37.18k\varphi_{\sigma} \\
t_{shape} &= 72 + 6.3k\mu \\
t_{texture} &= 153 + 16.3k\varphi
\end{aligned} \tag{3.2}
$$

Let us illustrate the meaning of the above set of equations in more detail. For example, let us describe the obtained timing function for the CAD actor. It takes 14.13 kilocycles (indicated by a k) to initialize the actor. If the macroblock is a boundary block, we assign $\mu = 1$, otherwise we assign $\mu = 0$. For $\mu = 1$, 135.12 kilocycles have to be spent on decoding the shape, plus 190-400 cycles for decoding of each bit contained in the arithmetic code. Depending on three possible actions to be taken for the arithmetic interval edges, the costs per bit slightly differ from each other. The semantics of each coding parameter are given in Table 3.1.

3.4.4 Validation of timing models

To validate the linear parametrical model, we have evaluated the relative model execution-time error of the timing model versus real execution. It is defined as the relative absolute difference ϵ between the calculated model execution time t_i and the measured value e_i of the real execution on the clock-cycle-true simulator of the target processor. The relative model execution-time error is specified by

$$\epsilon_{Ti,m,j} = \mid t_i(J) - \{e_i\}_j \mid / \operatorname*{mean}_j(\{e_i\}_j). \tag{3.3}$$

Table 3.2 portrays the relative model execution-time error for each of the actors for the validation test sequences. From the obtained results, we conclude that the largest average relative error is about 5.3%.

Parameter	Description
τ	1, if the macroblock belongs to Intra coded VOP; 0 otherwise.
$\tau^{'}$	NOT τ (this is logical inversion of τ).
μ	1, if the macroblock is of type 'boundary'; 0 otherwise.
$N_{bits1\ -3}$	Number of bits of type 1,2,3 (as defined in the standard) used to code shape.
ξ	1, if the macroblock is not transparent; 0 otherwise.
ω	1, if ξ=1 and at least one 8×8 block in the BAB is completely transparent; 0 otherwise.
$N_{empty\ -pix}$	Total number of first empty pixels in all 8×8 BAB sub-blocks (if pixels are checked in the scanning order), hence $N_{empty\ -pix} < 16 \times 16$.
φ	Number of non-transparent sub-blocks in the macroblock, hence $\varphi < 6$.
$N_{VLD\ -bytes}$	Number of the bitstream bytes shifted into the 64-bit local buffer, while reading VLD coded bits.
N_{DC}	Number of the non-zero DC DCT coefficients in the macroblock, thus $N_{DC} \leq 6$.
N_{DC+}	Number of the non-zero DC DCT coefficients coded by more than 1 bit, $N_{DC+} \leq N_{DC}$.
$N_{AC-\ NE}$	Number of non-zero AC coefficients coded by a "normal" VLC code.
$N_{AC-\ E1-3}$	Number of AC coefficients coded by an ESC code, using types 1, 2, or 3.
γ	1, if the macroblock is not transparent; 0 otherwise.
N_{AC}	Number of non-zero AC coefficients in the macroblock, $N_{AC} = N_{AC-NE} + N_{AC-E1} + N_{AC-E2} + N_{AC-E3}$, $N_{AC} \leq 6 \times 63$.
φ_σ	1, if $\varphi > 0$; 0 otherwise.

Table 3.1: *Parameters of the proposed AS VO MPEG-4 timing model and their semantics.*

Sequence	MBtype Dec	Shape-MC	CAD	CBP	MvD	Text-MC	Coeff Dec
Singer	1.47%	0.32%	0.86%	3.40%	0.27%	1.24%	2.87%
Stefan	2.04%	0.73%	0.74%	5.31%	0.47%	1.95%	4.34%

Seq.	IP	IDCT	shape	texture
Singer	1.31%	1.44%	0.52%	0.89%
Stefan	1.54%	1.94%	0.68%	0.97%

Table 3.2: *Relative error of the parametrical timing model at actor level of the graph presented in Fig. 3.3.*

Figure 3.4 shows an example of the model behavior for e.g. the Context Block Positioning (CBP) actor. The plotted curves exhibit the required processing time for a sequence of 60 macroblocks. The gray curve shows the timing model and the bold curve refers to the real execution. It can be seen that the model is quite accurate because both curves mostly coincide. According to Table 3.2, the worst-case accuracy has the CBP actor for "Stefan" sequence with an error of 5.3% only. The timing model provides a level of accuracy that is acceptable for soft real-time applications.

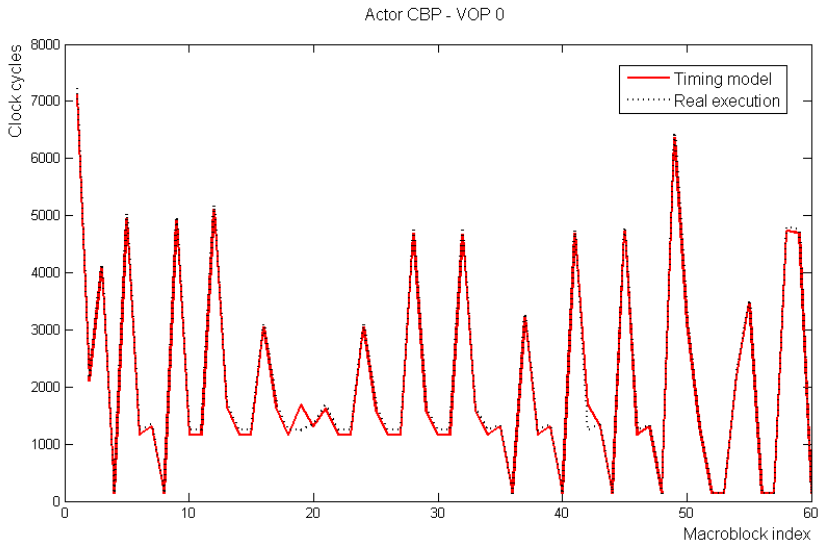


Figure 3.4: *Visual representation of the obtained timing model and real execution for 60 MBs of the CBP actor of the "Singer" sequence.*

3.5 Dynamic behavior of arbitrary-shaped VO

The dynamism in the required computation resources is presented and studied in established video-processing standards, like MPEG-2 [6, 20] and H.263 [50]. The presented dynamism mainly depends on the frame type and the scene contents and motion, resulting in a varying number of motion-compensated blocks. Apart from this dynamism, we have observed a much stronger dynamism in AS VO MPEG-4 decoding, due to the different macroblock types, the varying size of a video object and the variable number of objects per scene (see Section 2.4). For our architecture, we distinguish three different levels of dynamism.

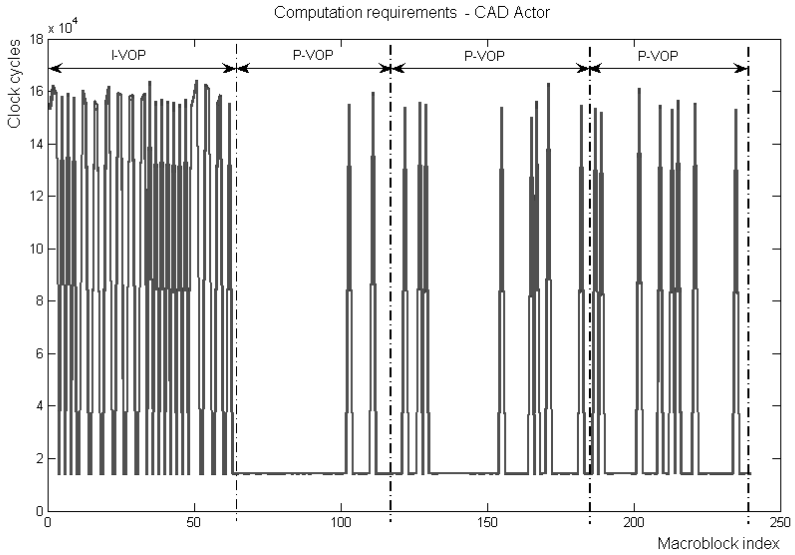


Figure 3.5: *Obtained characteristic of the CAD actor dynamism for a sequence of consecutive macroblocks. The VOP boundaries are indicated by the dotted vertical lines.*

1. *Actor level* - the required computation resources vary at the actor level, e.g. type of macroblock, presence of shape information, number of DCT coefficients.

2. *Job level* - video object decoding requires a variable number of loop invocations, due to the different number of macroblocks per VOP.

3. *Application level* - a varying number of jobs has to be activated, depending on the number of active video objects within the scene.

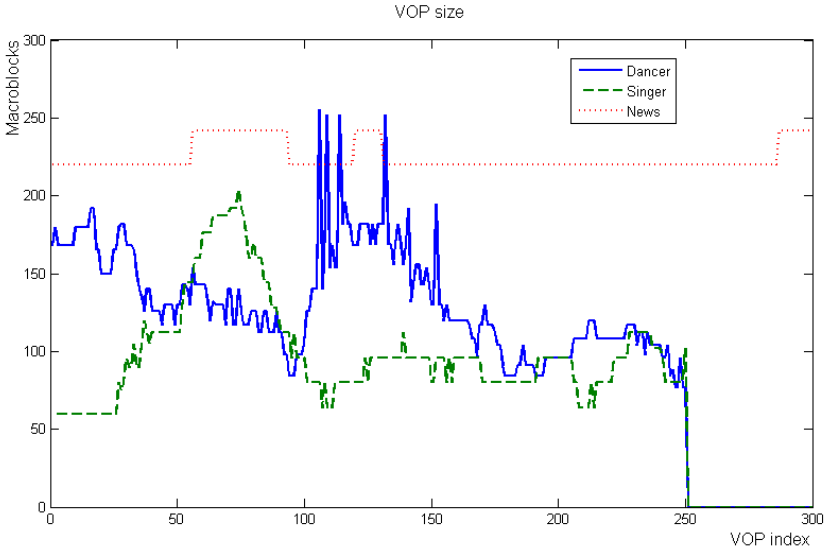


Figure 3.6: *Obtained characteristic of the CAD actor dynamism (CIF resolution).*

The examining of the AS VO MPEG-4 decoder at the actor level reveals the dynamism at the actor level and at the job level. At the application level, the AS VO MPEG-4 decoder would be instantiated multiple times and executed in parallel on the same platform (not discussed here further). For example, the study of the CAD actor portrays dynamism at both the actor and job level. In Figure 3.5, the processing of one macroblock is at the actor level, and the processing per frame is at the job level. The large variation at job level is seen when going from I-VOP to P-VOP, etc. Inside the I-VOP execution, we can notice also a large variation, but now at the actor level. The job variation is due to the number of macroblocks that may reuse the shape mask from the previously decoded motion-compensated VOPs. The measured overall ratio of the required computations between complete I-VOP decoding and P-VOP decoding equals for the most complex actor (CAD) to a factor of about 3.2.

Figure 3.5 provides experimental evidence for our conjecture that object-oriented coding leads to more dynamism than the conventional frame-based coding techniques. The dynamism at the job level, illustrated by Figure 3.6, is a new level of dynamism that was not present in MPEG-2 type of systems. In short, the MPEG-2 type of systems process fixed-resolution images and the corresponding number of blocks to decode, as compared to a high variability resulting from the varying size of arbitrary-shaped MPEG-4 video objects.

The study of the dynamism at the application-level is beyond the scope of research presented in this thesis. It is evident that the variability of the scene and corresponding number of video-object decoders introduce another level of dynamism. However, the hierarchical approach presented in Chapter 5 can be extended to deal with a varying number of jobs and to distribute the platform resources to jobs dynamically.

Given the presented accuracy of our timing model, the reader may come to the desire to reuse the parametrical timing model as a performance prediction model. This approach seems to be possible, but probably in limited cases. For MPEG-2, a statistical model of the resource usage can provide sufficient accuracy [109, 20], but it is limited to the actor level, which was defined at the video function level. We expect that our parametrical model is accurate for prediction modeling at the actor level and job level. It is difficult to foreclose the performance prediction at application level.

3.6 AS VO MPEG-4 decoding complexity

Having implemented the full standard and looking to the results of the previous section, at this point it is useful to discuss the decoding complexity of AS VO MPEG-4 decoding. In this section, we will present these results in the form of executed clock cycles required for completing the decoding job on a target processor.

1. Measurements of cycles and relative complexity
We have measured the execution time of the individual decoding tasks of the decoding job and also computed the total amount of required clock cycles. With these numbers, the average amount of clock cycles per macroblock was computed. The results of these measurements are portrayed by Table 3.3 and were measured for a GOV size of 12 VOPs. This table immediately shows the relative complexity of individual actors. The computation-demanding actors are Shape Motion Compensation (Shape MC), Context Arithmetic Decoding (CAD), Texture Motion Compensation (Text MC), Inverse Discrete Cosine Transform (IDCT), Repetitive padding (Rep. pad.), and Extended and Boundary Padding (EBP). Of these actors, CAD and IDCT are the most critical ones. One aspect of the complexity is the dynamism that was already discussed in the previous section. This dynamism can be also noticed in Table 3.3 as the CAD actor varies between 13% and 43% in relative complexity per sequence. This is explained by the strongly varying nature of objects in a video sequence and from one sequence to the other. For example, in the "News" sequence, the foreground object is rather static, whereas in the "Stefan" sequence, the tennis player shows large variations in pose and size. Similar variations in clock-cycle costs can be noticed for the ShapeMC and IDCT actors.

	Dancer	Singer	News	Fish	Stefan	Average
MB Type Dec	0.24%	0.32%	0.30%	0.26%	0.24%	0.27%
Shape MC	11.12%	16.07%	18.74%	9.18%	6.55%	12.33%
CAD	38.12%	15.65%	13.37%	33.79%	43.51%	28.89%
CBP	0.15%	0.22%	0.23%	0.16%	0.14%	0.18%
MvD	1.09%	1.71%	0.87%	1.35%	1.54%	1.31%
IP&IQ	4.08%	5.57%	6.35%	4.42%	3.64%	4.81%
Coeff Dec	2.67%	1.34%	0.63%	6.11%	2.07%	2.56%
Text MC	9.77%	13.02%	11.81%	9.92%	9.95%	10.90%
IDCT	13.90%	19.18%	21.80%	13.80%	12.75%	16.29%
Rep. pad.	9.33%	13.15%	13.04%	9.38%	9.02%	10.78%
EBP	9.51%	13.76%	12.86%	11.64%	10.58%	11.67%
Execution time (kcycles)	295,842	100,539	267,756	191,518	118,949	
Processed # macroblocks	2,064	720	2,640	1,520	671	
Average exec. / MB (cycles)	143,334	139,638	101,422	125,998	177,271	137,533

Table 3.3: *Distribution of complexity of individual tasks for one GOV of 12 VOPs at CIF resolution for various sequences and the required cycle counts for macroblocks.*

Table 3.3 shows the average clock-cycle count per macroblock at the bottom (averaged over all block types). This number enables us to compare it with results of MPEG-4 Simple Profile decoding and some results of the literature. The comparison with MPEG-4 Simple Profile is useful, because it is nowadays widely implemented in consumer devices. The average complexity of AS VO MPEG-4 decoding per macroblock within one GOV of our five test sequences is 137,533 clock-cycles. The MPEG-4 Simple Profile decoding complexity measured with the same clock-cycle-true simulator of the ARM7TDMI processor resulted in the average complexity of 73,406 clock cycles per macroblock. Hence, the decoding complexity of AS VO MPEG-4 decoding is 1.87 times higher than the MPEG-4 Simple Profile.

2. Discussion

This comparison result can be debated in several ways. For example, the number of video objects in a scene is not known beforehand by the decoder and is decided by the encoder. In the case of more video objects, the decoding complexity will be higher than in a simple case with few objects. This is because more shapes are involved in the decoding, more buffering is required and more covered macroblocks are processed, which are later overlapped by foreground objects. For this reason, the MPEG-4 standard limits the total number of macroblocks produced by the encoder. This limit is for Core

and Main profiles equal to two times the spatial resolution expressed in macroblocks. This limit is specified in the MPEG-4 standard by the parameter $video\_memory\_verifier\_max\_size$. Taking this parameter into account, the complexity of the AS VO decoding is calculated by

$$O_{AS\_decoding} = O_{MB} \times video\_memory\_verifier\_max\_size. \quad (3.4)$$

The parameter O_{MB} denotes the complexity per processed macroblock. To compare the complexity for the whole scene with multiple objects, we have used our averaged complexity per macroblock from Table 3.3 for the parameter O_{MB}. In this way, we have obtained an alternative comparison with MPEG-4 Simple Profile complexity: AS VO MPEG-4 decoding is then 3.75 times higher than the complexity of MPEG-4 Simple Profile decoding.[3].

3. Complexity of timing model calculation
Another form of complexity that is briefly discussed here is the involved complexity of regularly updating the measured computational costs using the proposed parametrical timing model. For an accurate profile of the computations, the model is executed for each macroblock. This involves the reading of the parameters, multiplications with the coefficients and adding the results together. The outcome may be useful for prediction of the involved computing at runtime to facilitate accurate resource management (e.g. in later chapters proposed as QoS). This concept was also considered in [41], but found to be too complex. We have verified with actual experiments whether this conclusion is valid for our case. Using the proposed clock-cycle-true simulator, the model calculation required 3,134 clock cycles against a average macroblock cost of 137,533 clock cycles, which is 2.3% overhead. This is small enough to be acceptable for runtime prediction. Furthermore, in the normal operation of the MPEG-4 decoder, also deblocking and deringing actors are enabled which reduces the overhead to only about 0.8%. Given the large potential benefit for quality control, this overhead is negligible.

4. Comparison to other standards
The above complexity comparison can be enlarged to other standards, although the numbers are sometimes not available or not suited for a fair comparison. An example is the recently published overview of the H.264 standard [120], stating that H.264 coding is 3 to 3.5 times more complex than the previous standards MPEG-2/H.263. This statement brings our experiment at the same level of complexity. The number of H.264 decoding is clearly an extension of MPEG-2 decoding, as it requires more precise motion compensation and involves loop filtering and extra intra prediction modes. The arithmetical de-

[3]Our estimate is that the complexity of MPEG-4 Advanced Simple Profile is within this range and closer to MPEG-4 Simple Profile.

coding is comparable in nature and complexity. In our case, we have shape motion compensation and the various padding techniques as extra complexity compared to MPEG-2. This globally explains why the order of magnitude of the complexity increase is about the same. Further comparisons and discussions on complexity are given at the end of this thesis in Chapter 7, where we report on several application experiments. Concluding, the AS VO MPEG-4 decoding complexity is in line with the increase of the complexity of recent video coding standards and as such can be seen as a representative experiment for platform evaluation.

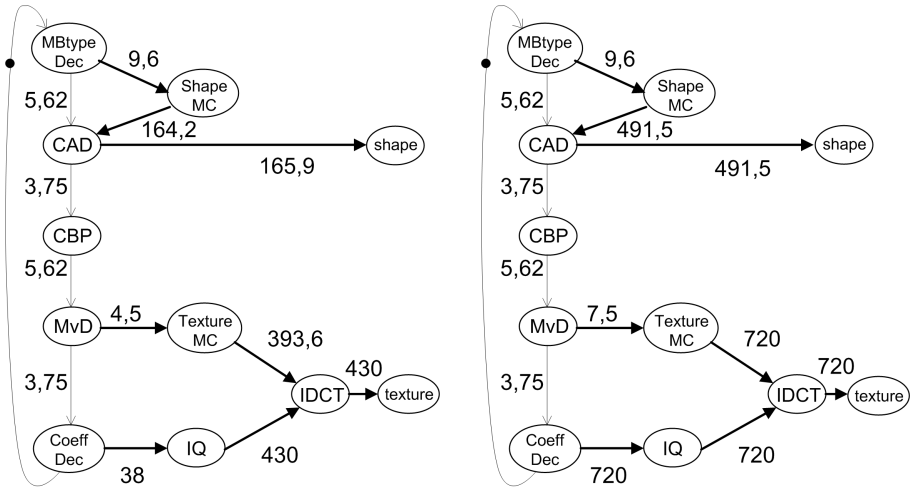
3.7 Parametrical model of communication resources

In this section it will be shown that the dynamism that was explored earlier for computation resources is valid for other types of resources as well. Similar to the WCET approach [11] for the computation resources, the same approach can be applied to other types of platform resources (memory, communication bandwidth), but again it results in inefficient resource allocation (worst-case allocation). For this reason, we also study the usage of the communication resources and propose a model that fulfils requirements on the accuracy and model complexity.

The timing properties of the individual actors needs to be completed with the bandwidth requirements between individual actors, in order to provide complete timing analysis of SDF on a multiprocessor platform [96]. A possible bandwidth limitation can result in postponing the activation of an actor execution, so that the overall execution time is larger and/or the timing analysis is less precise.

The parametrical models developed in the previous section are also adopted to describe a model for the bandwidth usage. For example, Figure 3.7 illustrates the large bandwidth difference between the outcome of the worst-case model of communication resources versus the actually needed resources for the same video stream.

When analyzing the MPEG-4 decoding, two types of connections are identified in the communication graphs. The control data and synchronization tokens are always presented and represent also *input-independent* connections. For example, the MBtype Dec actor communicates the macroblock type and the bitstream position to the CAD actor for every macroblock. This holds also for the ShapeMC actor, but now some extra information (e.g. reference position of the macroblock) is required. These connections have very static characteristics and we model them with a constant number of tokens, also due to the low amount of data as compared to the streaming parts.



(a) Worst-case estimation. (b) Actual requirements.

Figure 3.7: *An example of communication resource-usage model, (kByte/s) for AS-VO decoding (256×64 @ 30 Hz).*

Communication via *input-dependent* connections is highly inefficient, if resources are allocated based on a worst-case analysis. For example, the bandwidth required between actors CAD and Shape is in the worst-case approach 491.5 kByte/s compared to the actually needed 165.9 kByte/s (see Figure 3.7). Our results on the analysis of the MPEG-4 decoding show a factor of 2.5 between the required and the worst-case approach. Similar to Equation (3.1), we propose to model the communication resources with a linear parametrical equation, by

$$b_i(j) = n_{0,i} + n_{1,i} l_{1,i}(j) + n_{2,i} l_{2,i}(j) + \ldots \quad . \tag{3.5}$$

The variable n_i stands for the weighting coefficients of the term contributing to the communication, and the variable l_i denotes a specific input-data parameter, such as the number of macroblocks per VOP.

3.7.1 Derived bandwidth models for AS VO decoding

Similar to the derivation of the parametrical timing model, we comment on details of the derivation of the bandwidth model.

- The equations of the model are based on the type of macroblock and the BAB type. This can be derived by examining the graph of Figure 3.7 and

the meaning of the communication edges between actors. For example, the bitstream position is communicated from the actor MBtype Dec to the Shape MC actor, if a block is coded with motion compensation. In this case, the referenced block from the previously decoded frame is fetched, otherwise only the bitstream position is communicated to the actor CAD. Hence, the resulting bandwidth clearly depends on the BAB having motion compensation (the bandwidth switch parameter ξ in Table 3.4). The other parameters in the bandwidth model are derived in the similar way using the edges in the diagram.

- The amount of data transferred between actors were measured by instantiating the MPEG-4 decoding algorithm and including a special logging function to store the amount of involved data at the beginning of each actor implementation.
- The numerical values in Equations(3.6) and (3.7) are independent of input data. These numbers can be derived by the examining of data that has to be transferred between two actors. For example, if a decoded block is transferred, then the involved data structure is communicated of which the format is known. This leads to the indicated coefficient values.

The derived bandwidth model for the our decoding graph is split in two sets of equations. First, the set of Equations (3.6) provides details for the connections between actors that are independent of the input data. The numbers reflect the communicated flags, block types, etc. Second, the set of Equations (3.7) presents connections that vary with the input data. A detailed description of the corresponding parameters is given in Table 3.4.

$$\begin{aligned} b_{MBtypeDec,CAD} &= 5.62 \\ b_{MBtypeDec,ShapeMC} &= 9.6 \\ b_{CAD,CBP} &= 3.75 \\ b_{CBP,MvD} &= 5.62 \\ b_{MvD,CoeffDec} &= 3.75 \end{aligned} \tag{3.6}$$

$$\begin{aligned} b_{ShapeMC,CAD} &= 2 + 320 \cdot \xi \\ b_{CAD,shape} &= 256\Psi \\ b_{MvD,TextureMC} &= 4\omega \\ b_{TextureMC,IDCT} &= 3072\omega \\ b_{CoeffDec,IQ} &= 14\theta\chi \\ b_{IQ,IDCT} &= 384\chi \\ b_{IDCT,Texture} &= 384\chi \end{aligned} \tag{3.7}$$

Parameter	Description
ξ	1, if the macroblock shape is motion compensated; 0, otherwise.
Ψ	1, if the macroblock is boundary type; 0, otherwise.
ω	1, if the texture is motion compensated; 0, otherwise.
θ	Number of non-transparent sub-blocks in the macroblock, hence $\theta < 6$.
χ	1, if the macroblock is not transparent; 0, otherwise.

Table 3.4: *Parameter description of the bandwidth model.*

Let us briefly discuss an example of the parameters used in the data-dependent set of equations. For example, if the $j-th$ macroblock contains encoded shape information and belongs to an inter-coded VOP (P- or B-VOP) ($\xi = 1$), the required communicated data between the ShapeMC and CAD actors is equal to

$$b_{ShapeMC,CAD}(j) = 2 + 320 \cdot \xi \quad [Bytes]. \tag{3.8}$$

When knowing the distribution of the macroblock types (boundary, opaque, etc.) within one VOP, we obtain for the complete VOP the following expression

$$b_{ShapeMC,CAD}(j) = (2 + 320 \cdot \xi) \cdot P_{boundary}, \tag{3.9}$$

where $P_{boundary}$ denotes the fraction of the boundary macroblocks.

Similar to the discussion on using the timing model for runtime prediction, it is possible to apply the bandwidth model for runtime bandwidth prediction. In order to find the prediction of the required communication bandwidth for the next VOP, we use a modeling function similar to Equation (3.5) for every connection and multiply it by the number of macroblocks inside the VOP. The numerical results and details of our experiment on the bandwidth parametrical modeling are presented in the next subsection.

3.7.2 Validation of bandwidth model

To validate the bandwidth model, we have compared the bandwidth model output values with the actually used bandwidth for execution of the decoder. The model output and the bandwidth measurement were evaluated after each VOP. The final results are portrayed by Figure 3.8. The validation simulations show that our modeling technique for the bandwidth requirements was

consistently 4.7% too low. For this reason, we added a compensation bandwidth of 5% to the model results. This difference of 4.7% in the bandwidth is explained by the fact that the VOP header information is not considered in our bandwidth model. Our model is based on macroblock-based execution, so that periodic higher layer description information is omitted.

Figure 3.8 compares the output of the proposed bandwidth model with the worst-case approach for communicating different number of macroblocks. The model with its compensation allocates 2.5 times less bandwidth than the worst-case approach. This result has significant importance. It means that a large bandwidth reduction can be exploited for enhanced parallelism in the execution on MP-NoC. Probably, this obtained result has more practical impact than the parametrical timing model, because in a networked system bandwidth is becoming increasingly the most scarce resource in the system. We can also benefit from this result in terms of runtime bandwidth prediction. In [81], it was found that the ratio between the various macroblock types varies smoothly over one scene, which means that variations between consecutive VOPs are relatively small. This results in a stable prediction of the required bandwidth if recent system execution parameters are used.

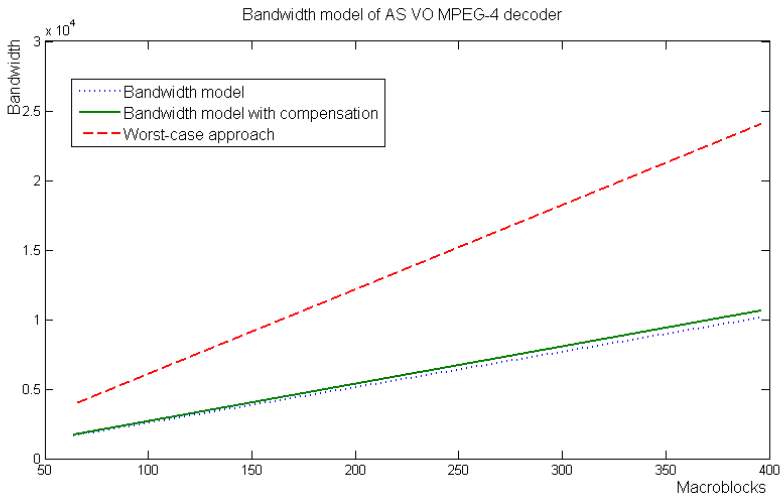


Figure 3.8: *Example of our modeling function compared to the worst-case approach.*

3.8 Multidimensional model of resources

The timing and bandwidth models presented in the previous sections are important to analyze an application per individual resource. However, the design of a processor with multiple computing cores (multiprocessor NoC), requires that the analysis of the actual resource usage becomes a multidimensional problem. In this section, we provide a generalized concept for modeling the resource usage of jobs with the purpose to facilitate Quality-of-Service management of multiple jobs. The intended computing platform is a multiprocessor NoC. This concept for multiprocessor systems is an idea and was not validated, because of lack of time. Since it is considered useful for further research, we present it here.

Our objective is to provide a multidimensional model of the resource usage of a job at different quality levels and additionally, a model of available system resources that serves our control management. This concept of using two models, i.e. the application resource usage and the platform resources, can be applied for runtime QoS management that will be presented in Chapter 5.

3.8.1 Job model at different quality levels

Let us now outline the resource-usage model for one job. Formally, the range of possible quality settings mapped into a vector $\mathbf{q_i}$ of a job i leads to a job resource-consumption R_i for a resource type J. The resource consumption is described by a function $R_{i,J}(\mathbf{q_i}) = f_J(\mathbf{q_i}, d_p)$, where d_p is an arbitrary input-data parameter that mostly influences the complexity of the computation. For example, the parameter d_p equals the size of a video object in terms of macroblocks. The function f specifies the requested amount of resources for a particular job. The resource type is $\mathbf{J} \in \{\mathbf{C}, \mathbf{D}, \mathbf{I}, \mathbf{B}, \mathbf{T}\}$, where $\mathbf{C}$ denotes the computation resources, $\mathbf{D}$ the data memory per actor, $\mathbf{I}$ the instruction memory per actor, $\mathbf{B}$ the required communication-port bandwidth, and $\mathbf{T}$ the bandwidth on each connection between a pair of actors.

For one job i, we specify the quality setting, so that for a set of jobs the set of chosen quality values leads to a chosen quality vector $\mathbf{q_c}$, in which the vector components refer to the chosen quality settings of individual jobs. Similar reasoning can be held for the required resources of $R_{i,\mathbf{J}}$, leading to a vector of resources $\mathbf{R_J}$. For example, when J refers to computations only ($\mathbf{J} = \mathbf{C}$), $\mathbf{R_C}(\mathbf{q_c}, \mathbf{d_p})$ is representing the required vector of computations for a set of jobs at chosen quality level $\mathbf{q_c}$ and input-data dependence $\mathbf{d_p}$. The vector $\mathbf{R_C}(\mathbf{q_c})$ can be used for finding the *accumulated* computation costs per set of jobs, executed at quality settings $\mathbf{q_c}$. If in the mapping this vector of required computation costs is compared and matching with the available resources, the allocation can be considered.

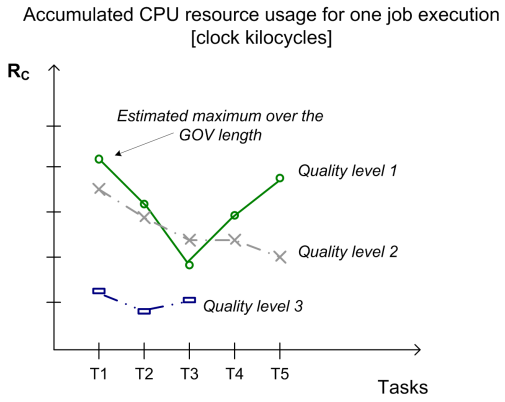


Figure 3.9: *Model of different quality levels per resource (J=C).*

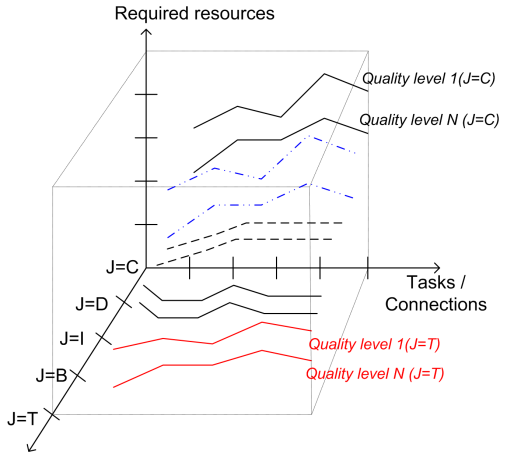


Figure 3.10: *Multidimensional visualization of required resources per quality level per resource.*

Figure 3.9 portrays an example of the computational requirements of a job, which is composed of several tasks, at different quality levels. The points represent the maximum requirements over the reservation period (in our case, the GOV length). In this example, when comparing quality level 2 with Quality level 3, the number of resources and the number of tasks is lower for Quality level 3. At Quality level 3, tasks T4 and T5 are even switched off. At the higher Quality level 1, task T3 has a lower resource requirement than for Quality level 2. However, this is not a typical case. Figure 3.10 visualizes the resource estimation for all different types of resources as described above.

3.8.2 Available and used system resources

The model of resource usage of a job has to be completed with the model of the usage of platform resources. The system resources that are available for a job execution are modeled and described in the following. We model the available computation and storage resources per processing tile. If P denotes the set of processing tiles, then we denote the total available resources as follows: computation resources of a tile as P_c, data storage capacity as P_d, instruction storage capacity as P_i, and communication port capacity as P_b. The communication availability is modeled per pair of processing tiles by the amount of data that is communicated per time unit as $D_{pa,pb}$. Both models, i.e. the available and used system resources, are visualized in Figure 3.11.

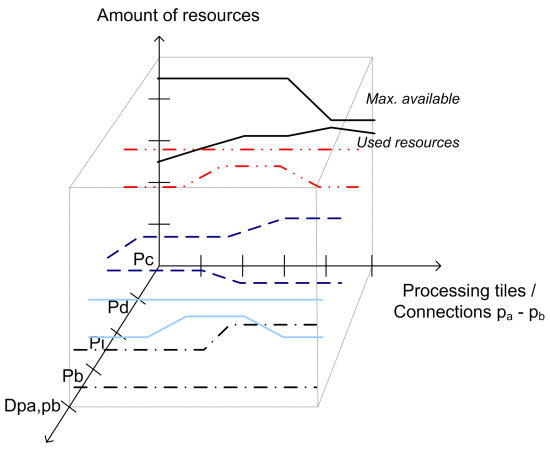


Figure 3.11: *System resource model, with explicitly modeled resource consumption extracted from runtime operation or allocated resources.*

We have modeled the decoding process of the AS VO MPEG-4 decoder with the objective to map this application on a multiprocessor network-on-chip (MP-NoC). The obtained performance prediction model at different quality levels is based on a set of linear equations (as presented in Sections 3.4 and 3.7), using parameters depending on the actual I-VOP and P-VOP bitstream characteristics of the MPEG-4 decoding. These models are connected to QoS management that will be discussed in Chapter 5.

The principal advantage of our technique is that the new mapping is evaluated analytically based on a set of linear equations instead of executing a clock-cycle-true simulation for each mapping. Consequently, it significantly decreases the complexity of the actor-to-processor assignment analysis. The conservative

way in which we derived our model coefficients supports the reliability of the model results and a possible usage of this model for a job with hard real-time constraints.

3.9 Conclusions

This chapter has started with formalizing the Synchronous Data Flow (SDF) graph and presenting an execution model for a restricted version of SDF, called Homogeneous SDF with consuming one data token from each incoming edge and producing one data token to each outgoing edge. The Homogeneous SDF graphs are easy to analyze with respect to throughput and deadlock occurrences. Besides these aspects, the token-based communication between actors fits well with the AS VO MPEG-4 decoding application, where execution is based on consecutive macroblock processing.

The essential part of this chapter focuses on the performance analysis of video processing algorithms modeled by HSDF, aiming execution on a multiprocessor platform. In our proposed approach, each video task called actor, is assigned with a parametrical function modeling the computational requirements. For simplicity, we have described the requirements in a linear function so that analysis becomes easier. It was shown that the overhead was a few percent or less than one percent if all video functions were activated. Each term in the summation within a model equation depends on (1) the coding parameters having input-data dependency and (2) coefficients representing the dependency of the involved processing on the target CPU. The experimental validation of the proposed parametrical timing model showed that the deviation of the actual execution time is few percent only. It was found that obtained timing model has a maximum deviation of 5.3% from the real clock-cycle-true execution on an Æthereal NoC with ARM7 cores.

This type of performance modeling is useful for analyzing several types of behavior. First, it can be used as an accurate estimation of an actor execution time based on input-data parameters. Second, it can function as WCET analysis if the worst-case parameters are derived (e.g. the maximum number of boundary macroblocks per video object is specified in the standard profile). These kinds of analysis can serve both types of execution: a hard real-time execution with a WCET approach, or the soft real-time operation with an accurate modeling.

Besides modeling of execution time, the allocation of communication resources is at least as important. Similar to the timing models, we have presented a technique to model requirements on communication bandwidth depending on

input data. Fortunately, it has been shown that it is possible to model the communication usage in the same way as computations, namely with a set of linear parametrical equations. The accuracy of this model is equally accurate or even slightly better. The comparison with the mostly used worst-case approach for communication resource allocation revealed that it saves an impressive factor of 2.5 on bandwidth consumption. This result has significant value, as many networked systems executing multimedia applications are increasingly bandwidth-constrained.

At the end of this chapter, we proposed a generic concept for combining parametrical resource usage models of several resources simultaneously into one model. This multidimensional model is intended to be used by a Quality-of-Service management system that will be presented in Chapter 5.

The analyzed AS VO MPEG-4 decoder shows dynamic behavior at the actor, job, and application level. Experiments showed that the actor and job level dynamism was visible in the required clock cycles for e.g. the CAD actor. The size of a video object is a critical factor in the dynamic behavior of the application. The complexity analysis revealed that the relative complexity of the actors in the graph varies considerably with the input scene. The AS VO MPEG-4 decoding application is considerably more complex than MPEG-4 Simple Profile coding and approaches the complexity of H.264 coding. The nature of the decoding processing tasks can also be found in the encoder, so that we envision that the derived models can be adapted to support also modeling of the corresponding encoding application.

Chapter 4

Algorithmic modification for enhanced parallelism

The direct transition of functional blocks to video tasks (actors in the SDF model) does not provide an optimal mapping, e.g. some tasks are designed to function at macroblock level and others at frame/picture level. The detailed analysis of individual video tasks and modification of the coding algorithms towards a unified, macroblock-based computation is the first result of this chapter. Secondly, we discuss the result of mapping this macroblock-based computation on a tile-based system, where we have achieved an efficient execution (for some tasks improvements about 70%). The final part of the chapter addresses the limitation of having a small data memory within individual processing tiles. For this purpose, we study a memory-rich application part of MPEG-4 coding, i.e. background sprite decoding. For the platform, we have selected, as an experimental NoC, a CELL processor simulator that encapsulates 8 processing elements where the local memory is limited to 256 kByte. The re-design of background-sprite MPEG-4 functions gives a mapping that satisfies the local memory limitations and it also provides a more efficient execution compared to the original design.

4.1 Introduction to uniform processing and sprite coding

The original algorithm for AS VO MPEG-4 decoding [52] was standardized without having any platform in mind. Therefore, the straightforward imple-

mentation following the MPEG-4 standard is not optimal for the direct mapping on a multiprocessor platform. The availability of a plurality of processing cores speedsup the processing when exploiting several types of parallelism. This parallelism is not visible in the original algorithm. For example, a potential split of the multiplexed DCT blocks during texture processing into individual color components, which are processed on a single processor, would introduce only extra communication and buffering overhead, while it would not speedup the execution. In contrast with this, the simultaneous processing of individual texture components on multiprocessor systems would contribute to an increase of the overall throughput. The improvement factor is influenced by the length of the critical path in the computation graph. In the AS VO MPEG-4 decoding graph using 4:2:0 color sampling, the critical path is the processing of the luminance component since this involves the most intensive processing.

The parallel execution of advanced multimedia coding is a topic of continuous study. Li *et al.* [68] discuss the encoding approaches of motion estimation on a parallel bus network. They exploit the granularity of the load partitions and associated overheads for minimization of the overall processing time. An alternative coding architecture is proposed by Fang [35]. This system is processing all bit planes in parallel in order to minimize the state memories in a JPEG-2000 encoder. The platform is based on a reconfigurable FIFO architecture. The common element in these studies is elegant exploitation of the granularity in processing, but the processing data itself is still based on conventional rectangular video pictures. Besides the parallel processing of individual color planes, we will explore further parallelism in this chapter by modification of the original algorithm using task splitting and introducing the uniform granularity.

The first part of this chapter addresses two algorithmic modifications: task splitting for shortening the critical path and introducing a uniform granularity in the processing, which are both improving the processing efficiency. It is shown that when choosing a uniform granularity of processing, we can avoid frame buffers between computation tasks and therefore reduce latency. We introduce a synchronization mechanism that allows the processing of the *extended Padding* and postprocessing filters (*deblocking & deringing*) at block level. Moreover, with task splitting, it can be expected that a well-chosen distribution of the data over the multiprocessors will add task-level parallelism that increases the system throughput. Finally, because of the multiprocessor, we exploit the inherent parallelism of the individual color components in the video signal. It will be shown that a substantial reduction of computing power can be achieved, combined with a lower latency.

The second part of this chapter explores a mapping on a multiprocessor system that has a local memory constraint for the individual processors. More specifically, we study a mapping of the memory-rich MPEG-4 sprite-decoding algorithm on a CELL processor [95]. From the architecture point of view, the CELL processor is a multiprocessor System-on-Chip similar to our tile-based platform. Our studies have revealed that a CELL processor system [38] provides high computational resources with low power consumption. However, the current implementation poses a limitation on different types of resources, mainly on local memories for most of the tiles in the CELL architecture. The key problem is that the MPEG-4 sprite decoding requires a large memory for buffering the sprite data. Therefore, we present a new sprite-decoding algorithm that reduces the memory cost of such decoding with a factor of four. Additionally, our algorithm offers the possibility of high-level data parallelism and consequently contributes to an increase of throughput rate.

The chapter is organized as follows. The first part deals with the mapping of the arbitrary-shaped MPEG-4 decoder onto an architecture requiring a unified processing granularity, including limitations on buffering between tasks. Section 4.2 discusses three generic forms of parallelism that should be considered when creating a parallel implementation of a multimedia application. Section 4.3 outlines the original MPEG-4 algorithm and its inefficiency for a straightforward mapping. Section 4.4 presents the modification of the repetitive padding task in MPEG-4 coding. The modifications of the extended padding algorithm and the post-processing filters are presented in Section 4.5. Section 4.6 describes the application of data-level parallelism principles and presents the complete redesigned computation graph. The last part of the chapter in Section 4.7 introduces the redesign of the background-sprite MPEG-4 decoding algorithm, in order to execute it on a tile-based architecture with small local memory. Section 4.8.1 presents the sprite-reconstruction principle and addresses the MPEG-4 standard decoding. Section 4.8.2 gives our new decoding algorithm while Section 4.9 provides details on the corresponding data structures. Section 4.10 describes the experiments and results of this study.

4.2 Parallelism Overview

This section addresses three generic forms of parallelism which are all briefly discussed in the framework of multimedia video coding. It is assumed that a video coder can be described with a flow graph as presented in Section 3.2. All tasks can be executed on different processors and depending on the nature of the algorithm, a parallel execution can be realized. At the end of this section, we also provide a strategy for extracting parallelism.

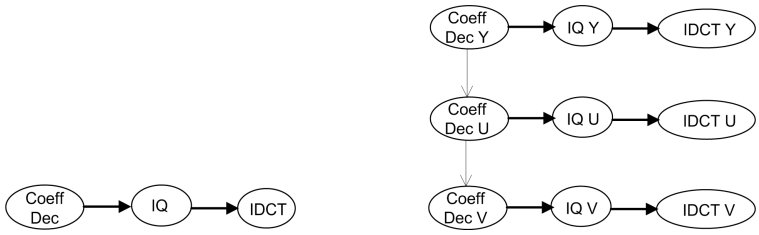
4.2.1 Task parallelism

An application is often described by a block diagram in which the relations between the different processing algorithms composing the application, are depicted. Regularly, processing blocks in the diagram represent algorithms which can operate concurrently on different sets of data. For example as in Figure 4.1, the Video Header parsing task in an MPEG-4 decoder can already decode data of a different Video-Object plane (VOP) that the Shape & Texture task is processing. This is again different from the data that the Rendering task is using within the specific time period. In this way, the different tasks from the block diagram are executed in parallel. In practice, the block diagram can serve as a starting point for extracting so-called *task-parallelism*. Not all tasks will have the same computational requirements (i.e. require the same amount of processing time). Using profiling and analysis techniques, the most computationally-expensive tasks can be identified. To increase the amount of task parallelism, these tasks should be subdivided into smaller ones. This typically results in a chain-structured set of tasks as shown in Figure 4.1. As a consequence, the transformation will result in a reduction of the execution latency of the application.

Figure 4.1: *An example of task-level parallelism for a simplified view on arbitrary-shaped video-object decoding, where each task is executed in parallel.*

4.2.2 Data parallelism

Data-level parallelism can be considered to increase the throughput and decrease the latency of an application. The idea behind data parallelism is to perform the same transformation on different data elements in parallel. For instance in an MPEG-4 decoder, it is possible to perform the inverse quantization in parallel form, e.g. by exploiting the separation of the luminance and the two chrominance planes. Figure 4.2(b) shows this kind of data parallelism that can be extracted from the computation chain shown in Figure 4.2(a). Alternatively, two chrominance planes may be combined into one color plane. Hence, a typical case is that the computation path that processes the luminance plane is the critical path. It should be considered whether more task-parallelism can be found in this chain to further increase the throughput.



(a) Original graph.

(b) Unrolled graph for each color component.

Figure 4.2: *Data-level parallelism for processing of texture.*

4.2.3 Communication granularity

A third aspect that can be used when extracting parallelism from an application, is exploring the best *granularity* at which data is communicated. For example, an inverse quantization task can send data at the level of individual coefficients to an inverse IDCT task, or it can send the data at the granularity level of complete blocks or even frames. The advantage of using larger grains of data is that the communication efficiency increases due to the smaller overhead in communicating the data between the tasks. However, tasks may have to wait longer, i.e. be idle, while they are waiting for data. Choosing the correct level of granularity at which data is communicated between tasks is important to prevent tasks from waiting and avoid spending too much time on synchronization.

4.2.4 Strategy to extract parallelism

In [108], an analysis technique for identifying task-level parallelism in applications is presented. The article presents a set of concurrency measures that help a designer in making a trade-off between the three types of parallelism discussed previously. Along with those concurrency measures, a strategy to extract the parallelism is now presented. As a first step, the application is profiled to identify tasks with a large execution time. These tasks are the computational bottlenecks that should be resolved by splitting these tasks into a sequence of computationally less intensive tasks. This step introduces additional task parallelism. The strategy continues with identifying candidate tasks for the extraction of data parallelism. The structure of the dataflow graph is used to find these tasks. A designer can use this information to exploit data parallelism that may be present in the application. The parallelism extraction strategy then continues with finding the right trade-off between the amount of time spent on the communication and the time spent on executing tasks,

i.e. the optimal communication granularity is determined. After these steps, all potential sources of task-level parallelism are considered. However, the resulting dataflow graph may contain many tasks that have a low requirement for the computational resources, thus these tasks may be idle for considerable time periods. Dealing with a large number of this type of tasks can complicate the mapping of the dataflow graph on the computational resources. It is often preferred to recombine some of the tasks to obtain a balanced workload for the processing cores of the platform. The last two steps of the parallelism-extraction strategy deal with this issue in an iterative way. Summarizing, the strategy involves the following steps:

1. Identification of *computational* bottlenecks and consecutive splitting of critical tasks;
2. Exploration of *data* parallelism for individual tasks;
3. Optimization of *communication* granularity between tasks.

4.3 Mixed granularity in AS VO MPEG-4 Decoding

Let us first briefly discuss the details of the original AS VO MPEG-4 decoder for both Intra- and Inter-coded VOPs. Figure 4.3 outlines a distributed version of a computation model for an AS VO MPEG-4 decoder. The final visual scene can be composed of several VOs. The decoding starts with the *Shape and Texture Processing* (as presented in detail in Section 3.4.1) at the left side of Figure 4.3, followed by *Extended* and *Boundary padding*, then applying the *Deblocking* and *Deringing filters* and providing the final shape and texture data to the *Frame renderer*. The renderer is a shared task and composes the original scene from the video background-sprite image and several VOs superimposed on it.

In order to come to a decision in choosing the communication/processing granularity for MPEG-4 decoding, we briefly examine the nature of the processing tasks. The key property of a macroblock containing shape information is that it consists both opaque and transparent pixels. For correct motion compensation of succeeding P- or B-VOPs, such a macroblock should assign a certain value to transparent pixels. Padding is an algorithm that interpolates transparent pixels from opaque pixels inside the VO. Further details of the repetitive padding will be described in Section 4.4. In contrast with the repetitive padding of pixels, which functions at the macroblock level, the extended and boundary padding and postprocessing filters are operating on the whole VOP. This means that the full decoding process involves functions that operate both at the macroblock and VOP level. These intrinsic differences in processing granularity introduce the following limitations:

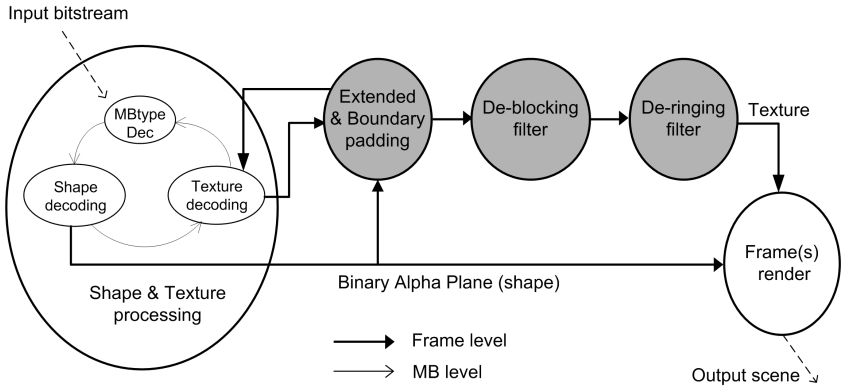


Figure 4.3: *Computation graph of the AS VO MPEG-4 decoder with padding and postprocessing video functions.*

- large buffers at the full-VOP resolution for VOP-based video functions,
- additional VOP delays between each VOP-based task, and
- unbalanced use of the communication resources.

Let us now propose an alternative for the decoding that exploits further parallelism in the decoding process. To this end, we propose a dataflow graph for a parallelism-enhanced implementation that is still compatible with the straightforward implementation of an MPEG-4 decoder.

1. We split the original *repetitive padding* into two tasks: pre-padding and texture padding, because the function is in the critical path and shape information needed for processing is available at earlier stages of processing.

2. We propose to increase the parallelism for the *extended padding* by facilitating synchronized processing, thereby enabling macroblock-based pipelined processing.

3. We introduce data-level parallelism to process the individual chrominance VOPs in parallel with the luminance VOP. This type of parallel data processing holds for the following tasks: Inverse Quantization (IQ), Inverse Discrete Cosine Transformation (IDCT), Texture Motion Compensation (Texture MC), Repetitive Padding, Extended Padding, and postprocessing filters.

The above aspects are discussed in detail in the following sections, where also the increase of parallelism is explored and experimentally validated.

4.4 Repetitive Padding

4.4.1 Task splitting of repetitive padding

The compliant MPEG-4 standard bitstream containing coded AS VOs has the following structure. For each time instant of the video object, the stream has encoded header information followed by shape and texture information of successive 16×16 pixel macroblocks. Due to the fact that the bitstream does not contain any markers for fast allocation of shape and texture information, the processing has to sequentially parse the original bitstream. However, we identified a possibility to increase parallelism by splitting the Repetitive-Padding task. This task should define the values of the transparent pixels for boundary macroblocks [16].

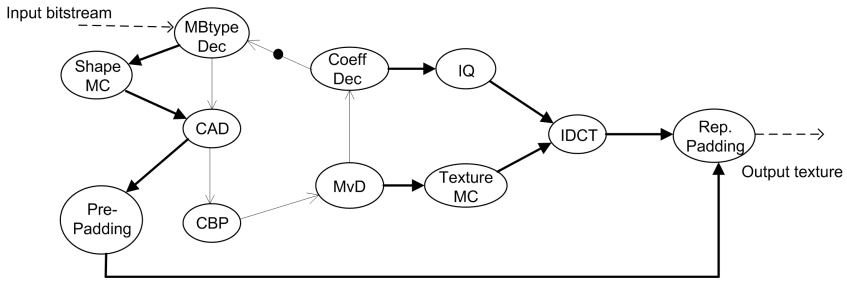


Figure 4.4: *Modified version of shape-texture decoding with the explicit splitting of repetitive padding into two tasks (Pre-Padding, Rep. Padding).*

After detailed analysis of the computation flow, we have found that the shape data is the most important element for the repetitive padding task. This hinted us to start processing of shape data immediately after the CAD task that decodes shape information. This results in a modified computation graph, in which the original repetitive padding is split into two tasks. The first subtask, which is in the computation graph denoted by *Pre-Padding*, identifies for each pixel if its value has to be taken from the original position, or whether its value should be copied from the border of the video object. In case it is copied from one or two borders, we assign a pointer to the special buffer for padded pixel values. The padded values are computed after the texture decoding provides the texture data. This task is executed in parallel with the texture processing. The functionality of the complementary task is to provide the original functionality of *Rep. Padding* by filling the above-mentioned pixel buffer with the real texture values and just copying the data to the output buffer of the repetitive-padding task.

4.4.2 Evaluation of modified repetitive padding

Figure 4.5 compares the original computation requirements with the algorithm modified for task splitting. The figure shows only a part of a complete sequence simulation, but the reduction in computing cycles is well visible. The minimum and maximum reduction was measured for that complete simulation. It was found that the depending on the data contents of a macroblock, the obtained reduction in computation requirements is between 11.6% and 68.5% of the original algorithm for repetitive padding. With the "Dancer" test sequence we achieved on the average a substantial 58.1% savings in computation effort as compared to the original algorithm for the luminance component. The savings even further improve when applying the data-level parallelism described in Section 4.6. The bold line depicts the contribution of the original sequential

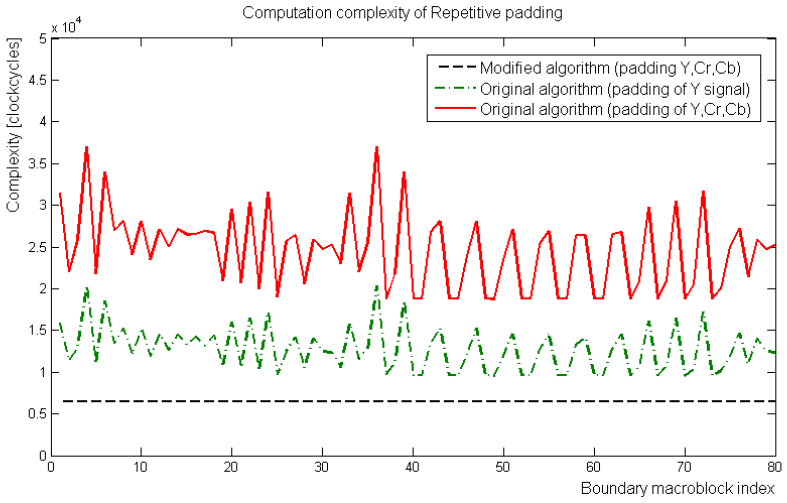


Figure 4.5: *Example of time interval showing the computation complexity of the original repetitive padding and the modified version (for the "Dancer" sequence).*

implementation for all signal components and the punctuated line shows the original padding for the luminance component. The modified implementation for one component (taking into account the advantage of data-parallelism) is visualized by a dotted line at the bottom of the figure. The optimized reduction in computations varies between 64.1% and 82.7% with an average of 71.6%. Further, it can be noticed that the remaining task after splitting (denoted as Rep. padding) has now constant computational requirements. This occurs because the computational variation in the data-dependent component has moved to the Pre-padding task. The remaining part involves only copying of texture data from buffered positions calculated earlier from BAB data.

4.5 Block-level pipelining and synchronization for extended padding

4.5.1 Optimization of communication granularity

The MPEG-4 standard defines the extended padding functions after the complete VOP was processed by previous tasks from the computation graph (see Figure 4.3). We modify this algorithm as follows. Instead of processing VOP by VOP, the tasks can also be carried out on slices of macroblocks, thereby enabling smaller granularity of processing.

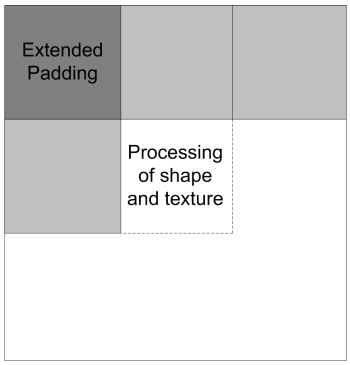


Figure 4.6: *Data dependencies for block-level extended padding. The extended padding can process a macroblock only when the next macroblocks at the right and below are fully decoded and padded.*

By introducing synchronization tokens and a corresponding modification of the processing tasks, the granularity of these tasks can be reduced from the VOP level to the macroblock level. To provide a macroblock-level task pipelining, we propose extra synchronization between the macroblock-type decoding task and the extended-padding task. The extended-padding task is idle until it receives the synchronization token from the macroblock-type decoding task. Furthermore, the extended padding can start padding of the macroblock only when the complete slice of macroblocks and the macroblock below the extended-padded macroblock is fully decoded and repetitively padded (see the gray blocks following the Extended-Padding block in Figure 4.6).

With respect to the latency involved by changing the granularity of extended-padding processing, the following can be stated. The contribution to the critical path of the whole decoding process is lowered to only memorizing the upcoming row of macroblocks for extended padding (see Figure 4.6), as compared to the processing of the whole VOP in the straightforward implementa-

tion. This is explained by referring to the original extended padding algorithm, which is visualized in Figure 2.18 and explained below that figure. The applied priority assignment starts at the block below the padded block which asks for the upcoming row of macroblocks.

4.5.2 Evaluation of the modified extended padding

Figure 4.7 portrays the experimental results comparing the original extended padding algorithm with our proposed modified algorithm running on slices. We have evaluated both algorithms by executing them on an clock-cycle-true ARM7TDMI processor simulator. The compiled code is identical to the one used in the final mapping. Let us discuss the results of the "Singer" test sequence in more detail.

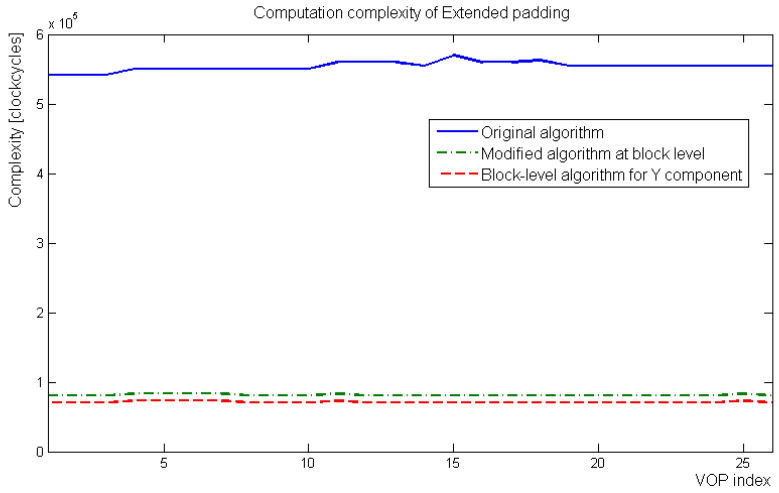


Figure 4.7: *Experimental results of modified extended padding on the "Singer" test sequence*

The contribution of the original implementation of extended padding to the overall latency was on the average 537.9 clock kilocycles per VOP containing 60 macroblocks. After the modification of the algorithm, the resulting contribution was on the average only 75.3 clock kilocycles and by introducing data-level parallelism and performing the individual color-component processing in parallel, it further decreased to 65.8 clock kilocycles. This represents only 12.2% of the contribution of the original algorithm to the overall latency. These results prove the attractive potential of our approach for modifications aiming at enhanced parallelism.

In general, the latency reduction depends on the fraction that the last slice occupies with respect to the vertical VOP height in slices. A small VOP width gives a small slice length and if the VOP height is large, the VOP latency drops significantly. The above-mentioned slice length fraction is varying between 5% of the VOP size ("Singer" sequence) and 50% of the VOP size ("Fish" sequence). A sequence can have the aforementioned fluctuation even within the same video material. However, for most of the processed sequences the remaining fraction is below 25%.

Additionally, the effect of changing the communication granularity from the original VOP-size towards the macroblock size also decreases the requirements on the internal buffer sizes. The straightforward implementation assumes that data are shared between processing tasks. Therefore, a straightforward implementation of Extended padding on a multiprocessor architecture without shared memory requires an internal buffer for the whole VOP. The parallel implementation requires the processed data to be fully stored in the internal memory of individual processors. In our new approach, the required buffer size is minimized to just one slice of macroblocks plus one macroblock to perform the MPEG-4 compliant extended padding algorithm.

4.6 Data-level parallelism within the full decoder

To further increase the throughput of the decoder, we also employ data-level parallelism. For each color component, we instantiate a separate pipeline of tasks, so that the execution of the Inverse Quantization, IDCT, etc., can start as soon as the decoding of coefficients for the luminance plane has finished. The subsequent processing of Cr and Cb chrominance components runs in parallel with the further processing of the luminance (Y) plane. Due to the smaller size of the chrominance planes, the splitting of chrominance parts to individual component processing is not decreasing the length of the critical path of the computation graph (no latency decrease), but it may be useful for better utilization of computation resources.

The critical path, depicted with bold arrows in the graph of Figure 4.8, contains the tasks for the original bitstream parsing for the decoding of shape information and the decoding of Y texture data. The decoding of texture information is most complex for the luminance component, due to the fact that the AS VO MPEG-4 standard currently supports only the 4:2:0 sampling. Figure 4.8 portrays the complete graph that contains also the proposed modifications as discussed in Section 4.4 and Section 4.5. The critical path through the computation graph that effects the latency of the complete decoding has the following two parts:

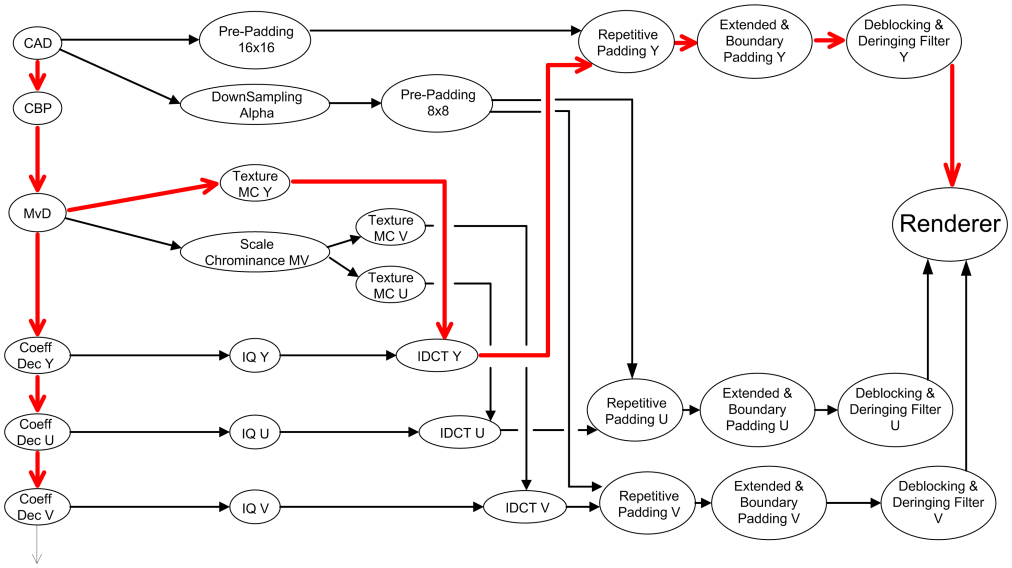


Figure 4.8: *AS-VO compliant MPEG-4 decoding computation graph employing task-level and data-level parallelism. Bold arrows indicates a critical path. The vertical critical path at the left originates from the repetitive decoding of each macroblock.*

- *Bitstream parsing:* Context Arithmetic Decoding (CAD), Coded Block Pattern (CBP), Motion Vector decoding (MvD), DCT coefficients decoding (Coeff Dec Y);

- *Texture processing for the luminance component:* Inverse Quantization (IQ), Inverse Discrete Cosine Transformation (IDCT), Repetitive Padding Y, Extended and Boundary Padding Y, Deblocking and Deringing Filters of the Y component.

The complexity of the Pre-Padding is significantly lower than Texture Motion Compensation (Texture MC Y) and similarly, the tasks Coef Dec Y and IQ Y are together less expensive than Texture MC Y (see the complexity discussion in Section 3.6). This explains why these tasks are not included in the critical path. It should be noted that the indicated bold path remains critical, only under the following condition: the tasks for different color components are executed on the same type of processing cores and at the same clock frequency. In other words, if one of the processor cores operates at a lower frequency, then the critical path may change to functions that are executed on that processor.

4.7 Sprite decoding on CELL processor

The remaining part of this chapter is devoted to adding and exploring a complementary part of the AS VO MPEG-4 decoding algorithm. This addition completes the decoder to provide full object-based video scenes containing both objects and background information. Prior to exploring the algorithm and its optimal mapping, we present an alternative platform for execution. This processor was adopted for sprite coding experiments, because it was claimed that this platform would be capable of executing AS VO MPEG-4 decoding and it was commercially available. With a bird's eye view, the CELL processor

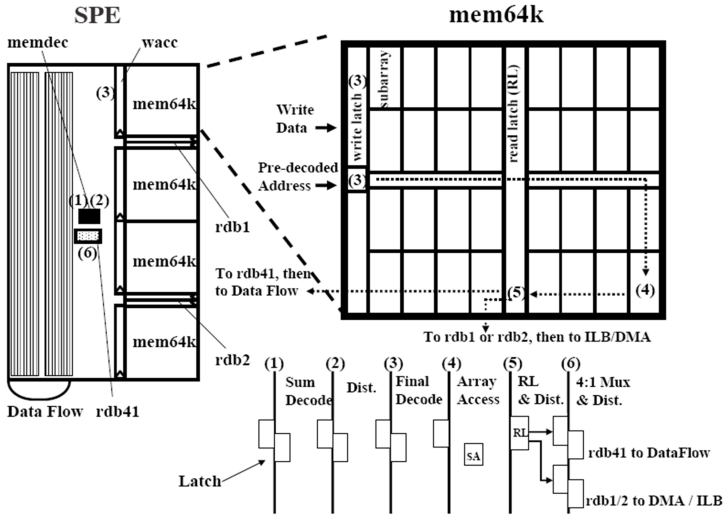


Figure 4.9: *Local memory structure in an SPE of the CELL processor (taken from [32]).*

has a similar tile-based architecture as the experimental platform explored elsewhere in this thesis. Our initial analysis has revealed that the CELL processor system [38] meets computational requirements of the block-based AS VO MPEG-4 decoding. However, the limited size of the local memory requires a modification of the MPEG-4 sprite-decoding algorithm. The implementation of a first-generation CELL processor consists of a 64-bit Power Processor Element (PPE) with its L2 cache and multiple *Synergistic Processor Elements* (SPEs). Each SPE has its own local memory [32]. Synergistic aspects in processing means that individual processing tiles are directly connected and perform pipelined execution of distributed algorithms. As shown in Figure 4.9, an SPE unit contains four 64 kByte memory blocks.

A major cost problem of the MPEG-4 decoding algorithm is the buffering resulting from accumulating backgrounds views of the decoded scene background. The accumulated background image, called sprite, requires a significant memory for construction. The visual Main Profile Level 2 (MP@L2) of the MPEG-4 standard bounds the maximum size of a reference image for the sprite reconstruction to 1584 MacroBlocks (MBs) at CIF resolution, which involves about 608 kByte (a single CIF video picture contains 396 MBs) [36]. The target processing-tile memory limitation of 256 kByte can handle decoding of rectangular video pictures or arbitrary-shaped video object decoding, but it cannot internally buffer the complete reference sprite image. For this reason, we present a new algorithm that decodes MPEG-4 compliant sprite-background sequences, while satisfying the target platform constraints on memory. Furthermore, it also exploits data-level parallelism. As a further benefit, it will enable the mapping of the full decoder on the memory-constrained platform. In the next section, we will present the modified sprite coding algorithm.

4.8 Background Sprite Decoding

4.8.1 Original MPEG-4 algorithm

A complete object-oriented video sequence requires the reconstruction of the background on top of which individual video objects are superimposed. The principles of the background sprite reconstruction have been presented in Section 2.5 of this thesis. The affine transformation of the sprite is modeled as a mapping between the decoded sprite plane and a current view plane. In the standard, the affine transformation is regularly called warping. This transformation is described by the following formulas:

$$x' = \frac{m_{00} \cdot x + m_{01} \cdot y + m_{02}}{m_{20} \cdot x + m_{21} \cdot y + 1}, \quad y' = \frac{m_{10} \cdot x + m_{11} \cdot y + m_{12}}{m_{20} \cdot x + m_{21} \cdot y + 1}. \tag{4.1}$$

In the MPEG-4 standard, the sprite decoding consists of four steps: shape/texture decoding, buffering of the complete sprite, decoding of the warping vector and geometrical warping (Figure 4.10). The received coded I-VOP (Intra-coded Video Object Plane) contains coded data for shape and texture of the reference sprite. The received coded S-VOP (Sprite Video Object Plane) contains coded warping vectors.

The Shape/Texture Decoding task in Figure 4.10 decompresses the coded sprite data and stores the luminance, chrominance and grayscale alpha data of a sprite in two-dimensional arrays. The width and height of the luminance array are specified by the syntax parameters `sprite_width` and `sprite_height`, respectively. The resolution of the sprite-reference image is usually several times larger than the video-scene resolution, so that most of the complete

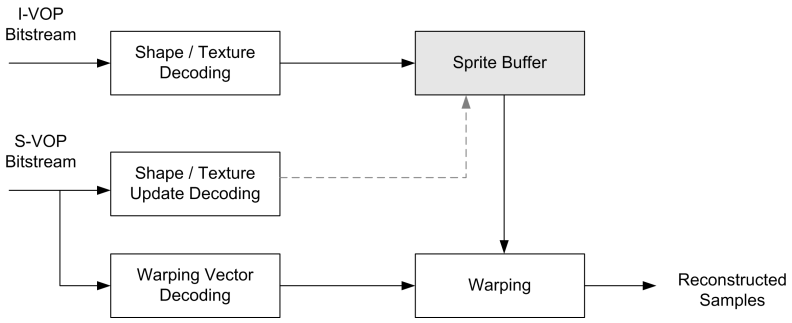


Figure 4.10: *Original MPEG-4 decoding of background sprite.*

background of a short sequence is captured. The chrominance and luminance planes are stored in the so-called Sprite Buffer and are used as references for the actual background reconstruction corresponding to the current camera view. The actual view-reconstruction process consists of decoding the warping vector from the coded data and applying the previously defined warping process onto the reference image in the Sprite Buffer.

4.8.2 Modified sprite-reconstruction algorithm

The implementation of the above-described algorithm has the inherent problem of a large reference sprite image that cannot be contained by the local memories of the SPE elements. The algorithm requires about 2.4 times more storage than the 256 kByte available at individual SPE cores. For this reason, we have designed a new algorithm that remains MPEG-4 compliant and additionally allows a higher degree of both data- and task-level parallelism.

The new algorithm exploits the partitioning and optimal buffering of sprite data, because it intrinsically involves a high amount of MBs. At this point, the reader may argue that a solution would exist in storing the sprite in a large off-tile memory. However, this solution will inevitably lead to a large latency, as the access to off-tile memory locations requires a considerable access time. The extra latency will decrease the real-time behavior of the sprite-decoding task.

The primary difference is in the way how MB data are stored. The original approach keeps the whole reference image in an uncompressed form. Instead, we propose to keep the reference sprite image in compressed form and decode only the part that is required for the reconstruction. As a consequence, the new decoding process is decomposed into four steps: (1) the construction of an information matrix for random access to MB data, (2) decoding of warping

vectors, (3) decoding of the required MB texture data, and (4) warping of the actual sprite.

Figure 4.11 portrays the block scheme of the new approach. Let us simplify the sprite decoding to the situation when the bitstream contains only one fully encoded sprite image and it is followed by a number of S-VOPs without updates of the texture/shape information. In a longer sequence with multiple sprites, this cycle is repeated when a new sprite image (I-VOP) is received. The decoding of the actual background involves the following steps:

1. Parsing of the I-VOP bitstream and construction of the *Random Access Data Matrix* (details are given in the next section).
2. Decoding of the warping vectors from an S-VOP.
3. Calculating the bounding box that defines the actually referenced sprite view.
4. Fetching and decoding of MB data that was not available in the previous referenced image for the actual bounding box.
5. Recalculating the image origin.
6. Warping the current view of the sprite image.

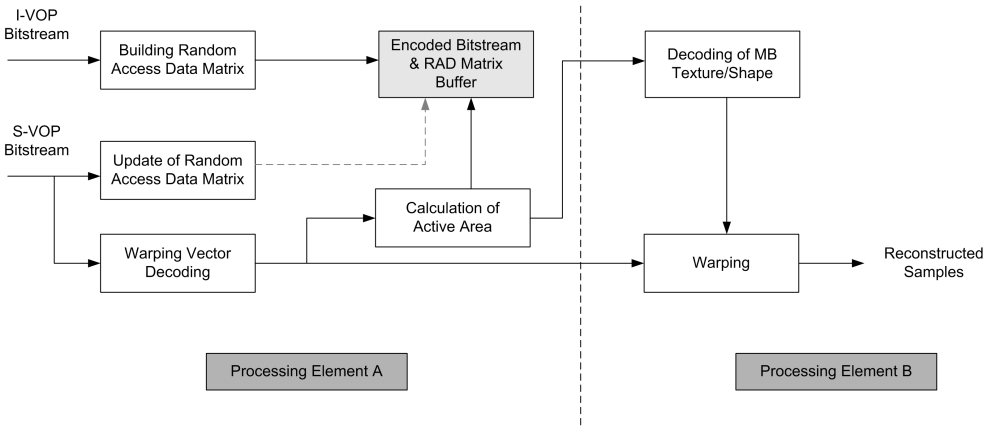


Figure 4.11: *Decoding diagram of the modified background-sprite reconstruction with a proposal for task-to-processor assignment.*

In Figure 4.11, the vertical dotted line indicates that the algorithm has to be split into at least two parts and mapped onto two processing elements. This split satisfies the SPE constraints on the memory size (256 kByte). More analysis and experimental evidence on the modification are given in Section 4.10.

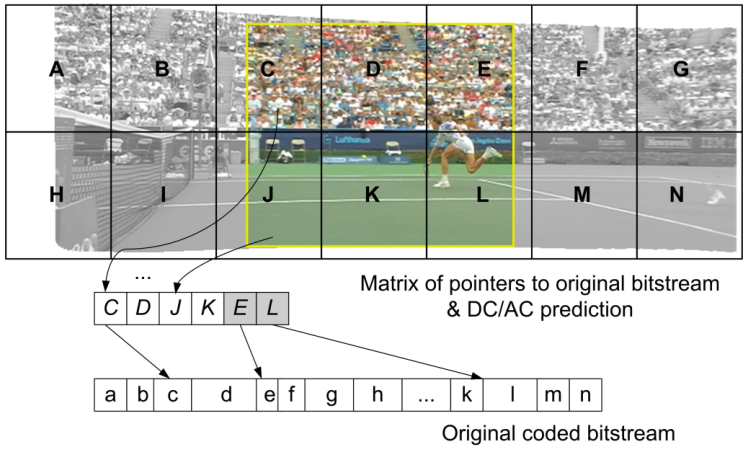


Figure 4.12: *Data organization for the modified sprite decoding algorithm, where the arrows indicate pointers to the required data.*

4.9 Construction of MB data Matrix for Random Access

The MPEG-4 compressed bitstream does not contain markers for accessing image data at a macroblock granularity level. At the first stage, we construct the MB *access matrix* for access to the bitstream of MB-compressed data. The texture processing is performed in three steps: DC/AC prediction based on the previous neighboring blocks, decoding of DCT coefficients and IDCT. To provide random access to the MB data, two approaches are available. First, the processing is organized such that data is buffered in matrices of 8×8 DCT coefficients. Second, an alternative is to buffer macroblock DC/AC predictors and postpone the decoding of DCT coefficients until the moment that the MB is required for the warping process. Since in the first approach the target matrix for storing DCT coefficients has the same size as the complete sprite, we have adopted the second approach in which MB data are kept to save memory.

Figure 4.12 exhibits the data organization for the modified sprite decoding algorithm. Note that memory blocks E, L containing pointers are gray, because these blocks require the decompression of encoded MB data (e, l in the figure). The calculation of the total active area (Step 5 in the algorithmic description) leads to the processing requirements for the e, l MBs in the further warping process. The algorithm inspects the matrix of pointers to identify the positions in the original coded bitstream buffer.

A major gain in the use of pointers for the decoding process is achieved during the reconstruction of the first frame (I-VOP). The MPEG-4 standard algorithm performs DC/AC prediction, DCT coefficients decoding and IDCT on the complete reference image. Instead, we perform first two simple processing stages (without buffering of DCT coefficients) on the whole image and only the last step performs actual full decoding but on a restricted image size. This last step introduces an extra overhead caused by the redundant decoding of the sprite area that remains identical between two consecutive sprite VOPs. However, this overhead can be removed if we *a priori* obtain the warping vectors for the first image-to-texture parsing and calculate the new active area in the second image. When using this modification, the number of MBs for the second image IDCT transformation is reduced by an impressive 76%. Additionally, we can speedup the texture processing by instantiating subsequent DCT coefficient decoding and IDCT transformation with parallel MB decoding.

4.10 Experiments and results of modified sprite decoding algorithm

We have implemented the modified version of the MPEG-4 sprite decoder. Figure 4.13 shows the results of decoding the well-known MPEG "Stefan" sequence. The amount of required MBs for decoding processing is illustrated at the left of Figure 4.13 by the (noisy) bold line at the bottom. The decoding of the background reference sprite requested for the first sprite image extracted from the complete reference image (in our case 225 MBs) is shown at the top of the same figure. For this test sequence, the maximum number of uploaded MBs for the decoding of the active sprite area was 41 MBs.

In Figure 4.14, we show the required buffer sizes in terms of MBs of the original and the new algorithm. The original decoding requires 1,701 MBs stored in an uncompressed way (1,584 is the maximum, but multiple 16-pixel MB grid alignments increases this number to 1,701). The maximum size of the reference sprite never exceeded 400 MBs. It was found that the original algorithm requires 4.25 times more memory than our modified algorithm. The figure also shows that our algorithm requires small bursts of blocks for decoding during the scene, but these bursts do not accumulate to a significant number, so that the total remains much smaller than the big burst required for the original algorithm. In the experiment, the amount of memory to store the original bitstream was 37.44 kByte and the resulting random-access data matrix needed 7 kByte of memory (fixed small memory space).

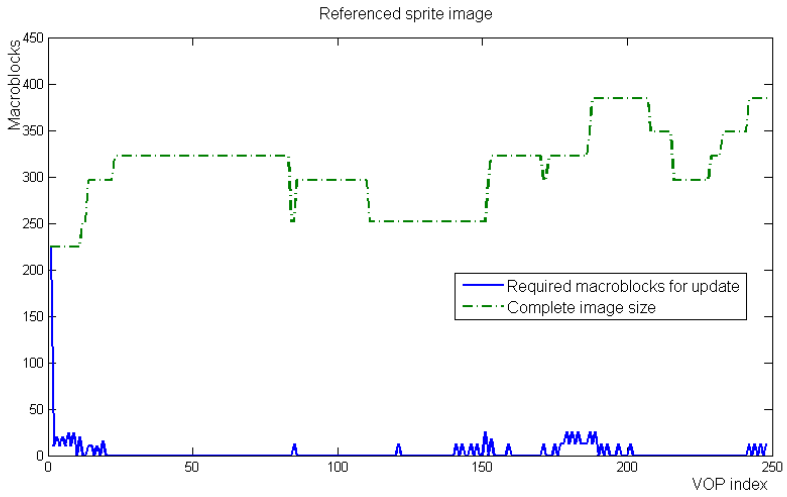


Figure 4.13: *Processing of "Stefan" sequence by the modified algorithm.*

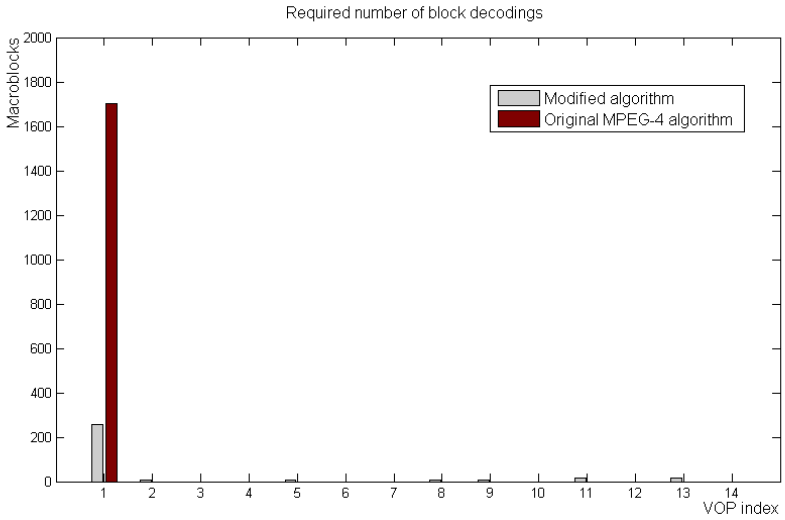


Figure 4.14: *Comparison of the required number of macroblocks for sprite decoding using the original and the modified algorithm.*

However, compared to the original algorithm, we found that our proposal has the disadvantage of occasional extra decoding of the same MBs in the case of special background movements. Such a movement occurs when the camera moves away and then back to the view that was used in a previous sprite decoding. When the camera returns to the old view, the same background has

to be decoded again. This disadvantage depends on the camera motion in the background. During our experiment with the "Stefan" sequence, we observed that the repetitive decoding of the same sprite involved 17% of extra MBs decoding iterations in terms of computation.

4.11 Conclusions

We have shown that a parallelism-enhanced implementation of AS VO MPEG-4 decoding on a multiprocessor platform offers significantly higher throughput due to the large reduction of the critical path, as compared to a conventional implementation. The chapter has discussed techniques to increase the task-level and data-level parallelism in the MPEG-4 decoder. The task-level parallelism is in general achieved by splitting a task on the critical path. The data-level parallelism addresses parallel processing of individual signal components. We have evaluated our algorithms experimentally on a clock-cycle-true simulator of a multiprocessor architecture using ARM7TDMI processors.

Repetitive padding in MPEG-4 decoding is the primary candidate for introducing task-level parallelism, because shape data are available at earlier stages of processing compared to texture data, so that processing can be already initiated. We have proposed to split this task into two tasks: the filtering of the shape data and the copying of pixel values. Only the second task remains at the original place in the critical path. This decreases the original computational complexity of the repetitive padding to 40.9% of the original task. By applying additional parallel color-component processing it drops to only 28.4% on the average.

The modification of the Extended Padding algorithm has two major impacts. The first is in the reduction of the overall decoding process latency. The standard describes the extended padding as postprocessing after the whole VOP is fully decoded. By changing the granularity to block level and introducing a new synchronization mechanism that is aware of having sufficient data for processing, we obtained an execution having only 12.2% of the original algorithm execution cycles. Additionally, the task-specific buffering is maximally one slice of macroblocks of the image resolution plus one macroblock. For example, at CIF resolution, this involves only 5.8% of the original internal buffer.

The value of the first part of this chapter is in the elements and strategy for systematically exploring parallelism. We have provided a strategy to exploit the parallelism that is generally available in multimedia coding algorithms. In the study, the identification of the critical path was carried out manually. We envision that such a manual optimization can be conducted by automated

tools for application analysis in the future. In order to realize this, the detailed knowledge of data availability in the data-flow graph is indispensable. The generalization to increase parallelism can be summarized as follows. First, all data inputs of a task on the critical path are evaluated for being available for processing. If some inputs are available at an earlier stage, then we split the task that operates on these data and exclude it from the critical path. Second, we unify the computation granularity. Third, explore the parallelism of individual component processing.

In the second part of this chapter, the redesign of the background sprite algorithm was addressed in order to map it onto a CELL processor. The modified algorithm features minimum use of local memory in the processing elements of the target processor network. A second feature is that it is based on constructing a new special information matrix to support random access to MB-coded data, which enables independent MB processing. This potentially allows decoding at more processing elements, thereby increasing the data-level parallelism. It was shown that the required memory reduces with a factor of about four, with only 17% computation overhead due to repetitive MB decoding for the complete sequence. It should be noted that this concept using a special mapping matrix can be reused for other applications as well. For example, any kind of processing on a variable-length coded data stream can be split up with the same principle, thereby facilitating enhanced parallelism.

When taking a broader view on the results of this chapter, we can remark that the presented techniques on extracting the parallelism suit any application domain and the parallelization techniques like task splitting and memory optimization, are not unique for our problem statement. However, what is special in our work is that we have mapped advanced multimedia algorithms onto a multiprocessor network. When analyzing why the task splitting and memory organization results were so successful, we have come to the conclusion that this is because we have redistributed the computational task over the network and at same time equalized memory and computational load. The result is an application mapping that fits much better to the multiprocessor network.

Chapter 5

Hierarchical Quality-of Service approach

This chapter structures the problem of QoS management for a tile-based multiprocessor platform, such that the individual application control is abstracted from the overall system control. This leads to a concept of two management layers that are communicating with each other and finding a balance in the negotiation on resources. For the proper QoS management for AS VO MPEG-4 implementation, we introduce task-level scalability properties into the algorithm. The chapter concludes with an optimization algorithm assigning a quality level to a set of parallel executed scalable applications. This optimization algorithm was experimentally validated with a setup of four applications in parallel, where it was shown that the quality can be actively controlled to the benefit of individual applications.

5.1 Introduction

The target multiprocessor NoC platform, which is executing a number of multimedia applications in parallel, requires an overall system control to ensure stable performance with the correct quality settings under various platform conditions. Unfortunately, this overall control task cannot be carried out by a conventional operating system, because it lacks the knowledge about the desired overall system usage and it has no notion about the meaning of the quality of the individual applications and the quality of the complete system.

It is evident that a special controller is needed to safeguard the quality of applications under various circumstances. The controller will be special as multiple applications will be executed in parallel.

A second aspect of the previous problem statement is the embedded form of the target system. For embedded applications, there will be system constraints on the available computing power and other resources. If the computational load becomes too high for the platform, we would like to reduce the effort involved for particular tasks without the complete abortion of the job execution. In order to do this in a quality-controlled way, we need two elements.

- The multimedia applications should have *scalable properties* in terms of performance and computational effort.
- The platform should be able to *control and monitor* a number of parallel applications and their resource usage.

The conclusion of this discussion is to implement a special Quality-of-Service (QoS) system that is capable to handle the control aspects dealing with the resource usage of multiple applications.

Current status of many system realizations is far from the above system definition. The mostly used mappings are static and based on dedicated hardware, so that they obstruct the re-usage of system resources. This aspect does not match with an application characterized by a varying number of objects and their sizes such as in AS VO MPEG-4 decoding. The consequence of such an object-oriented coding application is a variable resource-usage requirement during processing, thereby asking for a platform that supports this feature. This has motivated our research on designing a Quality-of-Service manager for the target multiprocessor system. QoS control has been subject of research already for a number of years. Therefore, we concentrate on the part that is usually missing in this type of research. In our case, we discuss multiple objects and tasks running in parallel on a multiprocessor platform. This covers the case of a single advanced application or a set of MPEG-4 decoding applications executed in parallel. This problem statement is a new element in this thesis and distinguishes itself from previous work on QoS management. Therefore, we focus particularly on the video-application part of the overall resource management.

In order to make the application suitable for QoS control, we first introduce a form of scalability into the existing AS VO MPEG-4 decoding algorithm. This form of scalability distinguishes several scalable tasks and tasks that can be

optionally omitted. Thus, scalability is achieved by task switching. This algorithm will be used in further experiments within this chapter. The author is aware that the proposed approach represents only a limited form of scalability. However, the design of a fully complexity-scalable decoding algorithm is a task of its own and beyond the scope of this thesis. The purpose of this chapter is to come to a new concept for multiprocessor QoS control.

Let us now briefly outline the QoS management that is proposed in this chapter. In order to allow single and multiple video applications in parallel to be controlled with the same QoS concept, we introduce a *hierarchical* QoS architecture, where a Local QoS controls an individual application while a Global QoS controls the complete set of active applications and optimizes the system behavior. For this purpose, we define a global cost function. This global function controls the set of applications and determines the overall system behavior. It should balance the offered quality of individual applications and their corresponding resource usage, while maintaining overall system performance. The model of selecting quality levels and assigning resources is supported by a heuristic optimization algorithm that will be executed at runtime.

For a single application, the Local QoS controls the quality. As video objects can change over time in size, shape and texture content, the processing requirements are more variable than with frame-based video processing. Our Local QoS mechanism relies on the earlier results of this thesis, i.e. the execution times of the timing and bandwidth models from Chapter 3 are actualized with the results of the previous decoding iteration. Some experimental evidence will be provided in Section 5.3. This type of resource-usage modeling is taking the varying requirements on computation into account, e.g. size of video objects. In general, the Local QoS control can use any suitable resource-usage prediction and is not bounded to our parametrical models. An alternative for resource-usage prediction is to apply a statistical model of the execution, such as used in e.g. [80].

The remainder of this chapter is structured as follows. Section 5.2 discusses the importance of tasks for the AS video-object reconstruction and defines the quality levels of our target application. Section 5.4 explains our hierarchical approach for QoS and presents a model of runtime resource management. Section 5.4.4 defines the *optimization algorithm* to maximize the overall cost function. This model was experimentally tested and results are discussed in Section 5.5. Section 5.6 summarizes the results of this chapter.

5.2 Development of scalability of AS VO MPEG-4 decoder

5.2.1 Scalability overview and introduction of concept

QoS management for Systems-on-Chip (SoCs) has been extensively studied for e.g. MPEG-4 3D graphics [15], wavelet coding [77], etc. The proposed QoS management approach computes the resource utilization as an algebraic function of the quality settings, based on e.g. the number of graphical triangles to be processed, or it explores the temporal scalability of video processing [23]. The following approaches for implementing scalability at receiving terminals have been published.

- Spatial scalability - the sequences are encoded at different spatial resolution per quality level [33, 40, 122].
- Temporal scalability - the base quality presents frames at lower frame rate and missing frames are transmitted in enhanced quality levels [70, 119].
- SNR scalability - different quantization steps for different quality levels are used [104, 121].
- Complexity scalability - the quality levels are scaled with the available computational resources and different types of coding algorithms are used per quality level. [74, 111].
- Object scalability - the scene is composed of a set of video objects from which less important objects can be skipped and the quality of the picture is evaluated compared to the scene with all video objects [100].
- Fine granularity scalability (FGS) - this is scalable coding of a video sequence for communication through channels with a wide range of bitrates, such as the Internet [73, 112].

From the listed approaches, the most attractive approach for solving our problem statement is using complexity scalability, but the design of a fully complexity-scalable algorithm is beyond the scope of this thesis.[1] An alternative choice can be object scalability, but degrading the scene to a lower number of video objects can completely change the semantic meaning of the scene. For this reason we do not apply this form of scalability. Instead, we have defined a new type of *task scalability* based on enabling / disabling tasks that compose a job. Based on the importance of the task processing to the overall decoding process, we distinguish *essential* tasks and *non-essential* tasks.

[1] All individual tasks have to be redesigned for complexity scalability and integrated into an overall scalable concept of the application.

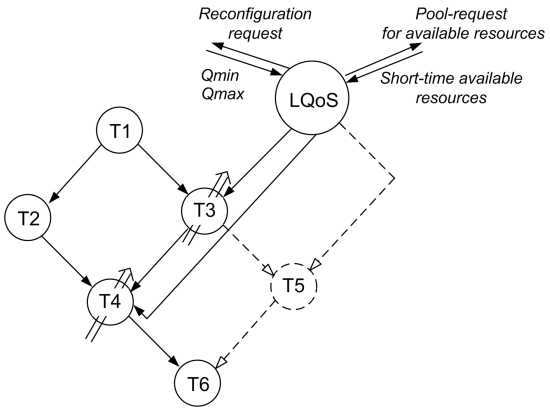


Figure 5.1: *Example of a job with complexity-scalable tasks [74]. Scalability is indicated by diagonal arrows, and the dotted arrows and dotted tasks that even can be skipped.*

Each job, e.g. an AS VO MPEG-4 decoder, is divided into communicating tasks. For each job and related quality level, we provide a detailed task graph, as portrayed by the general example in Figure 5.1. For introducing QoS at the task level, it is important to identify scalability options for each task[2] (see Figure 5.1). If a job contains tasks that may be completely idle, and in consequence, the corresponding communication resources are idle as well, they are denoted by dotted lines and circles (Task T5 in Figure 5.1). In the sequel of this chapter, we experiment only with the task-skipping scalability.

5.2.2 Task-level scalability of the AS VO MPEG-4 decoder

For task skipping, we classify the contribution of a task to the overall decoding process into *essential* tasks and *non-essential* tasks. The decoding of shape data and texture reconstruction for an object are considered essential tasks. The second class consists of tasks that enhance the output quality: the deblocking and deringing filters, and tasks supporting the correct reconstruction of the border of an object, like extended and repetitive padding. Note that the MPEG-4 standard does not allow skipping of the padding processing. However, we have found that when padding tasks would be skipped for scalability, compared to leaving out the complete object (the object scalability [100]), the artifacts on borders of video objects seem to be less quality degrading in the overall perception of the final scene. Therefore, we classify extended padding as a non-essential task.

[2]Scalability can be further enhanced by also exploring the different data edges between tasks based on a selected quality level. This is not further explored.

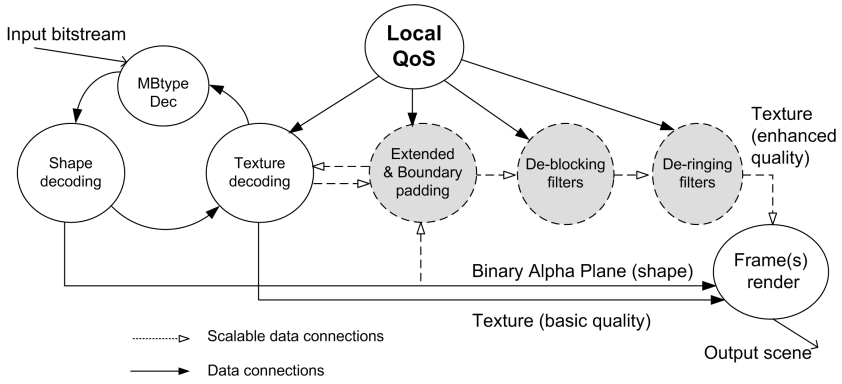


Figure 5.2: *The data-flow graph (DFG) of the AS VO MPEG-4 decoder with indicated task-level scalability.*

Figure 5.2 visualizes a task DFG of the AS VO MPEG-4 decoder. The presented graph also offers optional skipping of some tasks under certain conditions. For example, the extended and boundary padding task is not required if the currently processed VOP is a B-VOP and thus it does not serve as a reference picture (conditional skipping).

The Shape Decoding task is based on context arithmetic decoding. The Texture Decoding task in the diagram involves variable-length decoding, inverse quantization and inverse DCT (see Chapter 2). As classified earlier, both tasks are essential. The second large processing task, Extended and Boundary Padding, combines the shape and the texture data and constructs the texture information for pixels that are transparent in the current VOP. We can skip this task in case of resource shortage. The VOP postprocessing consists of two steps: the deblocking filter and the deringing filter (see details in Chapter 2). Either one or both of them can be omitted to implement task-skipping scalability.

At this point, the reader can debate whether task-skipping scalability is sufficient for our purpose. Let us briefly discuss a few examples of further forms of scalability. Our classification of essential and non-essential tasks is based on the importance of the shape and texture data for the object reconstruction. However, the MPEG-4 standard allows both lossy shape coding and lossy texture coding. Assuming that the shape of an object is more important than the texture, then in extreme cases having several objects, the correct shape information of a few objects can already help in the scene reconstruction without texture in some of the objects. Depending on the shape, some artificial texture can be added, or texture is re-used from previous frames. Similar reasoning

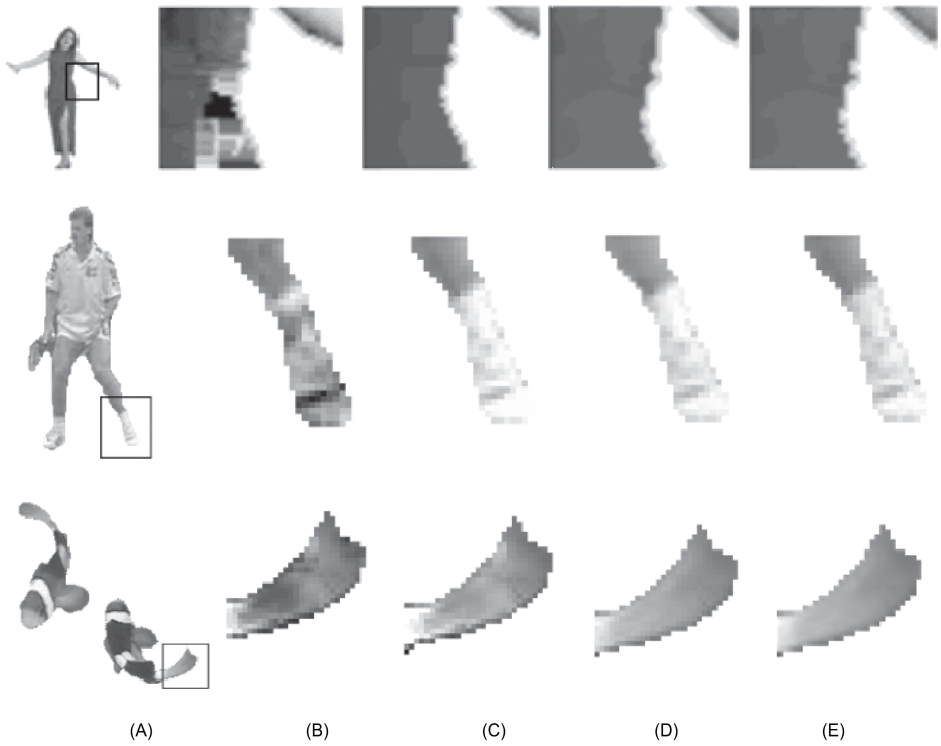


Figure 5.3: *Resulting quality after task skipping for several VO sequences. Images from left to right,(A) original VOP, (B) enlarged view without deblocking, deringing and padding tasks, (C) enlarged view without deblocking and deringing tasks, (D) enlarged view without deringing tasks, (E) fully decoded.*

can be applied when the priorities of shape and texture are exchanged. These simple examples show that scalability can be pursued in unconventional ways to lower the required resources of an application. This can be beneficial for e.g. mobile devices, but this form of scalability is not further explored.

5.2.3 Visual degradation caused by task skipping

The visual result of skipping the padding tasks is portrayed by Figure 5.3. The first column contains the fully decoded video objects. The second, third, and fourth column show enlarged views of object parts under different processing conditions. The lowest quality is seen in the second column when only essential tasks are carried out. The third column adds the padding tasks to the essential decoding tasks. The fourth column shows the quality improvement when deblocking and deringing is enabled.

It can be seen that each addition to the set of essential tasks, gives a no-

table improvement in quality. The improvement step from column (B) to (C) is larger than going from column (C) to (E), where going from (D) to (E) gives the smallest improvement. Table 5.1 shows the distribution of task complexities in percentage of the used computation resources for different video sequences. The addition of post-processing filters requires less resources compared to including padding, which also corresponds with the growth in visual quality.

Sequence	Shape & Text. Decoding	Extended & Bound. Padding	Deblocking Filters	Deringing Filters
Singer	28.26%	6.91%	31.44%	33.39%
Dancer	26.92%	5.07%	31.15%	36.86%
News	23.21%	5.88%	44.99%	25.92%
Fish	27.37%	5.47%	35.61%	31.55%

Table 5.1: *Average distribution of task complexities of VOP decoding.*

5.2.4 Measurement of quality degradation

To quantitatively compare the quality degradation, we have measured the Peak Signal-to-Noise Ratio (PSNR). In literature, the PSNR for arbitrary-shaped video objects was defined for a scene as measuring the difference of PSNRs between the case that a key object is inside the scene or excluded [100]. Since we aim at a perceptive quality measurement, this definition seems not suited for our case. Therefore, we propose to measure the PSNR of each object individually. The PSNR is measured by first obtaining the Mean Squared Error (MSE) of the VOP as follows

$$MSE_{VOP} = \frac{1}{\Lambda}\sum_{i=0}^{\Lambda}(A_i - B_i)^2, \tag{5.1}$$

where Λ is the set of all opaque pixels of the VOP, i.e. the visible part of the VOP, A is the original image and B is the reconstructed image. Subsequently, the PSNR is derived from the MSE_{VOP} by

$$PSNR = 10\ log_{10}\frac{255^2}{MSE_{VOP}}. \tag{5.2}$$

This definition holds for 8-bit video samples used in AS VO MPEG-4 decoding. We have measured the PSNR for the task-level scalability examples as presented in the previous section in Figure 5.3. The numerical measurements are presented in Figures 5.4-5.7. These figures illustrate the PSNRs of the task-level scalability settings for the luminance signal component. For the actual size of video objects at individual time instants, the reader is referred to

Sequence	Shape & Text. Decoding	Extended & Bound. Padding	Deblocking Filters	Deringing Filters
Singer	29.05	48.07	48.75	48.14
Dancer	25.16	34.27	34.35	34.50
News	23.63	37.63	37.89	37.82
Fish	29.60	32.01	32.01	32.13

Table 5.2: *Average PSNRs for various sequences using different task-level scalability settings. PSNRs in a certain column are measured with all functions enabled from the previous left up to and including the functions of the actual column.*

Appendix B.

The obtained values are shown in Table 5.2. It is noticeable that decoding with essential tasks only (shape and texture decoding) leads to a mean PSNR of 23-29 dB. When including padding tasks the mean PSNR increases with 3-19 dB. The postprocessing filters improve the picture quality only by a very small fraction. Note that enabling the deringing filter after deblocking for some sequences ("Singer" and "News" sequences in the table) even results in a slightly lower mean PSNR due to the smoothing nature of this type of postprocessing.

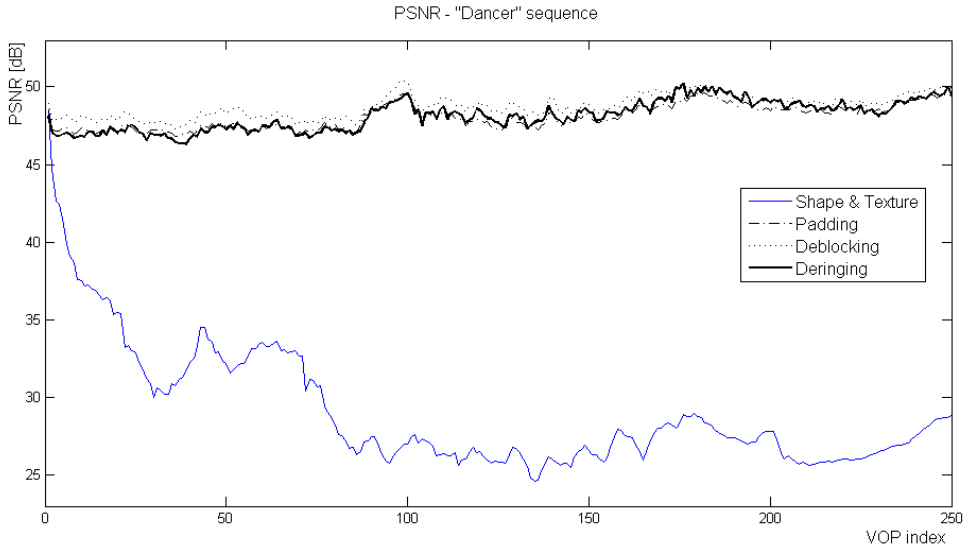


Figure 5.4: *PSNR of the luminance component of the "Dancer" sequence at CIF resolution, 25 frames/s.*

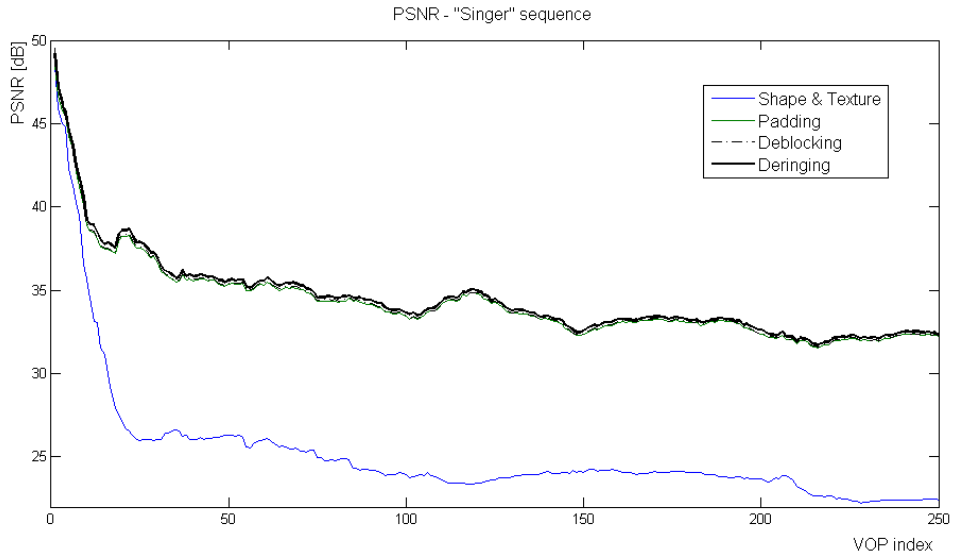


Figure 5.5: *PSNR of the luminance component of the "Singer" sequence at CIF resolution, 25 frames/s.*

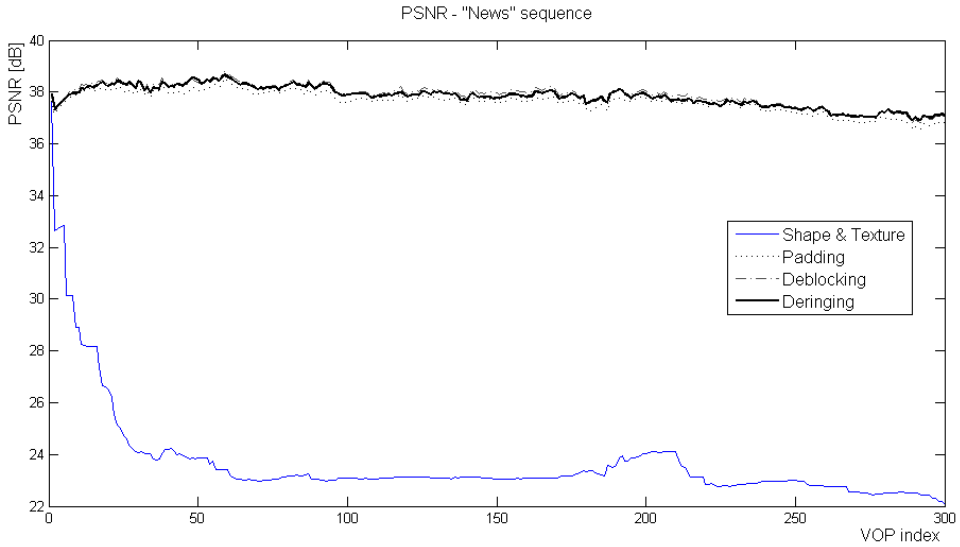


Figure 5.6: *PSNR of the luminance component of the "News" sequence at CIF resolution, 30 frames/s.*

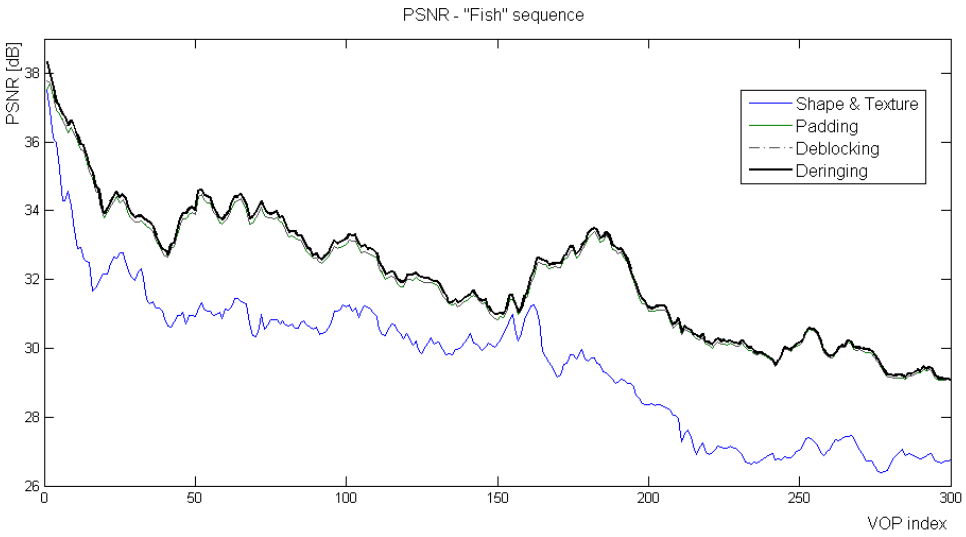


Figure 5.7: *PSNR of the luminance component of the "Fish" sequence at CIF resolution, 30 frames/s.*

5.3 Local QoS

The previously introduced quality changes by enabling task-level scalability are controlled by a Quality-of-Service (QoS) unit. For a single MPEG-4 application, a single QoS unit is sufficient for quality control. For example, this unit decides which tasks are enabled or disabled. Later in this chapter, we introduce the problem of multiple coding applications running in parallel, for which we need a more advanced QoS concept with overall system control. We now first study the simple case based on a single application. The control unit for that application is called Local QoS.

5.3.1 Local QoS concept

This section deals with the QoS control for an individual application. This so-called intra-application control allows to separate the control of individual applications from the control of the total system. At the compile-time design phase, we require analysis of the resource usage for all quality settings of each job to measure the behavior of the application under different circumstances. Using these measurements, the Local QoS can provide appropriate control of the application at runtime.

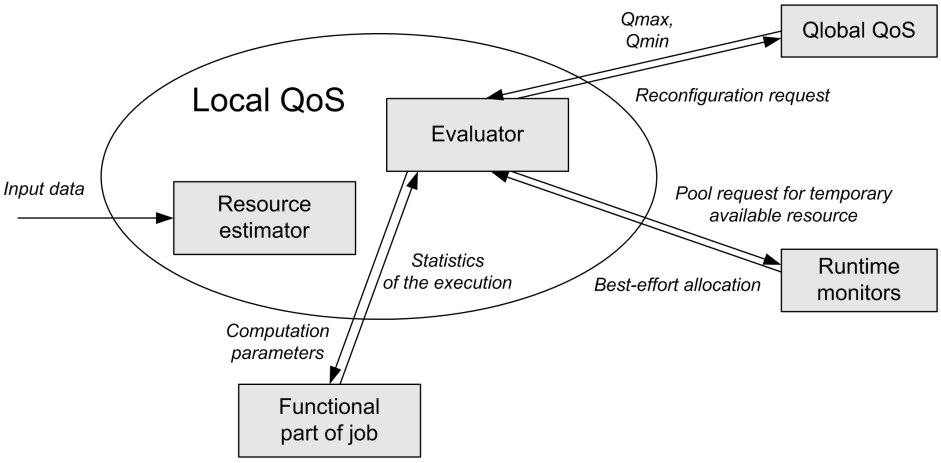


Figure 5.8: *Details of Local QoS components and the interaction with other layers.*

In Figure 5.8, the Local Qos unit contains a *Resource Estimator* that is responsible for runtime estimations of resource demands, depending on the input data and the characteristics of the platform. An *Evaluator* module compares the estimation of the required resources with the actually reserved resources for the job. Based on this evaluation, it schedules the activation of a job, or fires a request for modifying the resources to the Global QoS manager, which will be introduced later in this chapter.

5.3.2 Operability of Local QoS for AS VO MPEG-4 decoding

Figure 5.2 portrays the Local QoS connected to the functional part of the actual job. The Evaluator in the Local QoS periodically observes the difference between the estimated resource requirements and the actually used resources. Based on the estimation error and input data characteristics, the Local QoS unit handles two types of situations.

The first situation aims at controlling short-term variations of the resource utilization. To compensate those variations in the case that some of the tasks are scalable, the Local QoS manager can change their local quality settings. If this step does not sufficiently reduce the resource needs, the Local QoS can disable some steps of the decoding process by task skipping (e.g. a deblocking filter). The second situation occurs when the change in required resources has a long-term nature that needs control, e.g. the size of an object changing significantly. This case will be covered later when adding Global QoS control.

5.3.3 Resource-usage prediction of VOP decoding

The efficiency of the Local QoS control relies on the accuracy of the Resource Estimator block in Figure 5.8. Parametrical timing models as presented in Chapter 3 provide high accuracy, but the amount of parameters (for shape and texture reconstruction the model contains 17 parameters) needed for every macroblock seems not practical for periodic transmission within the video data. Therefore, we propose the following technique for obtaining the prediction.

1. The parameters are measured and inserted into the "user data" as part of the first VOP of the video object (the refresh rate can be chosen low enough to avoid too much overhead).

2. The prediction for the first VOP is calculated using the timing and bandwidth parametrical models as introduced in Chapter 3 and decoded parameters from the "user data" part of the input stream.

3. The prediction is used for the interaction with the Global QoS manager and the corresponding allocation of resources.

4. The parameter values of the actual decoding are extracted by the decoder.

5. The new model is calculated based on the measured parameter values from the actual decoding.

6. The size in macroblocks of the next VOP is decoded from the next VOP header.

7. The prediction is made by proportionally adjusting the parametrical models to the size of the next VOP. The resources are planned according to the proportionally updated model settings. Return to Step 3, unless the parameters are retransmitted.

A more formal description is now summarized. Let us denote $p(t_i)$ as the prediction of the resource usage of the VOP decoding, $N_{MB}(t_i)$ represent the amount of macroblocks composing a VOP at time t_i, and $m(t_i)$ the measured amount of actually used resources. The prediction of the resource usage of the next VOP is based on the following equations:

$$p(t_{i+1}) = m(t_i) + \Delta p(t_{i+1}), \tag{5.3}$$

$$\Delta p(t_{i+1}) = C \cdot (N_{MB}(i+1) - N_{MB}(i)). \tag{5.4}$$

By combining the above equations, the prediction of the resource usage of the next VOP at time t_{i+1} is obtained by:

	Dancer	Singer	News	Fish	Stefan
# VOPs	250	250	300	27	240
Rel. error $> 10\%$	34	36	0	5	82
Rel. error $> 20\%$	10	6	0	2	14
mean rel. error	5.31	4.43	1.54	6.47	8.25
std. deviation σ	7.39	5.40	1.41	8.01	7.11

Table 5.3: *Performance of prediction model at VOP level for various sequences with statistics of the prediction errors.*

$$p(t_{i+1}) = m(t_i) \cdot (N_{MB}(i+1)/N_{MB}(i)). \tag{5.5}$$

We have statistically evaluated the prediction model introduced above. Table 5.3 shows the measured mean value of the relative prediction error and the standard deviation of the relative prediction error for the five test sequences. It can be seen from the table, that the prediction model has a reasonably accurate prediction, since the mean relative error is small and the standard deviation is a small fraction, typically well below 10% of the total required execution time. Also the amount of VOPs with serious large errors above 20% is below 10% of the VOP length. We can conclude that the proposed prediction technique is a first feasible step for supporting the QoS control.

5.4 Hierarchical Quality-of-Service architecture

5.4.1 Introduction to QoS concepts

We now extend the single application execution to the case of running multiple applications in parallel on a multiprocessor system. Prior to presenting our layered QoS concept, we present a brief overview of QoS concepts of the literature.

The QoS concept has emerged from the optimization problem for network communication, employing applications with real-time requirements [4, 123]. Considering QoS at terminals [80], the early approaches have defined application execution at different quality levels. The system design was tuned for the delivery of one function or service. Later, concepts changed to resource management for several resources [65]. The introduction of scalability into applications (like MPEG-2 SNR scalability) allowed the implementation of QoS management at the decoder terminal [124, 117].

The QoS control is in general effectuated in three layers, which are as follows.

- *Resource layer* - the allocation of resources is considered as a basic and essential layer. This layer is typically controlled by an Operating System (OS). This involves a scheduling policy, allocation of memories and bandwidth allocation. We assume that an OS is part of the system.

- *Application layer* - the definition of the application-specific parameters and related interface to the resource layer.

- *User layer* - the delivered quality level and its degradation defined for a user.

In the domain of streaming applications and multiprocessor NoCs, all three layers have to be addressed [53].

In this thesis, we mainly focus on the above-mentioned application layer of QoS. More specifically, we investigate the *translation* of the application parameters into a request that is supplied to the resource layer and we present a concept for QoS that also enables the *controlled interoperability* of joint execution of *multiple* applications. For this purpose, we introduce a hierarchical QoS system that addresses both the optimization of resource allocation for individual applications and for a complete set of applications. Furthermore, we will not discuss the QoS service algorithms in high detail. Instead, we assume that an algorithm from literature can be adopted, given the broad availability of proposals this topic [76, 2]. Hence, we introduce a optimization protocol for hierarchical QoS, which is a missing link between the resource allocations and the algorithm requirements. In the sequel, we discuss two existing QoS approaches: the *reservation-based* and the *adaptation-based* approach.

In order to be able to provide a reservation-based management, the resource management layer has to provide *guaranteed services* [43] for allocation of resources. If the system would allow a runtime adaptation of resources, the *best-effort* [46] and monitoring services [102] are essential for the correct operation.

1. Reservation-based approach
The benefit of a reservation-based QoS mechanism is that it guarantees the delivery of a defined quality level of an application after the successful reservation of resources. However, in the past, the reservation-based technique was not adopted due to the inaccurate resource-usage prediction and the long reservation time [67].

2. Adaptation-based approach
The adaptation-based QoS mechanism [49] addresses a best-effort computation. Several programming models and middleware components were defined, e.g. see [103]. However, the adaptation of resources takes place only when deadlines are missed and can be limited to a small amount of parallel applications. Furthermore, the dependencies between tasks on different processors of a NoC can bring the system into deadlock. In the next section we define our hierarchical QoS approach that is primarily based on a reservation-based technique.

5.4.2 Layered architecture of QoS and requirements

The architecture of our new QoS concept has a hierarchical layered structure. It consists two communicating managers, instead of the conventional single resource manager. The layered approach separates the system control optimizing overall quality and behavior from the responsibilities of individual application QoS units. The advantage of the layered approach is that the Local QoS control of individual applications can designed along with the application and independent of the platform where they will be executed. Similarly, the Global QoS multiprocessor control can be designed without knowing the details of all applications that will be executed. Thus, applications can be reused on other platforms more easily. This separation of responsibilities supports compositionality and modularity of the system in order to upload new applications to the existing system. The responsibilities of the two QoS managers are as follows.

- *Global QoS manager* - it controls the total system performance involving all applications running in parallel. This manager optimizes the user benefits instead of a single video application.
- *Local QoS manager* - it controls an individual application within the assigned resources (see Figure 5.9), which were assigned by the Global QoS manager. This manager optimizes the application quality for the agreed amount of resources.

Since the responsibilities of both QoS managers are essentially different, it becomes apparent that a protocol between these two QoS managers will be needed.

The overview of the new architecture and QoS control is shown in Figure 5.9. Each application is divided into jobs and the platform supports the execution of each job. Each individual application is controlled by a Local QoS unit, which negotiates about the assigned resources with the Global QoS control. After

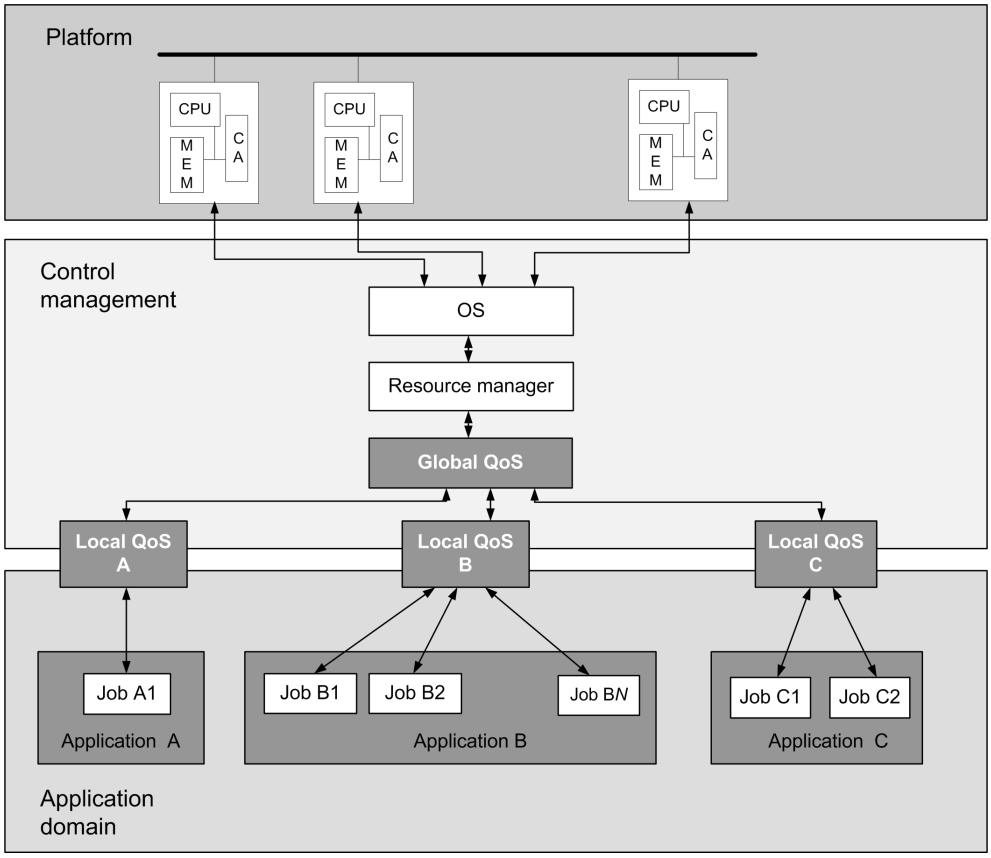


Figure 5.9: *Layered view of the system with the hierarchical QoS.*

negotiation, the Local QoS unit has an exact specification of the amount of resources that are assigned to the application. The Global QoS unit depends on the *Resource manager* for the actual resource assignment. The resource manager provides the real allocation of physical resources in conjunction with an operating system.

The control of an application execution is conceptually visualized in Figure 5.10. The execution starts with the activation of the Local QoS unit. The Local QoS unit has to activate the Resource Estimator (see Fig. 5.8), which calculates the resource-usage requirements at all quality levels of the required jobs. In Step 2, the query is send to the Global QoS unit, which decides on the highest quality level that can be guaranteed and allocated for the application. In Step 3, the response is communicated back to the Local QoS unit. In the case that resources can be assigned, the Local QoS unit allocates

individual tasks of the job at the chosen quality and invokes the job execution. In Step 5, the job signals the end of the processing period. In our case, this means the completion of the VOP decoding. In Step 6, the Local QoS unit evaluates the difference between the estimation and real execution and in the case the difference is above a predetermined threshold, the Local QoS adapts the estimation model. If this does not sufficiently lower the difference, a new negotiation on resources has to be performed (which involves restarting of the process from Step 2).

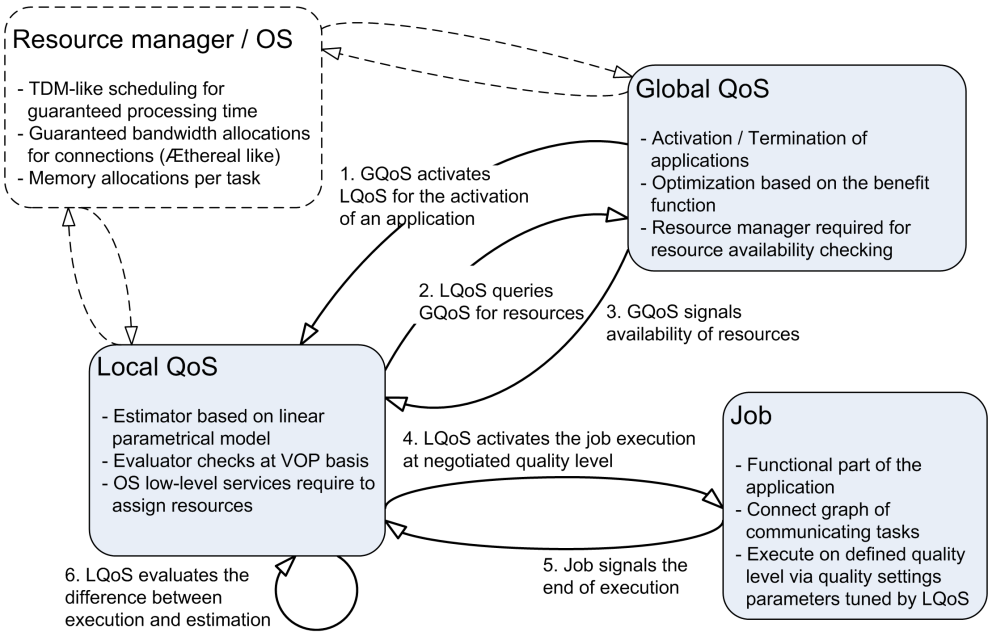


Figure 5.10: *Execution of a job in the system with the layered QoS control. The numbers at the arrows indicate the order of interactions.*

The use of the presented layered architecture for system quality control is most beneficial if a sufficiently accurate execution model for the application mapping can be provided. In the case there would be a large error in the resource estimation, in general, the estimation of total resources would be based on the number of jobs and their individual erroneous estimates on resource usage. However, the measurements as provided in Section 5.3.3 have shown that this rarely happens and the proposed simple prediction model already achieves an acceptable prediction accuracy.

Let us discuss a few architectural requirements and aspects of the layered QoS management.

- We require determinism in the QoS for each job, independent of other jobs. If two or more jobs share a resource, which has not a deterministic arbitration on serving several jobs, the predictability is not guaranteed. Hence, the scheduling algorithm should be deterministic.

- The task distribution over the processors should be explored according to worst-case rules. This is required in order to be able to analyze the worst-case use of communication resources. The implementation to do the worst-case analysis is based on using so-called virtual processors and virtual connections. The assignment of a task to a virtual processor is one task per one virtual processor. Similarly, the assignment to a virtual connection is one task-to-task communication link per virtual connection. The virtual processors and connections are runtime assigned to the existing resources of the platform based on the agreed allocations.

- We assume that the reconsideration of resource reservations after the reservation period is performed on a coarser granularity than the periodical control of a job by the Local QoS. The choice of a coarser granularity for a full reconsideration is motivated by the complexity of re-allocation processes (e.g. requiring task migration). For our MPEG-4 application, we have defined the reservation period to the length of a GOV and the periodic control of the Local QoS on a VOP basis[3]. In a real implementation, the length of the reservation period should be assigned based on the performance or response time of the resource manager or the OS.

5.4.3 QoS problem definition

This section concentrates more on the algorithm for actual QoS control. Let us now specify the QoS control problem in a more formal way. We describe the resource requirements of the job i at all defined quality settings (vector $\mathbf{q}_i$) per resource J to be a function of the quality settings, hence

$$R_{i,J}(\mathbf{q}_i) = f_J(\mathbf{q}_i). \tag{5.6}$$

The resource type $\mathbf{J} \in \{\mathbf{C}, \mathbf{D}, \mathbf{I}, \mathbf{B}, \mathbf{T}\}$, where $\mathbf{C}$ denotes the computation resources per task, $\mathbf{D}$ the data memory per task, $\mathbf{I}$ the instruction memory per task, $\mathbf{B}$ the required communication-port bandwidth, and $\mathbf{T}$ the bandwidth on each connection between two tasks.

Next to the definition of job resource requirements, we now define a benefit function. This function represents the overall system value function (e.g. qual-

[3]The typical length of an MPEG-2 GOP is 12-24 frames. However, MPEG-4 and related video standards use more variable GOV lengths. In such a case, we assume that a reasonable average GOV length is used in the same order of magnitude as a GOP length in MPEG-2.

ity) for the end-user. We define the benefit $\beta_i(\mathbf{q}_i(c))$ as the contribution of job i to the user benefit, at selected quality level c, giving the quality $\mathbf{q}_i(c)$.

Let P_J be the total amount of a particular resource type J per processing tile in the platform. For example, when we refer to data memory, P_D represents the total amount of data memory per processing tile and similarly, P_C stands for the total amount of computational resources per tile. For more details, the reader is referred to Section 3.8[4]. The total amount of resources of the system is computed by adding all P_J values for resource type J.

The optimization problem for our Global QoS management is now defined as follows.

$$\max \sum_{i=1}^{N} \beta_i(\mathbf{q}_i(c)), \text{with chosen quality } \mathbf{q}_i(c),$$

$$\begin{aligned}
\text{subject to} \quad & \sum_{i=1}^{N} R_{i,C}(\mathbf{q}_i(c)) < \sum_{j=1}^{M} P_C(j) \\
& \sum_{i=1}^{N} R_{i,D}(\mathbf{q}_i(c)) < \sum_{j=1}^{M} P_D(j) \\
& \sum_{i=1}^{N} R_{i,I}(\mathbf{q}_i(c)) < \sum_{j=1}^{M} P_I(j) \\
& \sum_{i=1}^{N} R_{i,B}(\mathbf{q}_i(c)) < \sum_{j=1}^{M} P_B(j) \\
& \sum_{i=1}^{N} R_{i,T}(\mathbf{q}_i(c)) < \sum_{j=1}^{M} P_T(j)
\end{aligned} \tag{5.7}$$

In the above equation, N denotes the number of jobs and M indicates the number of processing tiles. The optimization has to find the combination of jobs and their quality settings such that the overall benefit is maximized. The complexity of this optimization grows with the dimension of resources and job-description vectors. Therefore, in further experiments, we consider only computation, communication and data storage resources.

[4]In Chapter 3, we have used a more restricted version of tasks, called actors. In the current chapter, we allow the execution of generalized forms of tasks with an arbitrary nature of computing and communication. For example, a task execution is allowed before there is an input on all incoming edges. Furthermore, the application does not have to be always modeled by an SDF graph.

Although having specified the problem statement formally, at this point, we will not concentrate on finding an algorithm for the exact optimization, because such an algorithm would require to explore a large design space. This is due to the fact that the above definition of the problem can be transformed to the 0-1 Knapsack Problem [72]. Since this is an NP-hard problem, it cannot be implemented at runtime. Therefore, instead, in Section 5.4.4-B, we provide a heuristic runtime implementation which provides a near-optimal solution.

5.4.4 Heuristic algorithm for multi-job quality optimization

A. Strategic aspects for deriving the heuristic

In order to simplify the problem of finding the operational point close to the optimum as defined above, we first analyze in detail the application at the design phase. Second, after the analysis, we have computational and memory usage requirements of individual tasks, and communication requirements between tasks. Third, we select the maximum requirement that occurs within a particular time window (reservation period, depending on the chosen granularity). Fourth, this request is communicated from the Local QoS to the Global QoS control unit, which tries to satisfy this request. In the algorithm, we assume that a lower quality level requires less resources. This is a reasonable assumption for which the practical evidence was given in Table 5.1.

The resource manager (see Figure 5.9) controls the available physical resources in conjunction with the Global QoS manager, thereby using the resource-usage requirements defined at the design phase. Based on off-line measurements of the anticipated quality resulting in benefit $\beta_i(\mathbf{q}_i(c))$, the Global QoS manager strives for a quality setting that would satisfy the user. At this point, the resource manager (OS) is of key importance, as it keeps track of the free capacity of all physical resources in the platform. Given a set of resource requirements per task, the resource manager should find a physical processor with sufficient free capacity. It may happen that the resource manager cannot accommodate the resources for the new job. If the job has a high importance (higher benefit $\beta_i(\mathbf{q}_i(c))$), the Global QoS manager may decide to decrease the quality settings of some other jobs to release resources for the new job. The details of the algorithm optimizing the quality is presented in the following paragraphs.

B. Algorithm description

For simplicity, we assume that the system is in operation and has a set of running jobs. We consider four types of possible situations in the running system: (1) a job fires a request to be started, (2) a job requires more resources, (3) a job is terminated, (4) a job releases some resources. The second possibility

Algorithm 1 Heuristic QoS optimization

```
1: procedure MappingJob()
2: while not(IsEnoughResources(CandidateJob)) do
3:    JobToDecrease = FindMinBenefit(ActiveJobs + CandidateJob);
4:    if JobToDecrease! = CandidateJob then
5:       SetQualityLevel(JobToDecrease, NewQuality)
6:    else
7:       LowerQualityOfCandidate()
8:       if CandidateQuality < Minimum then
9:          return
10:      end if
11:   end if
12:   if IsEnoughResources(CandidateJob) then
13:      MapJob(CandidateJob)
14:      return
15:   end if
16: end while
17: if not(IsEnoughResources(CandidateJob)) then
18:    ReportInsufficientResources()
19: end if
20: endprocedure
```

is a special case of the first, and similarly, the fourth situation is a special case of the third. Let us further describe the algorithm of the negotiation process for a request to start a new job. The algorithm is a simplified version of checking the availability of resources for the chosen quality of a *Candidate Job*.

The algorithm starts by checking the ability of adding the *Candidate Job* to the list of *Active Jobs*. The *Resource Estimator* from the Local QoS manager of a candidate job calculates the required resources. In the case that there are not enough resources, a search for the minimum quality decrease of active jobs to the overall cost function is performed (minimize the decrease of Equation (5.7). For an efficient implementation, the Global QoS is storing a *sorted list* of such quality changes, which brings the searching algorithm for finding a new maximum to a linear complexity. The algorithm decreases the quality of the set of jobs by reducing the quality of individual jobs, followed by checking whether the system has sufficient resources for the *Candidate Job*. The algorithm ends when the system has enough resources for executing the new set of jobs. If the benefit cost function drops below the level at which the system tried to activate the *Candidate Job*, the algorithm also terminates. The search for a sub-optimal quality assignment based on the resource availability is summarized in Algorithm 1. The detailed description of individual functions used in the heuristic algorithm are listed in Table 5.4.

Function	Description
IsEnoughResources	The function checks if the system can reserve resources at the required benefit $\beta_i(\mathbf{q}_i(c))$. It returns "true" if the allocation is possible.
FindMinBenefit	The function search for a job that contributes with the lowest benefit β_i to the overall system value, returns the job index with the lowest benefit increase.
SetQualityLevel	The function assigns the quality level to the job. The function is called only for the jobs having a lower benefit than the actual benefit level and it is assumed that a lower benefit requires less resources.
LowerQualityOf Candidate	The function decreases the benefit level of a candidate job, the lowest level is 0.
MapJob	The function calls the *Resource manager* routines to allocate resources defined by the quality benefit level of a job.
ReportInsufficient Resources	The function reports to the Local QoS that it is not able to execute the candidate job.

Table 5.4: *Definition of Global QoS functions.*

In case that the platform releases more free resources, either by terminating some of the jobs, or by changing requirements of one or more jobs, depending on the input data or a user interaction, we introduce the following strategy. The algorithm starts with a job that gives the highest benefit increase and checks if it can increase a quality level by using new available resources. For efficiency, the system has information about the minimum resources that can increase at least one quality level of a job. If available resources are below this level, the negotiation algorithm stops (the algorithm stops only when it has finished the examination of all active jobs).

Let us discuss some aspects of the presented algorithm.

- The reader may be confused that we search both for maximum and minimum increase of benefit at different parts of algorithm. The algorithm principle is such that the decrease of the job quality that minimally lowers the augmentation of the benefit function results in a minimum decrease of the overall benefit function. It can lead to gradually decreasing the overall quality in a number of iterations.
- The benefit augmentation of an inserted job is included to the sorted list and put at the appropriate position prior to the next iteration.

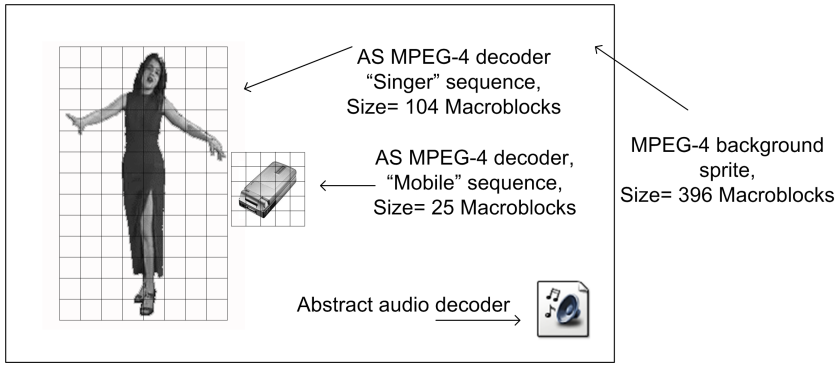


Figure 5.11: *Example of an object-based video scene with indicated benefits and computation complexities (Intended scene position is specified in the MPEG-4 BIFS scene description.*

- The proposed algorithm is sub-optimal due to the following reasons. First, this is because the system strives for satisfying a set of worst-case requirements over the allocation period. Second, sub-optimality occurs because the granularity of processing is limited, so that it cannot be guaranteed that the optimal point is inside the reservation interval.

5.5 Global QoS experiments and results

In this section, we report on experimenting with the presented heuristic algorithm in order to set a quality assignment for the AS VO MPEG-4 decoding application. We reused previously obtained results on the execution of the AS VO MPEG-4 decoder operating at different quality levels (see Section 5.2.2).

Job	Weight.
AS VO MPEG-4 decoder - "Singer" video object	0.9
AS VO MPEG-4 decoder - "Mobile" video object	0.7
Abstract audio decoder	0.5
MPEG-4 sprite background decoder	0.45

Table 5.5: *Benefit weights of individual jobs.*

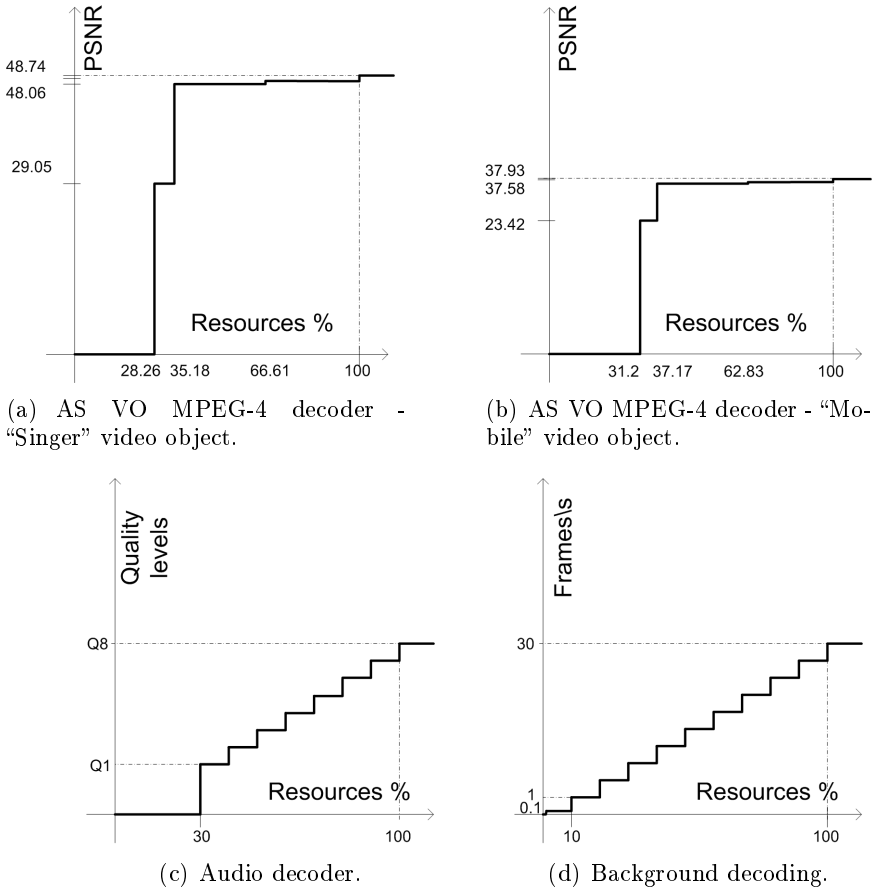
An example of a resulting video-object scene is shown in Figure 5.11. Evidently, the highest benefit weight is assigned to the decoding of the "Singer" video object, followed by the decoding of the "Mobile" video object with benefit weights 0.9 and 0.7, respectively. The video decoding is enhanced with supplementary audio decoding and MPEG-4 sprite decoding. The audio ob-

ject (we inserted a dummy job) has weighted benefit 0.5, and the background sprite decoding has the lowest benefit weight contribution of 0.45. Table 5.5 summarizes the predefined benefit weights of individual jobs of the complete experimental application. The figure also illustrates the different computation needs. The computational requirements of both video objects grow with the number of macroblocks that has to be processed. It can be observed from Figure 5.11 that the required number of macroblocks is varying per object, so that the required resources will also vary accordingly. The size of the "Singer" video object is a rather dynamic variable. It starts with 64 macroblocks, continuously grows to 204 macroblocks and then it decreases to about 96 macroblocks (see Figure 3.5 in Chapter 3). The varying nature of the video-object size and the corresponding computational requirements have an effect on the setting of different quality levels per job over the length of the appearance of the video object in the scene.

The Resource Estimator is a job-specific component and depends on the implementation of the functional part of the job. The interface between the Global QoS manager is specified to allow simple query-type requests about the required resources for a specific quality level of a job, for a specific set of input data. The output of the Resource Estimator is the matrix $R_{i,J}(\mathbf{q_i})$ per resource J as defined by Equation (5.6). In the conducted experiment, the Resource Estimator was executed with the actual parameters of the parametrical models, instead of using the prediction technique in Section 5.3.3, which was developed at the time of writing this thesis.

The quality dependencies on the available resources are depicted in Figure 5.12. Each figure refers to the requirements of a single object. Due to the size dependencies of arbitrary-shaped video objects, the quality distortion is represented by the percentage of required resources per complete job, normalized to the average size within the sequence. The different PSNR values refer to the averaged PSNRs values obtained at the different quality settings. The quality of the MPEG-4 background decoding is expressed in the frame throughput rate, rather than PSNR. This is because the PSNR of the background image depends strongly on the global camera motion and the visible part of scene background. The frame throughput characteristic fits better to the scalable approach on the quality degradation. The audio processing has similar characteristic as the background decoding. Whereas the first two jobs have a few quality levels and require a high amount of resources, the other two have a more linear characteristic of the obtained quality per used resources. This last property gives more opportunity for scaling the performance of the system.

Another essential component of the Local QoS manager is an *Evaluator* that has the responsibility to measure whether timing requirements of a job are



(a) AS VO MPEG-4 decoder - "Singer" video object.

(b) AS VO MPEG-4 decoder - "Mobile" video object.

(c) Audio decoder.

(d) Background decoding.

Figure 5.12: *Quality resulting from the used amount of resource per job. The PSNR values are average numbers for the duration of video objects.*

satisfied. In the ideal situation, the difference between the output of the Resource Estimator and the results of the real execution is zero. In practice, the input-data dependent processing may result in the under- or over-usage of resources. For example, when the object enlarges its size, a larger amount of resources is required. In such a case, the Local QoS fires a request to the Global QoS for a *reconfiguration* to obtain more resources. In the case the requested assignment is not possible, it lowers the quality level at which the job is executed.

We have simulated the heuristic optimization Algorithm 1 within a MATLAB

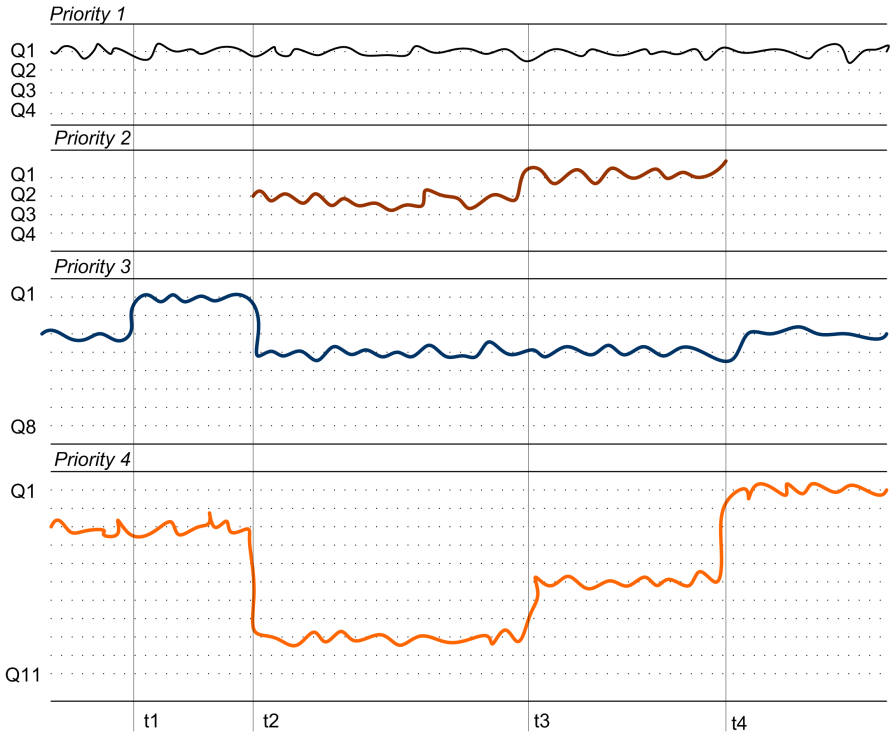


Figure 5.13: *Example of the dynamic change of qualities among multiple running jobs controlled in parallel.*

framework. Figure 5.13 portrays an example result of the quality assignment algorithm for setting the quality level for individual jobs. AS VO MPEG-4 decoding jobs use traces [5] of the clock-cycle-true ARM7TDMI simulator, which executes the AS VO MPEG-4 compliant decoder. The quality level Q4 means decoding without performing the deblocking filtering and padding of video objects (see [92]). For the audio object, we have applied a random resource-usage generator with eight abstract quality levels. For the background sprite image decoder, we again used the traces of the executed Core Profile MPEG-4 background sprite implementation on the ARM7TDMI simulator. For this job, we defined 11 quality levels, corresponding to the decrease of the frame rate for this video object in the range 30 to 0.1 frames/s. For the experiment, we have manually assigned tasks to specific processors for each job.

Let us discuss the dynamic behavior in Figure 5.13. We focus on times $t1..t4$, when the change of a job quality setting occurs. At time $t1$, Job 1 changes its

[5] The traces refer to the clock-cycle-true execution times of the job on the target multiprocessor system.

requirements on the resources and offers the resources to the system. These resources increase the quality level of Job 3. At time $t2$, Job 2 starts and decreases the quality level of Job 3 and Job 4. At time t3, Job 1 releases some resources (video-object size shrunk to almost 1/2 of the size in the previous allocation period) that fit for Job 2 and Job 4. Additionally, Job 2 terminates at $t4$. It can be observed that Job 3 did not reach its original quality before time $t2$. This is because the evaluation of the cost function has indicated that it was better to increase Job 4 to a higher quality level than to increase Job 3. At the beginning, there were not enough resources for Job 4, but Job 1 facilitated by decreasing its requirements (not in Figure 5.13) at $t1$ and $t3$.

The experiment above showes a functional behavior of the Global QoS optimization algorithm for a specific set of jobs. In order to get a realistic behavior, we have used traces of individual jobs executed on the clock-cycle-true simulator of the target processor (Job 1, Job 2 and Job 4). Furthermore, we set an amount of the available resources manually to a level at which the system cannot accommodate all jobs at the highest quality and the heuristic Algorithm 1 has to be performed.

5.6 Conclusions

We have proposed a new hierarchical QoS management system that is able to control a set of multiple jobs executed on a resource-constrained multiprocessor system. The QoS contol is split into two layers: a Global QoS for overall system control and a Local QoS for individual application control. Local QoS control is based on the resource-estimation functions that were derived in Chapter 3. The current chapter has addressed the QoS problem, and proposed and approach based on task switching within the AS VO MPEG-4 decoding algorithm to create a quality-scalable application. The enabling and disabling of tasks proved to be a simple, but effective method to create scalability of quality without having to redesign the complete decoding algorithm.

We have presented a hierarchical QoS architecture, containing both Local and Global QoS. The benefit of this proposal is that the control of individual applications is separated from the optimization of the global system performance. The concept is such that an operating system can be gluelessly integrated within our architecture, because it works on a much more local level (instructions) than the other algorithms. A second benefit of this architecture is its flexibility with respect to the used resource estimator. For example, the chosen prediction based on the parametrical models can be replaced by other types of resource-usage modeling.

A further contribution of this chapter is that we have defined a heuristic algorithm that searches suitable combinations of quality levels per jobs that can be mapped on the available resources. The algorithm decreases the quality of the set of jobs by reducing the quality of individual jobs, followed by checking whether the system has sufficient resources for serving a resource allocation request. At the Global QoS, the approach is based on a pure reservation-based approach. An extension of this approach within the Local QoS control will be presented in the next chapter.

The hierarchical approach was implemented in an experimental MPEG-4 decoder for arbitrary-shaped video objects. It should be noticed that the amount of saved resources depends on the contents of the test sequence and the utilization of the system. We have found that prediction of resource utilization can lead to significant quality improvement of the complete system when resources are offered for other applications. The proposed QoS mechanism runs fast enough to be executed in real time, because it operates only on sorted lists of benefit functions and related parameters.

When looking back to the results, it looks as if some of the concepts could have been borrowed from the literature in the past decade. However, there is a major difference between that research and our work. The concepts were all based on a single computer engine for which the scheduling can be analyzed analytically. In a multiprocessor system, the quality optimization is an NP-hard problem that cannot be easily solved analytically. Even in the multiprocessor case, the experimental mappings concentrate on one task per tile mapping. As this is inherently inefficient, we have chosen a sub-optimal solution that applies to the more general case of having a non-integer amount of tasks or jobs per processing tile.

Chapter 6

Local QoS for BW-constrained MP-NoC using BE services

The reservation-based QoS control from the previous chapter is not efficient in special cases. For example, when a job has a dynamic behavior within the adaptation interval, resources will be left unused. To improve the efficiency in all cases with respect to unused resources, we explore the possible benefit of adding Best-Effort (BE) computing principles for the communication resources within an MP-NoC. This exploration on best-effort bandwidth usage is a first step in investigating the complex problem of combining best-effort and reservation-based computing for multiple parallel tasks running within one multiprocessor system. For our bandwidth study, the original NoC architecture is extended by monitoring units to provide communication resource usage during execution. Experiments reveal that the proposed mixed approach of reservation-based and best-effort computing yields full bandwidth utilization, so that the SNR of the video object signal improves by 1-5 dB.

6.1 Introduction

The disadvantage of a reservation-based computing architecture is that the resources may be left unused when the execution of the video processing is not as difficult as predicted. This inevitably leads to a lose of computing efficiency. This phenomenon also occurs during AS VO MPEG-4 decoding. Initial experiments to start up the research of this chapter have shown that the efficiency of the reservation-based QoS control yields only an average efficiency

of about 70%. For this reason, we focus on further maximizing the possible output quality by using the reservation-based technique in combination with a best-effort runtime adaptation of the computation. The concept behind this idea is that the local QoS system can control towards higher quality as soon as resources become available. This assumes that scalable video-coding algorithms are used, as introduced in the previous chapter.

Let us give an idea here how reservation-based and best-effort computing can be combined in one system. The execution model is such that at global level, reservation-based computing is employed, whereas the best-effort quality improvement is obtained by the activation of idle tasks. The activation is possible when sufficient remaining resources are reported. Most of the existing literature reports on studies of scheduling of tasks on a single processor [19, 79].

In this thesis, we study a multiprocessor system in which a network of communicating processors is included. The efficient mapping of multiple tasks onto a multiprocessor system is an unsolved case in the scientific literature. This is due to the large design space and the fact that operating systems are basically designed for single processor execution. The mapping can be explored for efficient parallel computing, efficient bandwidth usage of the available communication links and a suitable memory architecture for global and local data. It is only recently that the first publications on this topic have become available. In [17], an integration of hard/soft real-time tasks and best-effort jobs executed on a multiprocessor system is studied. The study involves computing only and does not address other resources. The optimal scheduling of tasks over a multiprocessor network is not yet found. If the processors operate independently of each other, then the solution of [19] can be applied on an individual processor basis. This chapter attempts to contribute to the above problem exploration by concentrating on a parameter that was not yet studied: the bandwidth between the processors. In order to do this, we have to make assumptions on how we deal with computing. We have chosen map tasks on individual processors and we will act as if the communication network is the most limiting factor in the efficiency of the execution[1].

In order to serve bandwidth control in our multiprocessor realization, the NoC architecture is extended with bandwidth-monitoring elements. The concept is such that when reservation-based computing and bandwidth do not fully utilize the available communication bandwidth, then the remaining bandwidth is filled with best-effort tasks that improve the quality, but can be partially aborted. Therefore, we present a new experimental architecture, which includes event-

[1] In many modern multimedia applications, the bandwidth of processors to the memory is one of the most limiting factors [55].

based monitoring services which are used by the Local QoS for best-effort control. The new architecture originates from a cooperation between our work and [24], which is an extension of the Æthereal NoC (see Chapter 3). We have experimentally validated the combined concept of reservation-based and best-effort computing by executing an AS VO MPEG-4 decoder and installing the above-mentioned bandwidth-monitoring probes. Experiments later in the chapter report on a significant image quality improvement, where the absolute PSNR of approximately 35 dB is enhanced with 1-5 dB.

This chapter is organized as follows. Section 6.2 discusses the execution of an AS VO MPEG-4 decoder on a pure reservation-based system and motivates in more detail the extension with best-effort computing. Section 6.3 explains the NoC extension with network-monitoring services. Section 6.4 defines the combination of best-effort computing with reservation-based processing. The experimental mapping of the AS VO MPEG-4 decoder is tested and the resulting quality improvement is summarized in Section 6.5. Section 6.6 concludes this chapter.

6.2 Limitations with reservation-based QoS

The reservation-based QoS as defined in the previous chapter, fulfils the requirement of a predictable system. One of the problems in reservation-based computing is that the resources are locked for a particular job for a certain period. From this, it can be directly concluded that the reservation period is influencing the efficiency of the resource usage. On the other hand, the reservation period cannot be chosen arbitrarily small, since this would generate too much overhead and loss of efficiency. Furthermore, a small reservation period would hamper exploiting the temporal correlation over a certain period.

Let us further elaborate on this aspect. The most well-known temporal dependency within the MPEG hybrid coding architecture is the motion estimation and compensation. The temporal depth in MPEG-2 coding is restricted to a Group Of Pictures (GOP), which also specifies the periodic restart of intraframe coding. This forms a natural point of granularity for the reservation of resources[2] However, the MPEG-4 standard does not have a GOP length but uses a Group Of Video object planes (GOV), which is a sequence of pictures per individual video object. For a realistic case, we defined the GOV to 12 frames or more. This discussion leads to the following limitations of reservation-based processing.

[2]A typical GOP length is 12, 15, 21 or 24 pictures per GOP for MPEG-2 coding, for DVD and DVB standards.

1. Relatively long resource-reservation interval
For AS VO MPEG-4 decoding, the reservation of resources for the whole GOV requires that the system has sufficient resources for decoding each Video Object Plane (VOP). However, the MPEG-4 GOV length is not known in advance and is determined by the actual encoder. Therefore, the QoS control of the decoder has to decide on the reservation of resources for the decoding application for the complete length of a GOV. The GOV number is a variable parameter: we have observed sequences of up to several hundreds of VOPs in one GOV. In the worst case, the decoder QoS control has to decide only on a fragment of the GOV size. Consequently, this approach can sometimes lead to a QoS decision for a lower quality level for a long sequence of VOPs. This lowering of the chosen quality level already occurs when only one VOP cannot be decoded within the available resources.

2. Slow response on the increase of available resources
We have observed that the reservation-based QoS is also sloth in covering the increase of available resources. The time for the reallocation of resources and increase of the guaranteed quality level for an application is only possible at the end of the reservation period. When the quality levels of other jobs change or when a termination of other applications occurs, the new available resources cannot be directly used for the subsequent VOP decoding. The decoding at a higher quality level starts at the first frame of the *next reservation period* after having detected that resources are available. In the case that such an increase of resources appears at the beginning of the reservation period, the response of the system may be too slow for the system user. These two limitations motivated to supplement the reservation-based model with a runtime QoS adaptation.

6.3 Bandwidth monitoring within an NoC

The observation of ongoing computations in a system has received considerable attention in the literature of which a few publications will be discussed here. NoC monitoring systems have been proposed [24] for observing the communication at runtime. This work was mainly driven by testing and debugging aspects of newly designed system. Passive hardware monitors make use of a real-time observability solution, called SPY [114]. The runtime use of monitored data by the Operating System (OS) has been proposed by Nollet *et al.* [79]. This proposed solution requires an extra NoC to communicate the monitoring data for feeding the OS. Even though this solution can be used for monitoring of NoC communication, the extra NoC required for monitoring data only is simply too expensive for embedded applications.

Instead, we present a solution based on the Æthereal [42] NoC that is extended with runtime monitoring features inside the NoC, so that an extra network is avoided. The network extensions in the form of components were designed by the author of [24]. Our contribution in this thesis is to deploy this novel concept for bandwidth control of an advanced multimedia application. Up till now, the monitoring was used mainly for debugging purposes. The role of monitoring becomes more valuable when it is coupled to an advanced resource management that can explore the monitoring information for better distribution of resources. Our mechanism of using the monitoring information is highlighted in the next section. Here, we describe individual components and their meaning by which the NoC architecture is extended.

Figure 6.1 illustrates the NoC architecture with routers (R) and Network Interfaces (NI). The new NoC monitoring in Figure 6.1 consists of configurable *monitoring probes* (P), attached to the R and NI components, and their associated programming model. Also, the system uses a monitoring traffic management strategy.

The *monitoring probes* are responsible for collecting the required information from the NoC components. The probes capture the monitored information in the form of events. Multiple classes of events can be generated by each probe, based on a predefined instance of an event model. Monitoring probes are not necessarily attached to all NoC components. The placement of probes is a designer choice and is related to the cost versus observability tradeoff.

The *traffic management* regulates the traffic in the NoC by control signals from the Monitoring Service Access point (MSA) to the probes. These signals are also used to configure the probes. Similarly, the monitoring data from the probes to the MSA are used to obtain the monitoring information from the NoC. The proposed architecture with the new components is flexible for the mapping of applications and allows different types of QoS control. Hence, it covers the already available NoC communication services, e.g. guaranteed throughput (GT) or best-effort (BE) connections[3], or even dedicated solutions for the traffic information in case of a fixed mapping.

The above framework is integrated in our experiments in the following way. The presented NoC with communication-monitoring features offers the combination of mixed GT and BE connections. GT connections provide the reservation-based QoS control of the communication. BE connections are inserted to complete and plan resource usage up to or close to 100%.

[3]The reader should note the difference between reservation-based services and the guaranteed throughput mentioned here. GT is a reservation-based service for *communication resources*, whereas BE is a principle that applies to any kind of resource usage.

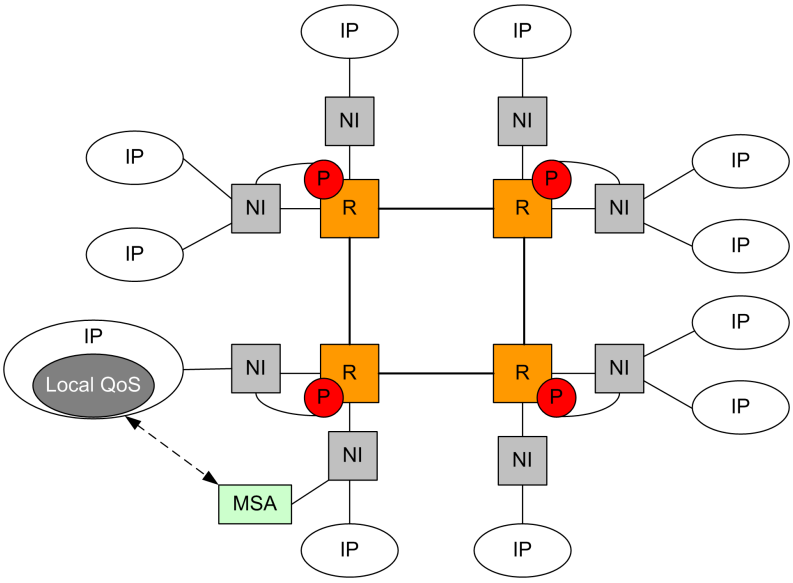


Figure 6.1: *NoC architecture diagram with a Monitoring Service Access (MSA) point connected to the Local QoS control.*

When comparing the earlier mentioned NoC monitoring solutions to the proposed one in this section, the advantage of our monitor-extended NoC is that it provides monitoring data as a guaranteed service from the router to a control unit (in our particular case the Local QoS managers) and a special extra NoC for monitoring data is not required in our approach.

6.4 Combining best-effort and reservation-based QoS management

In Section 6.2, we have identified the limitations of the reservation-based QoS model, and we indicated how to improve the efficiency by adding best-effort computing on top of reservation-based services. The advantage of this combined approach is that relying on guaranteed services avoids deadlock in executing jobs and it ensures a predictable performance and quality, whereas best-effort services increase efficiency of the mapping on the platform[4].

[4] An alternative case exists where all services are based on best-effort computing. It is known that in such a case the system can come into deadlock situations. These situations occur when e.g. one of the tasks is waiting for another one that has no sufficient computing capacity assigned to it. With guaranteed capacity, this cannot happen.

For clarity, we briefly repeat the functionality of the *Global QoS manager*. Global QoS assigns the resources and appropriate quality level to each application. The estimator calculates for each application the amount of required resources for processing the set of new data. The resource request is then evaluated with the available resources and the Global QoS manager sets the highest quality that just fits to the platform resources. These resources are reserved for the application until the end of the reservation period or till the moment that an exceptional situation occurs.

The *Local QoS manager* is responsible for monitoring the prediction model and real resource consumption. The monitoring of execution is performed at finer granularity, in our case at the VOP level. The Local QoS sets the parameters for scalable communication connections and scalable tasks at runtime, based on the resource availability. A diagram visualizing the connection of the Global QoS manager to the Local QoS manager and the usage of runtime monitors is depicted in Figure 6.2.

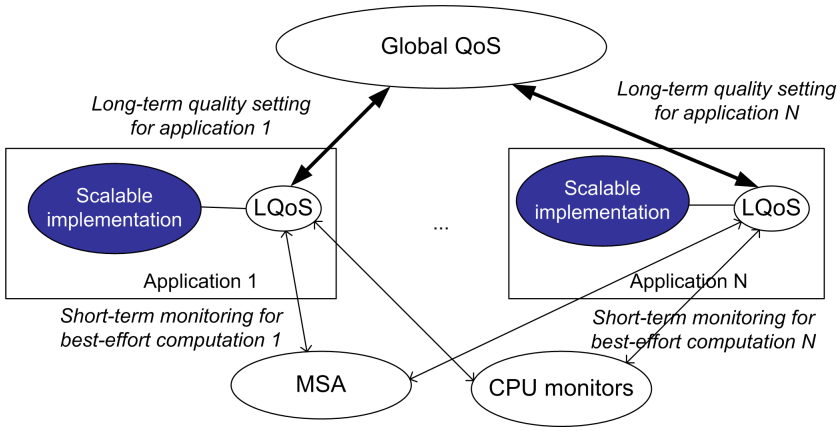


Figure 6.2: *Diagram for the combined QoS management containing MSA and CPU monitoring features.*

The Local QoS algorithm for extending reservation-based services with the *best-effort* principle is as follows. The algorithm checks for all tasks and connections at higher quality than the actual quality q_s if there are sufficient BE services to execute the application at higher quality. If such quality can be enabled, the highest level is selected and activated. A pseudo-formal specification is given in Algorithm 2.

The observability of the platform at runtime enables us to employ so-called *best-effort* principles to obtain a higher quality level for a short time (fragment

Algorithm 2 Local QoS best-effort extension

```
1: procedure CheckBestEffortAvailability()
2: for p = Current_Quality to Highest_Quality do
3:    if
4:    CheckAvailableResources(candidateTasksForBE); and
5:    CheckAvailableResources(candidateConnectionsForBE); then
6:       BE_QualityLevel = p;
7:    end if
8: end for
9: if  BE_QualityLevel! = CurrentQualityLevel then
10:   ActivateBEForQualityLevel(BE_QualityLevel);
11: end if
12: endprocedure
```

of the reservation period). If we would have only a reservation-based solution, the system would have to wait until the next suitable time (i.e. end of the reservation period) for changing the quality level and the corresponding resource allocation.

In the new solution, using the MPEG-4 decoding application, the Local QoS calculates for each VOP the resource requirements of the succeeding VOP and compares it with the runtime information from the MSA monitor and CPU monitors. The Local QoS temporary sets parameters for scalable tasks to a higher quality level for decoding of the next VOP only within the actual GOV. After the GOV, the Local QoS fires a request for setting the higher quality for the whole next GOV.

6.5 Bandwidth control experiment with AS VO MPEG-4 decoding

6.5.1 Scalable task-level AS VO MPEG-4 decoding

The scalable task-level AS VO MPEG-4 decoding as presented in Section 5.2, provides scalability in different types of resources.

- *Computation scalability* - the decoding chain is modified at runtime for activating/deactivating additional tasks (Padding tasks, Deringing, Deblocking).

- *Communication scalability* - the task-level scalability can modify the bandwidth requirements based on the task-to-processor assignment.

In order to deploy the combination of QoS techniques, we initiate the system with the worst-case mapping with respect to communication, in which we map each task to a different processor. After each GOV, this mapping is reconsidered. The mapping of several tasks to one tile is possible, however, a careful scheduling algorithm and monitoring is a research topic on its own. For this aspect, we refer to [19, 5], where scheduling problems are addressed.

We have defined three quality levels of our experimental AS VO MPEG-4 decoding to enable the task-level scalability.

- Level 0 - basic quality, the video object shape is fully decoded; the basic quality of texture after performing IDCT is communicated to the *Rendering* task.
- Level 1 - medium quality, the MPEG-4 padding [52] of the texture data is activated. As a consequence, there are no artifacts on video object edges. The deblocking filter is applied to the fully padded VOP.
- Level 2 - highest quality, the complete chain including the deringing filter is executed.

Prior to conducting the experiment, we indicate the complexity of the video objects occurring in several test sequences. All sequences have CIF resolution. Since objects have various sizes, we indicate the relative computational effort with respect to those sizes to obtain a more objective presentation of the complexity. Table 6.1 shows the distribution of task complexities as a fraction of the video object size. Table 6.2 shows the corresponding fractions required for communication. To map the tasks of Table 6.1 and Table 6.2, we have employed three processing tiles, where all decoding tasks are executed on one tile, all padding tasks on another, etc.

Sequence	Shape & Texture decoding	Paddings & Deblocking filter	Deringing filters
Singer	28.26%	38.35%	33.39%
Dancer	26.92%	36.22%	36.86%
News	23.21%	50.87%	25.92%
Fish	27.37%	41.08%	31.55%
Tennis	33.54%	34.88%	31.58%
Average	27.86%	40.28%	31.86%

Table 6.1: *Relative distribution of task complexity of VOP decoding tasks.*

Sequence	Level 0 Shape & Text.	Level 1: L0 & Padding & Deblock.	Level 2: All tasks proc.
Singer	39.8%	83.0%	100%
Dancer	39.4%	81.6%	100%
News	37.8%	83.5%	100%
Fish	42.6%	89.3%	100%
Average	39.7%	83.8%	100%

Table 6.2: *Cumulative bandwidth of AS-VO MPEG-4 decoder at different quality levels.*

6.5.2 Experimental architecture

Our experimental system architecture employs a 2×4 mesh Æthereal NoC with eight ARM processing cores. The ARM cores are one-to-one connected to Network Interfaces (NI). We have implemented a centralized performance-monitoring service. Each router has a probe for performance monitoring of the communication-link utilization. Each probe sends monitoring performance data to the MSA by means of a low-bandwidth GT connection through the closest located NI, which was designed with an extra NI port for this purpose. The single MSA connects to NI7 (see Figure 6.3) by means of an extra NI port. The complete overview is given in Figure 6.3. Apart from the previously mentioned extra NI ports, the communication of monitoring data is distributed over the regular NoC network in embedded form.

We have chosen the Advanced Coding Efficiency (ACE) Profile from the MPEG-4 standard with Level 3, at CCIR-601 resolution. The experiment was conducted in this resolution (not to be confused with CIF resolution for finding the complexity of tasks). Figure 6.4 illustrates the varying communication requirements of the "Stefan" AS VO sequence that was segmented from the original resolution of 688×464 pixels. The bold line indicates the other applications running in parallel within the system.

Reservation of resources. At the start of the GOV, the *Estimator* calculates the computation and communication resource requirements at all three quality levels. Next, the Global QoS selects the quality level at which all VOPs can be decoded. Periodically, at GOV level, the Local QoS fires a request to the Global QoS to raise the quality level of the reservation-based setting, when the best-effort adaptation was applied at the end of the GOV period. In our experiment (Figure 6.4), the lowest quality Level 0 is selected between VOP indexes 13 to 22 because of the exceeded upper limit of the available bandwidth.

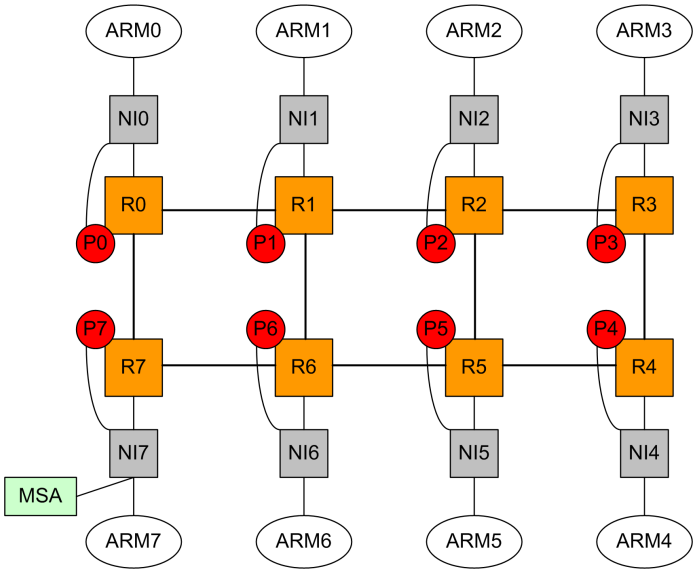


Figure 6.3: *Experimental system architecture with 2×4 mesh Æthereal NoC connected with eight ARM cores and the MSA.*

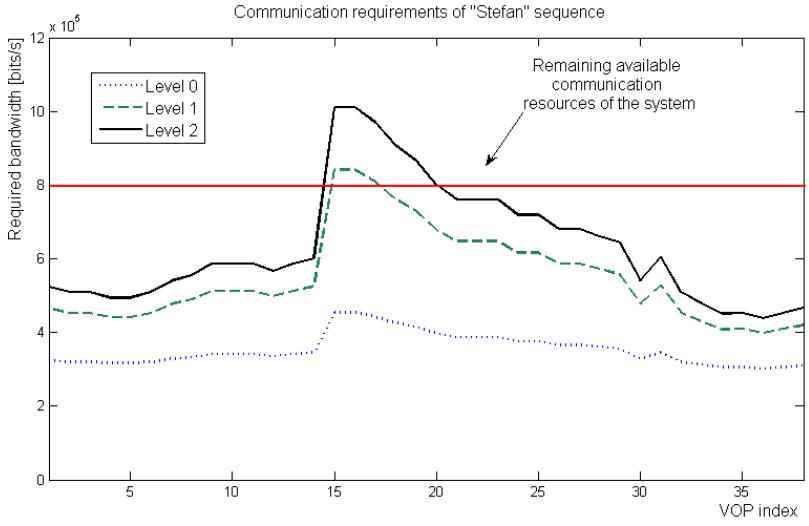


Figure 6.4: *Communication requirements of the "Stefan" tennis sequence. The bold line represents the remaining communication resources left for other applications also executed within the system.*

Best-effort computation. The Local QoS performs prior to the start of every VOP a check whether the NoC can offer supplementary BE bandwidth to obtain higher quality levels. The algorithm for doing this is indicated in Algorithm 2. The scalable task is then activated for the current VOP.

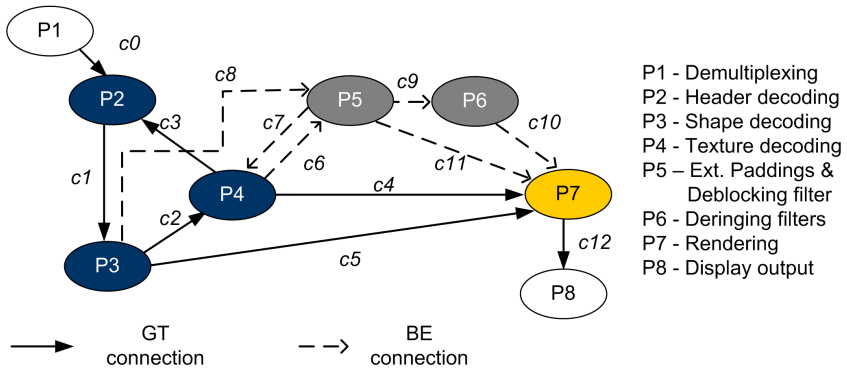


Figure 6.5: *Scalable mapping of an AS VO MPEG-4 VO decoder using task skipping. The dotted lines are optionally enabled or disabled in the same as in the previous chapter. P1 and P8 are extra tasks for data generation and collection.*

In our setup, we integrated *alien traffic generators* that program the system to a maximum level of communication activity. We assigned the following connections from Figure 6.5 to the corresponding quality levels:

- Level 0 : c0–c5, c12,
- Level 1 : all connections at Level 0 + c6, c7, c8, c11,
- Level 2 : all connections at Level 1 + c9, c10.

The Local QoS has to monitor the connections c6–c11, as they are of BE type. As is depicted in Figure 6.4, the initial quality is at quality Level 0. Prior to starting the next VOP decoding, the Local QoS checks the status of the connections and if the estimated communication resources are available, then it activates the scalable tasks at the highest possible level.

6.5.3 Experiment with a combined bandwidth control

This section provides the numerical results of executing the AS VO MPEG-4 decoder on the system with enabled best-effort computation. First, in order to illustrate the obtained image quality, we provide measurements of the PSNR at the different quality levels as defined previously.

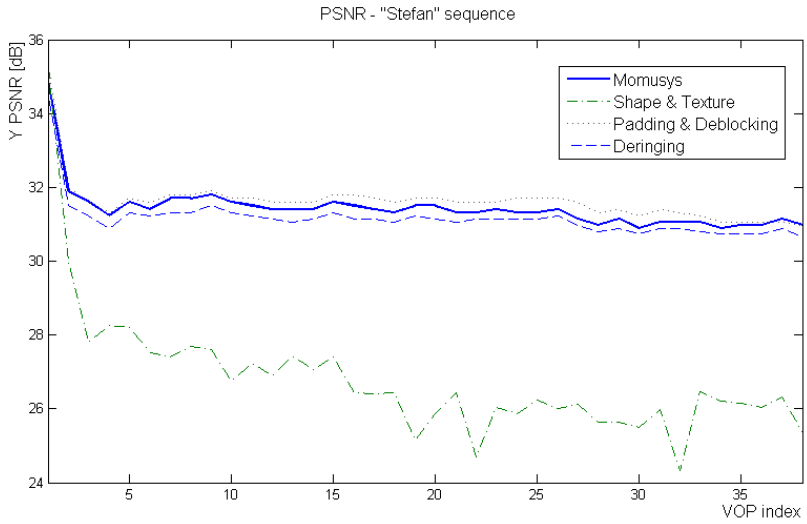


Figure 6.6: *PSNR of "Stefan" sequence at CCIR-601 resolution decoded at different quality levels and by the referenced MoMuSys decoder.*

Figure 6.6 portrays the resulting quality of the MoMuSys decoder [5] without mapping. The quality levels are based on adding an increasing amount of post-processing, such as deblocking and deringing filters. The bold line in Figure 6.6 represents the reconstructed quality of the MoMuSys decoder with enabled deblocking and deringing filters at an average PSNR of 31.43 dB.

The decoding of our experimental decoder without padding and postprocessing filters results in an average PSNR of 26.41 dB (see the bottom line). The experimental decoder with enabled padding and deblocking filter results in an average PSNR of 31.63 dB (dotted line) and the full decoding algorithm provides an average PSNR of 31.17 dB. Note that the deringing filter decreases the image quality with about 0.4 dB. However, this is typical for this type of smoothing filter, but the perceptual quality is higher even when the PSNR is lower.[6]

The next measurement involves the influence of enabling best-effort computing at the decoder. The default operation is reservation-based, but for some of the VOPs we employ the best-effort computing when it is possible given the

[5] The MoMuSys decoder is the result of the European project with the same name in which MPEG-4 object-oriented coding was studied at the time of standardization.

[6] The PSNR is a commonly accepted image quality measurement in coding experiments, but for filtered images this pixel-based measurement is not accurately reflecting a human perception.

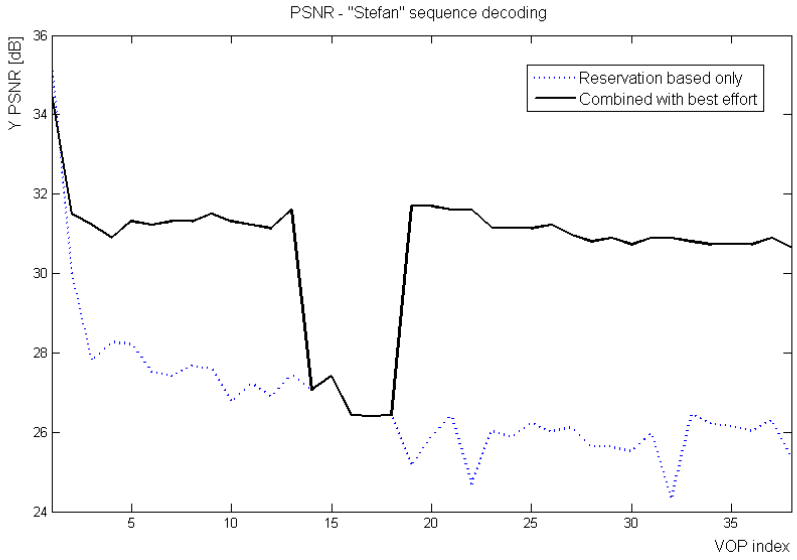


Figure 6.7: *PSNR of "Stefan" sequence decoded at the reservation system and with enabled best-effort computing.*

resource limitations (see the related Appendix). The Figure is explained from the top to the bottom. It would be possible in the reservation-based approach to assign Quality Level 0 for the whole GOV. Due to the insufficient amount of communication resources for VOPs 13-22 (as indicated in Figure 6.4), higher quality levels than Level 0 cannot be assigned for the whole GOV reservation period. Our combined system involving also BE services is decoding only VOPs 14–18 at Quality Level 0. Quality Level 1 is achieved for VOP 13 in the transition towards Quality Level 2 for lower VOP indexes and the interval 19–22 in the opposite transition to higher VOP indexes. Consequently, Quality Level 2 is taken for VOPs from 1–12 and 23–38.

The obtained PSNR of the original and the combined approach is depicted in Figure 6.7, where the dotted line represents the reservation-based approach and the solid line represents the combined solution. The gain in the quality obtained by our new combined approach is illustrated in Figure 6.8. The maximal difference in the PSNR is 6.89 dB and the average increase is 3.86 dB. This quality difference is clearly noticeable by an inexperienced viewer. A visual example of such a quality difference can be seen in Figure 5.3. Given the size of the quality difference, a switching effect occurs in the reconstructed video sequence. This effect is clearly visible for VOPs in the range 10-20. This can be combated by applying a scalable decoding algorithm such as MPEG-4

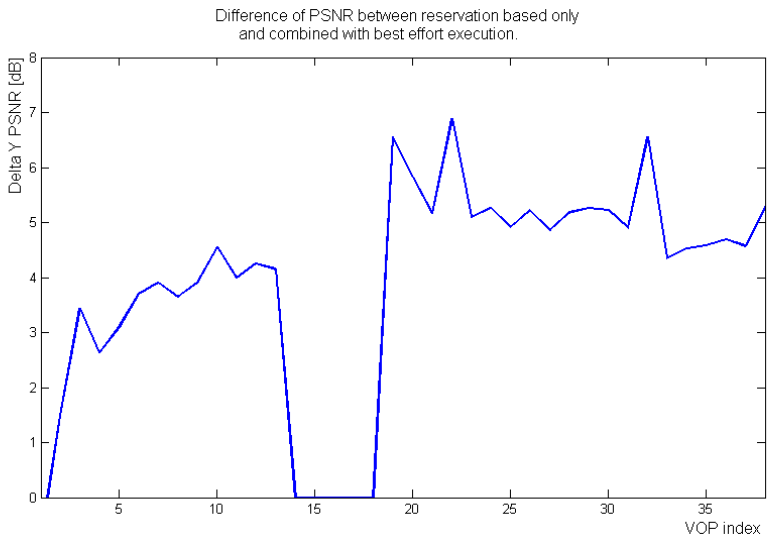


Figure 6.8: *PSNR gain per VOP of the combined system using both reservation-based and best-effort principles.*

FGS. However, the usefulness of this profile was already proven in other work and it was beyond the scope of this thesis.

The communication monitoring typically introduces communication overhead that is orders of magnitude lower than the required bandwidth of the experimental multimedia application. It should be noted that the obtained time fraction where the quality level is higher than with decoding using the reservation-based approach only, is highly dependent on the video input data, the length of the reservation period and the runtime status of the platform. Given the limitations of the design space in this experiment, we comment on the outcome in the following conclusions.

6.6 Conclusions

We have experimentally tested the proposed combination of a reservation-based and the best-effort approach for controlling communication resources. The experiment was based on the mapping of AS VO MPEG-4 decoding on a multiprocessor NoC. Since the amount of objects is unknown in advance and the decoding characteristics are highly variable in resource usage, the execution should have guarantees at least on decoding at the lowest quality. The higher quality levels were achieved by adding the best-effort tasks to the reservation-based processing at the lowest quality. Coupled to these decisions,

we have used best-effort communication connections instead of the initialized guaranteed-throughput connections, where it was required.

Quality-of-Service in the target architecture is facilitated by link probes in all Network Interfaces, so that bandwidth usage can be measured. The monitored data are collected by the Monitoring Service that is connected to the Local QoS unit. The Local QoS units are controlled by the Global QoS unit as indicated in the previous chapter.

The complete system was experimentally verified with a network of eight ARM processor cores, using an MPEG-4 Video Object decoder running with the ACE profile and at CCIR-601 resolution. The proposed framework has shown that the adaptation of bandwidth at the VOP level within a GOV reservation period, can improve the image quality significantly. We derived with a bitrate setting yielding a good quality with an absolute PSNR of approximately 35 dB. The quality improvement using best-effort control of bandwidth was about 4 dB on the average.

The main conclusion of this chapter is that by using a combined approach of reservation-based QoS control with guaranteed communication and best-effort communication for quality enhancements, allows to nearly fully use all bandwidth resources of the target platform. In this way, we are able to reach quality levels that are higher than in the case of only reservation-based control. The experiments have shown that the video quality enhancement resulting from this strategy can be quite substantial. Another system aspect of the followed approach is that the reaction time of a coding application becomes shorter on changes in tasks and jobs. On the short term, this can be positive (tasks are completed) and negative (new guaranteed tasks are added). On the longer term, the average quality of complete applications is always outperforming the quality reservation-based processing only.

It is good to consider that the above results were obtained with serious limitations of the possible design space. We have assumed a one task per processor mapping, the absence of computing constraints and the lack of an operating system that would be suited for parallel multitasking on a multiprocessor system. With these limitations in mind, the results in this chapter are a contributing step in the path to a more complete solution with less constraints. The next step in further research would therefore be the application of best-effort techniques to both computation and bandwidth control to find a more balanced QoS.

Chapter 7

Conclusions

The final concluding chapter of this thesis starts with an overview of the individual chapters. As a representative example of using various techniques of this thesis, we present a mobile application of the AS VO MPEG-4 decoding in which scalability played an important role. The third section of this chapter attempts to define the primary conclusions of this research work. Finally, some recommendations for future work are given.

7.1 Chapter conclusions

In this thesis, we have explored the mapping of an advanced multimedia application such as Arbitrary Shaped (AS) Video Object (VO) MPEG-4 decoding on a network of processors. The work presented in this thesis has contributed to a larger embedded-systems research project[1] aiming at the mapping of applications onto multiprocessor platforms, like Network-on-Chip (NoC). In the research, we have mainly focused on the upper design layers, dealing with the application and its control for an efficient execution. The aspects addressed for the mapping are performance modeling of the MPEG-4 decoding, granularity optimization of the decoding algorithm, task-level scalability and Quality-of-Service (QoS). Let us now summarize the outcomes of the individual chapters.

Chapter 1 has defined the scope of this thesis and has introduced the research problems caused by the the desired execution of object-based decoding on a

[1]The PROGRESS program of the Dutch Technology Foundation STW under the PreMaDoNa project EES.6390.

multiprocessor System-on-Chip. We have divided the problem statement into four parts: (1) partitioning of the application to facilitate pipelined execution, (2) modeling of the performance, (3) introduction of scalability in the algorithm to support Quality-of-Service control, and (4) the usage of two control principles for efficient bandwidth control on an NoC. The chapter contributions were all published at international peer-reviewed conferences, journals and a published book chapter.

Chapter 2 serves as a reference chapter for the details of the AS VO MPEG-4 decoding and presents a brief introduction to the network-based multiprocessor. It can be concluded from the extensive description of AS VO MPEG-4 decoding, that the conventional DCT coding techniques from MPEG-1/2 are extended with the coding of object shapes. As a further extension, specific processing in the form of padding and block-based filtering is added to improve the quality of the object borders. At the system level, the AS VO MPEG-4 decoding allows the designer to think in individual planes and objects that together make up the scene composition. To support the research in this thesis, a standard-compliant and validated AS VO MPEG-4 decoder was developed, which was new at the time of the research start. The second part of the chapter provides the characteristics of the Network-on-Chip as the target platform. The platform should be able to handle the features of MPEG-4 decoding: the combination of high-level control-driven operations and streaming-oriented processing at the video-data level. We have proposed a specific modification of the NoC to support QoS control in the form of network probes for runtime monitoring. The platform features a tile-based computing network, in which each tile is separated from the network by buffered communication. This allows multiple instantiations of objects, each having its own size and dynamics. Finally, we have presented two experimental setups, using clock-cycle-true implementations of the Æthereal NoC and the CELL processor.

Chapter 3 has introduced a data-flow model for describing an application as a set of communicating actors. This model supports the distributed computing over a plurality of processors and pipelined execution for high efficiency. The research result of the chapter is the extension of the Synchronous Data Flow graph (SDF) by a linear parametrical model of required computation resources. The linear equations in the model are based on the important coding parameters of the input stream (BAB-type of the block, number of non-transparent sub-blocks, number of AC coefficients coded by an ESC code, etc.) and weighting coefficients that depend on the target processor architecture. Similarly, we have proposed a parametrical bandwidth-usage model for the communication resources. It was found that our obtained parametrical timing model has only 5.31% deviation from the real execution on an Æthereal NoC with ARM7TDMI cores. The comparison with the mostly used worst-case approach for commu-

nication resource allocation revealed that it saves a factor of 2.5 of worst-case allocated resources. Both parametrical models are important for the design phase exploration and are later used for verifying the possibility of resource-usage prediction with such models.

Chapter 4 concentrates on improving the partitioning of tasks in order to obtain an improved execution performance. To this end, we have provided an algorithmic modification of the MPEG-4 padding tasks in order to shorten the critical path of the decoding algorithm and to unify the processing granularity in the decoding process. By changing the granularity to macroblock level and introducing a new synchronization mechanism that is aware of having sufficient data for processing, we have obtained an execution having only 12.2% (for extended padding) and 40.9% (for repetitive padding) of the original algorithm execution cycles. The unified macroblock processing granularity avoids frame buffers between individual processing stages and simplifies the mapping on tile-based NoC. In the second part, the redesign of the background sprite algorithm was addressed for usage on a CELL processor. By introducing a random-access feature to the compressed data, a large sprite can be segmented into smaller parts that can be individually processed without full decoding. This approach in data parallelism also facilitates distributed processing over various cores.

Chapter 5 presents our hierarchical Quality-of-Service. To serve scalable execution, we have classified tasks composing the AS VO MPEG-4 decoding into two classes. The first class contains essential tasks that cannot be skipped, while the second class is filled with the enhancement functions removing coding artifacts (e.g. blockiness). Scalability of AS VO MPEG-4 decoding was obtained by enabling/disabling functions of the non-essential tasks next to the essential tasks. The hierarchical QoS control is based on an application-specific Local QoS manager and a Global QoS manager responsible for the overall system control. In our experimental implementation, the Local QoS provides the estimation of the resource usage of an application and monitors the real execution. The Global QoS selects the best quality levels of the active applications and reserves resources for the application. The key contribution of this chapter is that we have defined a heuristic algorithm that searches suitable combinations of quality levels for individual jobs, so that a set of jobs that can be mapped on the available resources and provide the highest benefit for the end user. The heuristic is based on keeping a sorted list of the quality degradation per job, resulting in the highest increase of available resources.

Chapter 6 addresses the combination of reservation-based QoS and best-effort principles. This combination was studied with respect to bandwidth control. In order to monitor the bandwidth usage, the network interfaces were enhanced

with monitoring probes and the employed bandwidth was communicated to the Local QoS control using an Monitoring Service Access point. The task-level scalability of Chapter 5 was reused, but now for the combination of reservation-based and best-effort bandwidth control. The presented experimental study has shown that the usage of reservation-based QoS only results in the decoding at a significantly lower quality level. The reservation-based approach guarantees that the VO will be always decoded at least at the lowest quality level, the best-effort computing improves the quality by using the resources as much as they are available. The availability is controlled by the Global QoS system which is the only unit that knows which applications are executing in parallel.

7.2 Evaluation of AS VO MPEG-4 computation complexity

Table 7.1 provides an overview of the different coding standards and relation of the AS VO MPEG-4 processing tasks to the tasks of the other standards. The data for the MPEG-2 encoder listed in the first column are taken from [74] and were measured on a Pentium processor. The second column in the table is taken from [10], where the computation was measured on a DSP type of processor. The data in the third column are relative to the gate counts of a dedicated chip solution for H.264 decoding [22].

The table shows that several tasks are similar in all coding systems, such as motion compensation and IDCT/DCT processing. Each of the realizations has one significant task that is clearly more complex than the other tasks. It can be seen that Context Arithmetic Decoding (CAD) is the most complex task of AS VO MPEG-4 decoding and it is also unique compare to other standards. The summation of shape-related processing as covered by CAD and Shape MC covers more than 40% of the total decoding complexity. Since this is unique, our estimate is that the AS VO MPEG-4 decoder has the same order of magnitude of complexity as an H.264 decoder. This make the experiments with the proposed decoder realistic and representative for processor system design analysis.

7.3 Example application of presented work

This section summarizes a test mapping which was based on some of the proposed techniques in this thesis. The test contributed to a European research project, called Space4U, where design-time performance prediction was investigated for component-based software engineering. The test was carried out at the end of the Space4U research project in 2005.

MPEG2 encoder Pentium	Rel. compl.	MPEG-4 Simp.Prof. TI320C80	Rel. compl.	H.264 decoder	Rel. compl.	AS VO MPEG-4 ARM7	Rel. compl.
				Main control	10.0%	MB Type Dec	0.27%
						Shape MC	12.33%
Motion Est.	32%					CAD	28.89%
						CBP	0.18%
						MvD	1.31%
Quant	10%	Quant	4%			IP&IQ	4.81%
VLC	18%	VLD, Parse	38%	Parser-CABAC	9.7%	Coeff Dec	2.56%
MC	8%	MC	29%	Intra, Inter pred.	13.2% 32.0%	Text MC	10.90%
DCT	14%	IDCT	29%	IQ/IT	9.1%	IDCT	16.29%
						Rep. pad.	10.78%
				Deblocking	16.3%	Ext./bd. padding	11.67%
Others	18%				9.7%		

Table 7.1: *Comparison of different standards and the relative task complexities within AS VO MPEG-4 decoding.*

The experimental setup was as follows. The employed AS VO MPEG-4 decoder was ported to the clock-cycle-true simulator of the ARM7TDMI processor[2] that was used to construct the NoC. This resulted in obtaining realistic data on the execution of individual components on such a processor. The obtained data were used for the estimation of the resource usage and applied as a prediction for the performance evaluation tools.

The AS VO MPEG-4 decoder and the parametrical model developed in Chapter 3 were used by the Space4U project to perform the design-time performance prediction. The scientific results were presented in [13] and [14], with, on the average, a 90% accuracy in predicting the real-time MPEG-4 decoder behavior. As a spinoff of the joint work with the Space4U project, we have ported the AS VO MPEG-4 decoder to a so-called iPaq PDA. Figure 7.1 portrays the PDA executing the MPEG-4 decoder (decoding of the "Singer" sequence at sub-second frame rate with non-optimized code). To our knowledge, this is the first implementation of an AS VO MPEG-4 decoder on a commercially

[2] This is a very popular embedded processor for mobile applications.

Figure 7.1: *Executed MPEG-4 AS VO decoder on iPaq with ARM 400 MHz.*

available portable device. Recently, another example has been published that deals with content-based media processing on a mobile PDA [105].

7.4 Conclusions on research contributions

In this section, we summarize the key elements of the obtained results from this thesis. An efficient mapping of advanced multimedia applications onto an NoC requires that the dynamic resource usage in such applications can be tracked and used to control this mapping. For this purpose, the resource usage needs to be measured and the measurements can be used for prediction of the behavior of the system. Secondly, the word *efficient* reflects that resources are used fully, irrespective of the amount of applications running in parallel, or the amount of processors and network features in the NoC. This has motivated the study of a hierarchical layered Quality-of-Service control. Given the boundaries and setting of the above, the principal contributions of this thesis are threefold.

1. Linear parametrical performance model
The presented linear parametrical model is basically a set of linear equations using coding parameters and weighting coefficients based on the computing target processor features. This representation has proven to be a simple and effective means to provide an accurate estimate of the required performance.

This setup was experimentally verified for computing and bandwidth usage. It goes without saying that the computation of a set of linear equations is a relatively simple task for a processor and it thus poses little overhead for the system. There is little or no work in literature that has been carried out at the same level of detail (clock-cycle-true simulation) and accuracy for a modern multimedia coding application.

2. Hierarchical QoS
This thesis combines multiple video-object decoding applications and a multiprocessor platform in one problem statement[3]. For this problem statement, and to provide system compositionality, we have separated one application from the other and on top distinguish a Global overview of the complete set of applications running in parallel. This vision has resulted in the presented hierarchical QoS system. The hierarchy in QoS was known, but its application on a multiprocessor is novel. To support QoS-based execution of the multimedia application, task-level scalability has been presented. The author claims that these techniques can be more generically applied to a multitude of audiovisual applications going far beyond the presented MPEG-4 decoding test case. The provided overview of other related coding standards has shown that there is a substantial commonality between such standards and that the complexity of our decoder is representative for other modern systems.

3. Best-effort combined with reservation-based bandwidth control
The main conclusion of this part of the work is that by using a combined approach of reservation-based QoS control with guaranteed communication and best-effort communication for quality enhancements, allows to nearly fully use all bandwidth resources of the target platform. In this way, we are able to reach quality levels that are substantially higher than in the case of only reservation-based control. These results were obtained with serious limitations of the possible design space. We have assumed the absence of computing constraints and the lack of an operating system that would be suited for parallel multitasking on a multiprocessor system. With these limitations in mind, the results of this work are a contributing step in the path to a more complete solution for multiple task execution on a multiprocessor system. The next step in further research would therefore be the application of best-effort techniques to both computation and bandwidth control to find a more balanced QoS.

[3]This problem statement is certainly a compelling one, and with the trend towards multiprocessor systems, it will remain so for the upcoming period, not in the least because the optimal software design recipe for multiprocessor systems is not yet known.

7.5 Future work

We have identified the following aspects for further investigation.

1. Heterogenous networks.
The experiments in this thesis are typically based on a network of ARM7 processors or the CELL processor. These NoCs are essentially homogeneous processor networks. It is known from earlier work that the use of application-specific processors in combination with general-purpose processors increases the efficiency of the computing platform for many video processing applications. Therefore, the incorporation of special coprocessors fitting within the DFG network, is a natural next step in this research work.

2. Semi-automatic tools for distributed mapping.
The mapping on the processor network presented in this thesis was performed manually. As already indicated, there is no optimal algorithm for mapping advanced multimedia applications on a multiprocessor system. Given the trend towards an increased used of multiprocessors, tools to help the architects and designers in solving complex mapping problems would be no luxury and probably will become indispensable in the coming years. It will take some time before the knowledge of experienced architects can be found in a software tool; therefore semi-automatic tools can be interesting to explore.

3. Dynamic mapping.
In this thesis, the application mapping was assumed to be static, but in more advanced systems this may not be valid. It will be interesting to see how the proposed efficiency measures work out for dynamic changes in the application. For example, the system may hop from one DFG to another in a certain time period. To maintain mapping efficiency during such a transition is a challenge of its own.

Appendix A

Visual bitstream structure

Coded visual data consists of several different types, such as video data, 2D mesh data or facial animation parameters or user defined data. A *visual object sequence* is the highest syntactic structure of the coded visual bitstream. The visual objects sequence is bounded with a unique codes called *visual_object_sequence_start_code* and *visual_object_sequence_end_code*. One visual sequence contains one or more concurrently coded visual objects. The start and the end codes are unique codes in the bitstream used for identifying some of the structures in the coding syntax The start code of a

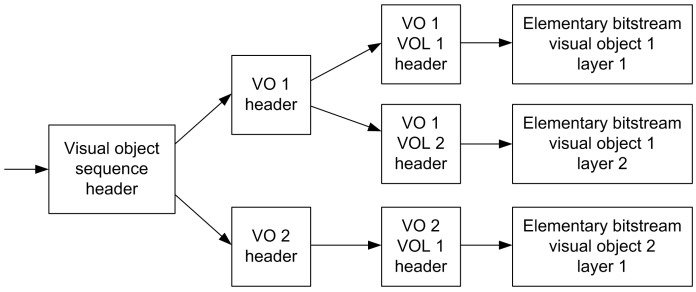


Figure A.1: *Example of the logical structure of the configuration end elementary stream data.*

visual object is followed by a profile and level information, and a visual object id. These configuration data are followed by a *video object*, a still texture object, a mesh object or a facial/body animation object. The start code of the videal objects is followed by one or more *video object layers*. The syntax for

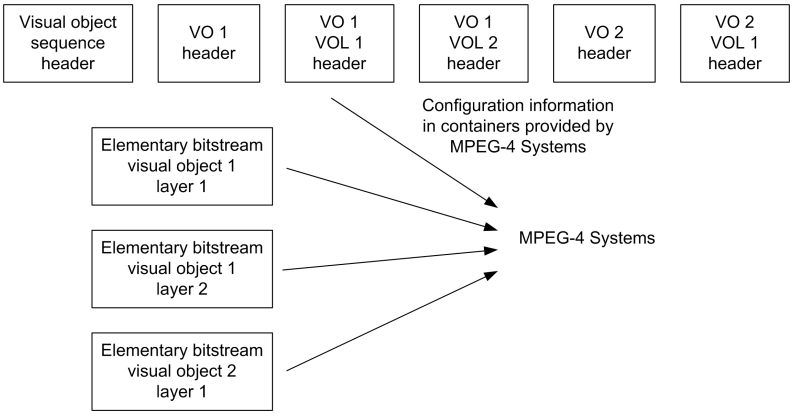


Figure A.2: *Example of the visual bitstream with separate configuration.*

visual bitstream defines two types of information: configuration information and elementary stream data. The data for a single layer of a visual object are enclosed in the elementary stream data.

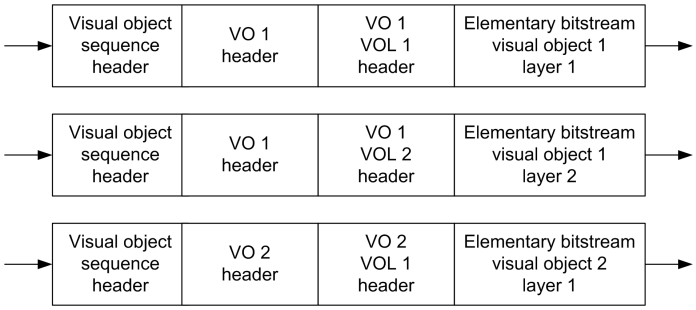


Figure A.3: *Example of the visual bitstream with combined configuration.*

Figure A.1 represents the logical structure of visual information. The ISO/IEC 14496 allows both, the separate or combined coding of configuration and elementary stream data.

1. *Separate Configuration* is defined such that the configuration information are always carried separately as illustrated by Figure A.2. The system specification defines containers that are used to configuration information.

2. *Combined Configuration* associates the elementary stream data with at most one instance of each of Visual Object Sequence, Visual Object and Video Object Layer configuration information (see Figure A.3).

Appendix B

Test video sequences

Stefan sprite sequence

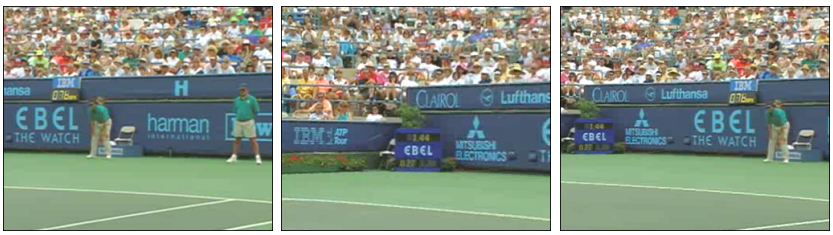


Figure B.1: *The "News" sequence.*

The motion of the original sequence is high due to the nature of sport sequences. The "Stefan" background sequence was used in Chapter 4 for optimizing the local memory buffer for the sprite image. Figure B.1 illustrates three different camera views: a view at the start of the sequence, moving camera to the left and the last camera motion to the original position.

Arbitrary-shaped sequences

Figure B.2 and Figure B.3 illustrate decoded texture and shape information of the sequences used in experiments presented in this thesis. The content of sequences affect the actual size of processed VOPs. For example, sequences visualized in lines 1, 2 and 3 have a high variation in the size compared to sequences in lines 3 and 4.

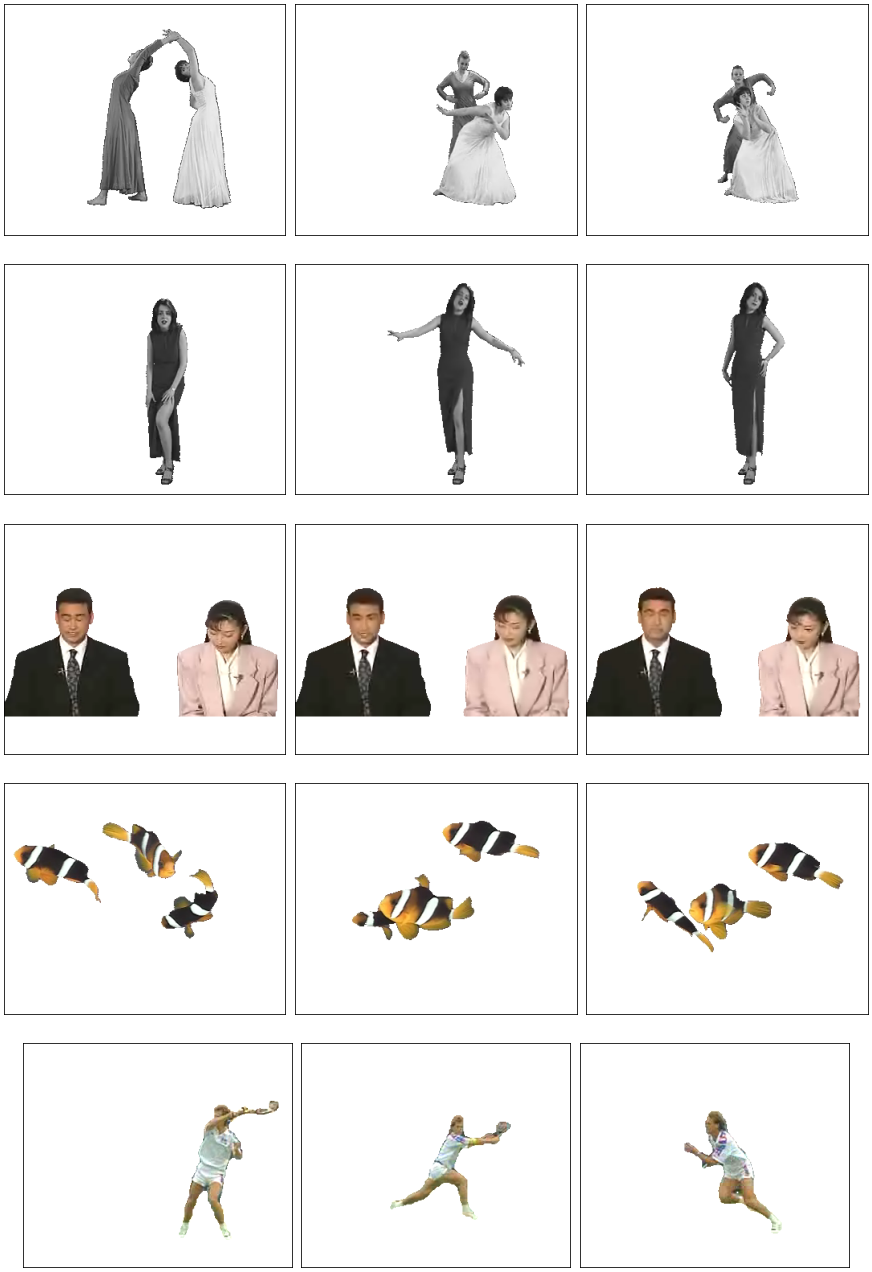


Figure B.2: *Examples of decoded texture information per sequence. From top, "Dancer", "Singer", "News", "Fish", "Stefan" sequences.*

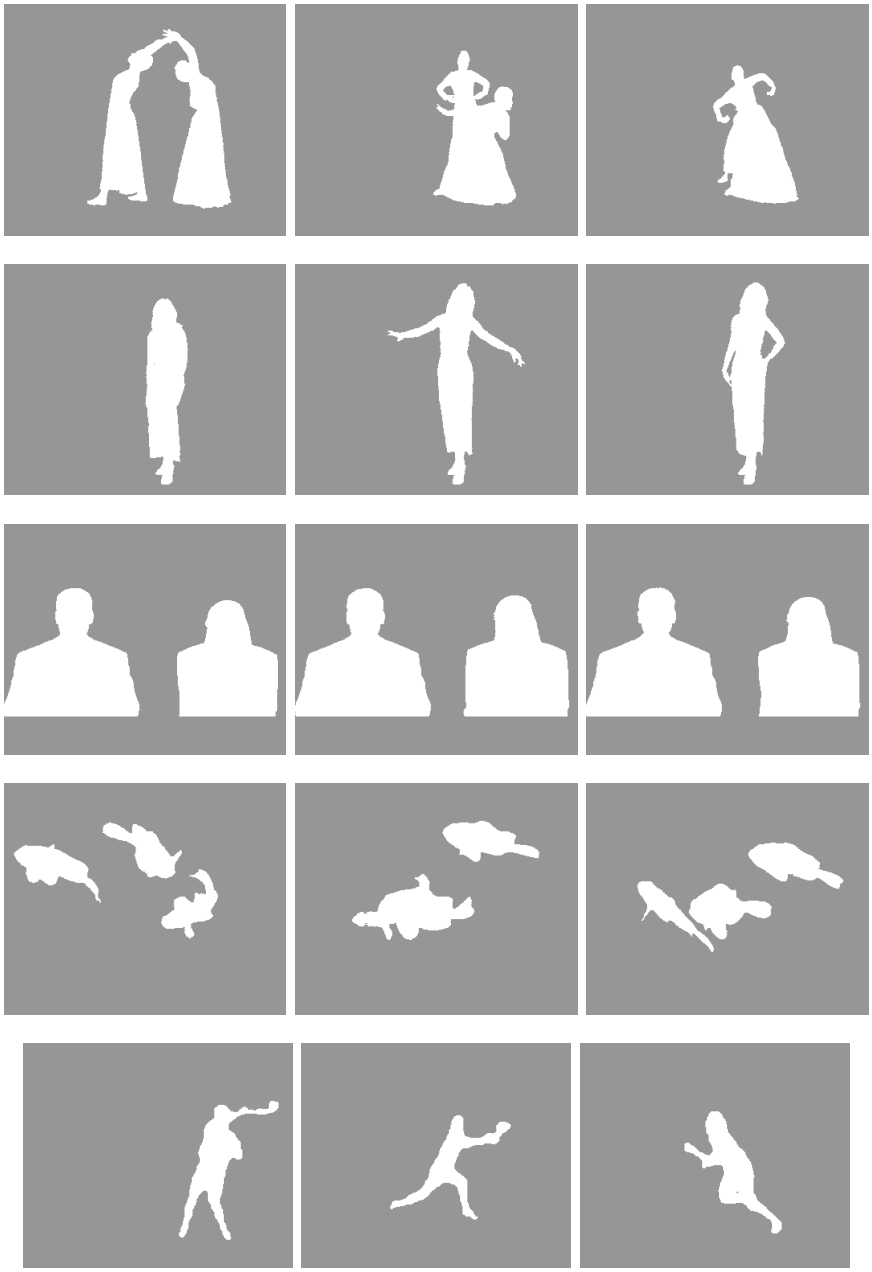


Figure B.3: *Examples of shape information per sequence. From top, "Dancer", "Singer", "News", "Fish", "Stefan" sequences.*

	Dancer	Singer	News	Fish	Stefan	Sprite
VOPs	250	250	300	300	239	239
Frames	25	25	30	30	30	30
Average # MBs	131.6	99.4	224.5	144.8	61.4	378
VOP size range (MB)	77-255	60-204	220-242	112-182	30-110	378
Scene resolution	352x288	352x288	352x288	352x288	336x280	336x280

Table B.1: *Sequences' parameters.*

	Dancer	Singer	News	Fish	Stefan
MB Type	702,546	319,745	791,489	501,535	279,813
Shape MC	32 903,082	16 160,475	50 173,903	17 583,050	7 796,445
CAD	112 774,044	15 738,453	35 811,118	64 708,242	51 755,425
CBP	452,943	216,785	620,471	296,857	172,049
MvD	3 229,157	1 718,802	2 342,352	2 594,366	1 833,053
IP&IQ	12 082,489	5 601,999	17 014,919	8 455,911	4 331,412
Coeff Dec	7 912,108	1 349,719	1 683,062	11 694,274	2 457,035
Text MC	28 914,351	13 090,094	31 621,407	19 008,157	11 840,196
IDCT	41 126,982	19 286,613	58 362,770	26 422,328	15 170,353
Rep. pad.	27 606,203	13 220,757	34 903,745	17 955,571	10 724,965
EBP	28 138,732	13 836,290	34 431,104	22 298,086	12 588,596
Exec. time (kcycles)	295,842	100,539	267,756	191,518	118,949
Processed MBs	2,064	720	2,640	1,520	671
Average exec.	143,334	139,638	101,422	125,998	177,271

Table B.2: *Clock-cycle execution time of individual task for a GOV (CIF).*

Table B.1 provides the parameters of individual sequences. As is indicated in the third line, the average number of required macroblocks per sequence length differs from 61.4 ("Stefan") to 224.5 ("News") macroblocks between various sequences. Further, the dynamism in the object size is also variable, whereas the "News" sequence has 22 MBs variation as compared to the "Dancer" sequence, which has a variation of 178 MBs. Table B.2 lists the measured complexity of decoding one GOV (12 frames) on the target clock-cycle-true simulator. The data are measured in clock cycles. The required complexity per task highly varies per sequence. For example, the CAD task requires 3 times more clock cycles than the Shape MC task for the "Dancer" sequence, while for the "News" sequence, this ratio is only 0.71 times.

References

[1] ARM Developer Suite Version 1.2. *Debug Target Guide*. Ref: DUI0058D, Issued November 2001, http://www.arm.com.

[2] J.H. Anderson, J.M. Calandrino, and U.C. Devi, Real-time scheduling on multicore platforms, *Proc. of 12th IEEE Real-Time and Embedded Tech. and App. Symp.*, ISBN 0-7695-2516-4, Apr. 2006, pp. 179–190.

[3] H. Arakida, M. Takahashi, Y. Tsuboi, T. Nishikawa, *et al.*, A 160 mW, 80 nA standby, MPEG-4 audiovisual LSI with 16 Mb embedded DRAM and a 5 GOPS adaptive post filter, *Proc. of IEEE Int. Solid-State Circuits Conf. (ISSCC)*, ISSN 0193-6530, Feb. 2003, pp. 42–476.

[4] C. Aurrecoechea, A. Campbell, and L. Hauw, A Survey of QoS Architectures, *ACM/Springer Verlag Multimedia Systems Journal*, Ser. 3, Vol. 6, ISSN 0942-4962, May 1998, pp. 138–151.

[5] S. Banachowski, T. Bisson, and S.A. Brandt, Integrating best-effort scheduling into a real-time system, *Proc. of 25th IEEE Int. Real-Time Systems Symp.*, ISBN 0-7695-2247-5, Dec. 2004, pp. 139–150.

[6] A.C. Bavier, A.B. Montz, and L.L. Peterson, Predicting MPEG execution times, *Proc. of SIGMETRICS/PERFORMANCE Int. Conf. on Measurement and Modeling of Computer Systems*, ISBN 0-89791-982-3, June 1998, pp. 131–140.

[7] M. Bekooij, R. Hoes, O. Moreira, P. Poplavko, M. Pastrnak, *et al.*, Chapter 5: Dataflow Analysis for Real-time Embedded Multiprocessor System Design, in P. van der Stock (Ed.), *Dynamic and Robust Streaming in and between Connected Consumer-Electronic Devices*, Springer, Dordrecht, ISBN 1-4020-3453-9, May 2005, pp. 81–108.

[8] M. Bekooij, O. Moreira, P. Poplavko, B. Mesman, M. Pastrnak, *et al.*, Predictable Embedded Multiprocessor System Design, *Proc. of 8th Int.*

Work. on Software and Compilers for Embedded Systems (SCOPES), ISBN 3-540-23035-1, Sept. 2004, pp. 77–91.

[9] L. Benini and G. De Michelli, Networks on chips: A new SoC paradigm, *IEEE Computer*, Ser. 1, Vol. 35, ISSN 0018-9162, Jan. 2002, pp. 70–80.

[10] M. Berekovic, H.-J. Stolberg, M.B. Kulaczewski, P. Pirsch, H. Möller, *et al.*, Instruction Set Extensions for MPEG-4 Video, *Journal of VLSI Signal Processing*, Ser. 1, Vol. 23, ISSN 0922-5773, Oct. 1999, pp. 27–49.

[11] M. Berekovic, H.-J. Stollberg, and P. Pirsch, Multicore System-On-Chip Architecture for MPEG-4 Streaming Video, *IEEE Trans. on Circuits and Systems for Video Technology*, Ser. 8, Vol. 12, ISSN 1051-8215, Aug. 2002, pp. 688–699.

[12] T. Bjerregaard and J. Spars, A Router Architecture for Connection-Oriented Service Guarantees in the MANGO Clockless Network-on-Chip, *Proc. of the Design, Automation and Test in Europe Conference (DATE)*, ISBN 0-7695-2288-2, Mar. 2005, pp. 1226–1231.

[13] E. Bondarev, M. Pastrnak, P.H.N. de With, and M.R.V. Chaudron, On Design-Time Performance Predictions of Object-Based MPEG-4 Video Applications, *Proc. of DSP Valley's Annual Research and Tech. Symp. (DARTS) and Signal Proc. Symp. (SPS)*, Apr. 2005, pp. 19–22.

[14] E. Bondarev, M. Pastrnak, P.H.N. de With, and M.R.V. Chaudron, Predictable Component-Based Software Design of Real-Time MPEG-4 Video Applications, *Proc. of Visual Communications and Image Processing (VCIP)*, July 2005, pp. 2288–2298.

[15] J. Bormans, N.P. Ngoc, G. Deconinck, and G. Lafruit, Chapter: Terminal QoS: advanced resource management for cost-effective multimedia appliances in dynamic contexts, in T. Basten, M. Geilen, and H. de Groot (Ed.), *Ambient intelligence: impact on embedded system design*, Kluwer Academic Publ., NL., ISBN 978-1-4020-7668-8, Jan. 2003, pp. 183–201.

[16] N. Brady, MPEG-4 Standardized Methods for the Compression of Arbitrarily Shaped Video Objects, *IEEE Trans. on Circuits and Systems for Video Tech.*, Ser. 8, Vol. 9, ISSN 1051-8215, Dec. 1999, pp. 1170–1189.

[17] B.B. Brandenburg and J.H. Anderson, Integrating Hard/Soft Real-Time Tasks and Best-Effort Jobs on Multiprocessors, *Proc. of the 19th Euromicro Conference on Real-Time Systems*, ISBN 0-7695-2914-3, July 2007, pp. 61–70.

[18] S. Brandt, G. Nutt, T. Berk, and M. Humphrey, Soft Real-time Application Execution with Dynamic Quality of Service Assurance, *Proc. of 6th IEEE/IFIP Workshop on Quality of Service*, May 1998, pp. 154–163.

[19] R.J. Bril, *Real-time scheduling for media processing using conditionally guaranteed budgets*, PhD thesis, University of Technology Eindhoven, ISBN 90-74445-62-4, Sept. 2004.

[20] L.-O. Burchard and P. Altenbernd, Estimating decoding times of MPEG-2 video streams, *Proc. of IEEE Int. Conf. on Image Processing*, ISBN 0-7803-6297-7, Sept. 2000, pp. 560–563.

[21] P. Chandra, A. Fisher, C. Kosak, and P. Steenkiste, Network Support for Application-Oriented Quality of Service, *Proc. of 6th IEEE/IFIP Work. on Quality of Service*, May 1998, pp. 187–195.

[22] Ch.-Jr Chen, T.-C. Lian and L.-G. Chen, Hardware Architecture Design of an H.264/AVC Video Codec, *Proc. of the 2006 Conf. on Asia South Pacific design automation*, ISBN 0-7803-9451-8, Jan. 2006, pp. 750–757.

[23] Y. Chen, Z. Zhong, T.-H. Lan, S. Peng, and K. van Zon, Complexity Scalable MPEG-2 Video Decoding for Media Processors, *IEEE Trans. on Circuits and Systems for Video Technology*, Ser. 8, Vol. 12, ISSN 1051-8215, Aug. 2002, pp. 678–687.

[24] C. Ciordas, T. Basten, A. Radulescu, K. Goossens, *et al.*, An event-based monitoring service for Network-on-Chip, *ACM Trans. on Design Automation of Eletronic Systems*, Ser. 4, Vol. 10, 2005, pp. 702–723.

[25] P. Cumming, Chapter 5: The TI OMAP platform approach to SoC, in G. Martin and H. Chang (Ed.), *Winning the SoC Revolution*, Kluwer Academic Publishers, ISBN 1-4020-7495-6, June 2003, pp. 97–118.

[26] W. J. Dally and B. Towles, Route packets, not wires: on-chip interconnection networks, *Proc. of 38th Design Automation Conference (DAC)*, ISBN 1-58113-297-2, June 2001, pp. 684–689.

[27] G. De Micheli and L. Benini, *Networks on Chips: Technology and Tools*, Morgan Kaufmann, San Francisco, ISBN 0-12-370521-5, July 2006.

[28] J.A. de Oliveira and H. van Antwerpen, Chapter 4: The Philips NEXPERIA digital video platform, in G. Martin and H. Chang (Ed.), *Winning the SoC Revolution*, Kluwer Academic Publishers, ISBN 1-4020-7495-6, June 2003, pp. 67–96.

[29] P.H.N. de With and E.G.T. Jaspers, Design of multimedia software and future system architectures, *Proc. of the SPIE Embedded Processors for Multimedia and Communications*, Vol. 5309, Jan. 2004, pp. 58–69.

[30] K. Denolf, *Low Power Design of Block-Based Video Codecs*, PhD thesis, University of Technology Eindhoven, June 2007.

[31] K. Denolf, C. De Vleeschouwer, R. Turney, G. Lafruit, and J. Bormans, Memory centric design of an MPEG-4 video encoder, *IEEE Trans. on Circuits and Systems for Video Technology*, Ser. 5, Vol. 15, ISSN 1051-8215, May 2005, pp. 609–619.

[32] S.H. Dhong, O. Takahashi, M. White, T. Asano, *et al.*, A 4.8 GHz Fully Pipelined Embedded SRAM in the Streaming Processor of a CELL Processor, *Proc. of IEEE Int. Solid-State Circuits Conf. (ISSCC)*, ISSN 0193-6530, Feb. 2005, pp. 134–135.

[33] R. Dugad and N. Ahuja, A scheme for spatial scalability using nonscalable encoders, *IEEE Trans. on Circuits and Systems for Video Technology*, Ser. 10, Vol. 13, ISSN 1051-8215, Oct. 2003, pp. 993–999.

[34] A. Dutta, R. Jensen, and A. Rieckmann, Viper: A multiprocessor SoC for advanced set-top box and digital TV systems, *IEEE Design and Test of Computers*, Ser. 5, Vol. 18, ISSN 0740-7475, Sept. 2001, pp. 21–31.

[35] H.-C. Fang, Parallel Embedded Block Coding Architecture for JPEG 2000, *IEEE Trans. on Circuits and Systems for Video Technology*, Ser. 9, Vol. 15, ISSN 1051-8215, Sept. 2005, pp. 1086–1097.

[36] D. Farin, P.H.N. de With, and W. Effelsberg, Minimizing MPEG-4 Sprite Coding-Cost Using Multi-Sprites, *Proc. of SPIE Visual Commun. and Image Proc.*, Vol. 5308/1, ISBN 0-8194-5211-4, Jan. 2004, pp. 234–245.

[37] D.S. Farin, *Automatic Video Segmentation Employing Object/Camera Modeling Techniques*, PhD thesis, University of Technology Eindhoven, ISBN 90-386-2381-X, Dec. 2005.

[38] B. Flachs, S. Asano, S.H. Dhong, P. Hotstee, *et al.*, A Streaming Processing Unit for a CELL Processor, *Proc. of IEEE Int. Solid-State Circuits Conference (ISSCC)*, ISSN 0193-6530, Feb. 2005, pp. 134–135.

[39] P.J. Fortier and H.E. Michel, *Computer Systems Performance Evaluation and Prediction*, Digital Press, ISBN 978-1-55558-260-9, June 2003.

[40] E. Francois and J. Vieron, Extended Spatial Scalability : A Generalization of Spatial Scalability for Non Dyadic Configurations, *Proc. of IEEE Int. Conf. on Image Processing*, ISSN 1522-4880, Oct. 2006, pp. 169–172.

[41] S.V. Gheorghita, *Dealing with Dynamism in Embedded System Design*, PhD thesis, TU Eindhoven, ISBN 978-90-386-1644-5, Dec. 2007.

[42] K. Goossens, J. Dielissen, *et al.*, The Æthereal network on chip: Concepts, architectures, and implementations, *IEEE Design and Test of Computers*, Ser. 5, Vol. 22, ISSN 0740-7475, Sept. 2005, pp. 21–31.

[43] K. Goossens, J. van Meerbergen, A. Peeters, and P. Wielage, Networks on silicon: Combining best-effort and guaranteed services, *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*, ISBN 0-7695-1471-5, Mar. 2002, pp. 423–425.

[44] P. Guerrier and A. Greiner, A generic architecture for on-chip packet-switched interconnections, *Proc. of Design, Automation and Test in Europe Conference and Exhibition (DATE)*, ISBN 1-58113-244-1, Mar. 2000, pp. 250–256.

[45] J. Gustafsson, B. Lisper, R. Kirner, and P. Puschner, Input-Dependency Analysis for Hard Real-Time Software, *Proc. of 9th IEEE Int. Conf. on Object-oriented Real-time Dependable Systems*, ISBN 0-1795-2054-5, Oct. 2003, pp. 53–60.

[46] A. Hansson, K. Goossens, *et al.* A unified approach to mapping and routing on a network on chip for both best-effort and guaranteed service traffic. *VLSI Design*, 2007:Article ID 68432, 16 pages, 2007.

[47] C. Hentschel, M. Gabrani, K. van Zon, R.J. Bril, and L. Steffen, Scalable Video Alghorithms and Quality-of-Service Resource Management, *Digest of Technical Papers of IEEE Int. Conf. on Consumer Electronics (ICCE)*, June 2001, pp. 338–339.

[48] P. Hoang and J. Rabaey, Scheduling of DSP Programs onto Multiprocessors for Maximum Throughput, *IEEE Trans. on Signal Processing*, Ser. 6, Vol. 41, ISSN 1053-587X, June 1993, pp. 2225–2235.

[49] J. Huang, P.-J. Wan, and D.-Z. Du, Criticality- and QoS-Based Multiresource Negotiation and Adaptation, *Journal Real-Time Systems*, Ser. 3, Vol. 15, ISSN 0922-6443, Oct. 2004, pp. 249–273.

[50] Ch.J. Hughes, P. Kaul, S.V. Adve, R. Jain, Ch. Park, and J. Srinivasan, Variability in the execution of multimedia applications and implications for architecture, *Proc. of 28th Int. Symp. on Computer Architecture*, ISSN 0163-5964, July 2001, pp. 254–265.

[51] ISO/IEC 14496-11, *Coding of audio-visual objects, Part 11: Scene description and Application engine (BIFS, XMT, MPEG-J)*, 2005.

[52] ISO/IEC 14496-2:1999/ Amd 1:2000, *Coding of Audio-Visual Objects - Part 2:Visual, Amendement 1: Visual Extensions*, Maui, Dec. 1999.

[53] Jingwen J. and K. Nahrstedt, Qos specification languages for distributed multimedia applications: a survey and taxonomy, *IEEE Multimedia*, Ser. 3, Vol. 11, ISSN 1070-986X, 2004, pp. 74–87.

[54] R. Jain, *The art of computer system performance analysis, Techniques for Experimental Design, Measurement, Simulation and Modeling*, John Wiley & Sons Ltd., ISBN 978-0-471-50336-1, April 1991.

[55] E.G.T. Jaspers, *Architecture design of video processing systems on a chip*, PhD thesis, TU Eindhoven, ISBN 90-74445-57-8, Apr. 2003.

[56] E.G.T. Jaspers, P.H.N. de With, and J.G.W.M. Janssen, A flexible heterogeneous video processor system for TV applications, *IEEE Trans. on Conumer Electronics*, Ser. 1, Vol. 45, ISSN 0098-3063, Feb. 1999, pp. 1–12.

[57] G. Kahn, The semantics of a simple language for parallel programming, *Proc. of IFIP Congress*, 1974, pp. 471–475.

[58] P. Kauff and K. Schuur, Shape-adaptive DCT with block-based DC separation and DC correction, *IEEE Trans. on Circuits and Systems for Video Tech.*, Ser. 3, Vol. 8, ISSN 1051-8215, June 1998, pp. 237–242.

[59] L. Kleinrock, *Theory, Volume 1, Queueing Systems*, John Wiley & Sons Ltd., ISBN 0-471-49110-1, April 1975.

[60] R. Koenen, *Overview of the MPEG-4 Standard - (V.15 - Beijing Version)*, WG11 (MPEG), July 2000.

[61] M. Kunter, A. Krutz, M. Droese, M. Frater, and T. Sikora, Object-based multiple sprite coding of unsegmented videos using H.264/AVC, *Proc. of IEEE Int. Conf. on Image Processing (ICIP)*, ISBN 1-4244-1437-7, Sept. 2007, pp. I–65–I–68.

[62] T.-W. Kuo, L.-P. Chang, Y.-H. Liu, and K.J. Lin, Efficient Online Schedulability Tests for Real-Time Systems, *IEEE Trans. on Software Engineering*, Ser. 8, Vol. 29, ISSN 0098-5589, Aug. 2003, pp. 734–751.

[63] G. Lafruit, Nam Pham Ngoc, W. van Raemdonck, N. Tack, and J. Bormans, Terminal QoS for real-time 3D visualization using scalable MPEG-4 coding, *IEEE Trans. on Circuits and Systems for Video Technology*, Ser. 11, Vol. 13, ISSN 1051-8215, Nov. 2003, pp. 1136–1143.

[64] R. Lauwereins, M. Engels, M. Ade, and J. A. Peperstraete, Graphe-II: A System-Level Prototyping Environment for DSP, *IEEE Computer*, Ser. 2, Vol. 28, ISSN 0018-9162, Feb. 1995, pp. 35–43.

[65] C. Lee, J. Lehoczky, D. Siewiorek, R. Rajkumar, and J. Hansen, A scalable solution to the multi-resource QoS problem, ISBN 0-7695-0475-2, 1999, pp. 315–326.

[66] E. A. Lee and D. G. Messerschmitt, Static scheduling of synchronous data flow programs for digital signal processing., *IEEE Trans. on Computers*, Ser. 1, Vol. 36, ISSN 0016-9340, 1987, pp. 24–35.

[67] B. Li, *Agilos: A Middleware Control Architecture for Application-Aware Quality of Service Adaptations*, PhD thesis, University of Illinois at Urbana-Champaign, 2000.

[68] P. Li, B. Veeravalli, and A.A. Kassim, Design and Implementation of Parallel Video Encoding Strategies Using Divisible Load Analysis, *IEEE Trans. on Circuits and Systems for Video Technology*, Ser. 9, Vol. 15, ISSN 1051-8215, Sept. 2005, pp. 1098–1112.

[69] Y.-T. S. Li, S. Malik, and A. Wolfe, Efficient Microarchitecture Modeling and Path Analysis for Real-Time Software, *Proc. of 16th IEEE Real-Time Systems Symposium*, ISBN 0-8186-7337-0, Dec. 1995, pp. 298–307.

[70] T. Liu and J.R. Kender, Computational approaches to temporal sampling of video sequences, *ACM Trans. on Multimedia Computing, Communic., and Applications*, Ser. 2, Vol. 3, ISSN 1551-6857, May 2007, pp. 1–23.

[71] Q. Ma and P. Steenkiste, Quality-of-Service Routing for Traffic with Performance Guarantees, *Proc. of IFIP 5th Int. Work. on Quality of Service*, ISBN 0-4128-0940-0, 1997, pp. 115–126.

[72] S. Martello and P Toth, *Knapsack Problems - Algorithms and Computer Implementations*, John Wiley & Sons Ltd., ISBN 0-471-92420-2, 1990.

[73] S.H. Mian, Analysis of MPEG-4 scalable encoded video, *IEE Proceedings Communications*, Ser. 3, Vol. 151, ISSN 1350-2425, 2004, pp. 270–279.

[74] S.O. Mietens, *Complexity Scalable MPEG Encoding*, PhD thesis, University of Technology Eindhoven, ISBN 90-386-2040-3, Feb. 2004.

[75] M. Millberg, E. Nilsson, R. Thid, S. Kumar, *et al.*, The Nostrum backbone - a communication protocol stack for networks on chip, *Proc. Int. Conf. on VLSI Design.*, ISBN 0-7695-2072-3, Jan. 2004, pp. 693–696.

[76] O. Moreira, J.-D. Mol, M. Bekooij, and J. van Meerbergen, Multiprocessor resource allocation for hard-real-time streaming with a dynamic job-mix, *Proc. of 11th IEEE Real Time and Embedded Technology and Applications Symposium*, ISSN 1080-1812, Mar. 2005, pp. 332–341.

[77] L. Nachtergaele, B. Vanhoof, M. Peón, G. Lafruit, J. Bormans, and I. Bolsens, Implementation of a scalable MPEG-4 wavelet-based visual texture compression system, *Proc. of 36th ACM/IEEE conference on Design automation*, ISBN 1-58133-109-7, June 1999, pp. 333–336.

[78] H. Nakayama, T. Yoshitake, H. Komazaki, Y. Watanabe, H. Araki, *et al.*, An MPEG-4 video LSI with an error-resilient codec core based on a fast motion estimation algorithm, *Proc. of IEEE Int. Solid-State Circuits Conf. (ISSCC)*, ISSN 0193-6530, Feb. 2002, pp. 368–474.

[79] V. Nollet, T. Marescaux, and D. Verkest, Operating-system controlled network on chip, *Proc. of 41st Design Automation Conference*, ISSN 0738-100X, June 2004, pp. 256–259.

[80] C.M. Otero-Perez, L. Steffens, P. van der Stok, S. van Loo, *et al.*, Chapter: QoS-based resource management for ambient intelligence, in T. Basten, M. Geilen, and H. de Groot (Ed.), *Ambient intelligence: impact on embedded system design*, Kluwer Academic Publ., NL., Dordrecht, ISBN 978-1-4020-7668-8, Jan. 2003, pp. 159–182.

[81] M. Pastrnak and P.H.N. de With, Data Storage Exploration and Bandwidth Analysis for Distributed MPEG-4 Decoding, *Proc. of 8th IEEE Int. Symp. on Consumer Electronics (ISCE)*, ISBN 0-7803-8527-6, Sept. 2004, pp. 67–72.

[82] M. Pastrnak and P.H.N. de With, On the Computing Analysis of Arbitrary Shape Coding in MPEG-4, *Proc. of 25th Int. Symp. on Information Theory in the Benelux*, ISBN 90-71048-20-9, June 2004, pp. 193–200.

[83] M. Pastrnak and P.H.N. de With, Multidimensional Model of Estimated Resource Usage for Multimedia NoC QoS, *Proc. of 27th Int. Symp. on Inf. Theory in the Benelux*, ISBN 90-71048-22-5, June 2006, pp. 109–116.

[84] M. Pastrnak, P.H.N. de With, C. Ciordas, J.L. van Meerbergen, and K. Goossens, Mixed Adaptation and Fixed-reservation QoS for Improving Picture Quality and Resource Usage of Multimedia (NoC) Chips, *Proc. of 10th IEEE Int. Symp. on Consumer Electronics (ISCE)*, ISBN 1-4244-0215-8, June 2006, pp. 207–212.

[85] M. Pastrnak, P.H.N. de With, S. Stuijk, and J.L. van Meerbergen, Parallel Implementation of Arbitrary-shaped MPEG-4 Decoder for Multiprocessor Systems, *Proc. of Visual Communications and Image Processing (VCIP)*, ISBN 0-8194-6117-2, Jan. 2006, pp. 60771I–1..60771I–10.

[86] M. Pastrnak, P.H.N. de With, and J.L. van Meerbergen, QoS Concept for Scalable MPEG-4 Video Object Decodidng on Multimedia (NoC) Chips, *IEEE Trans. on Consumer Electronics*, No. 4, Vol. 52, ISSN 0098-3063, Nov. 2006, pp. 1418–1426.

[87] M. Pastrnak, P.H.N. de With, and J.L. van Meerbergen, Realization of QoS Management Using Negotiation Algorithms for Multiprocessor NoC, *Proc. of IEEE Int. Symp. on Circuits and Systems (ISCAS)*, ISBN 0-7803-9390-2, May 2006, pp. 1912–1915.

[88] M. Pastrnak, D.S. Farin, and P.H.N. de With, Adaptive Decoding of MPEG-4 Sprites for Memory-Constrained Embedded Systems, *Proc. of 26th Int. Symp. on Information Theory in the Benelux*, ISBN 90-71048-21-7, May 2005, pp. 137–144.

[89] M. Pastrnak, P. Poplavko, P.H.N. de With, and D.S. Farin, Data-flow timing Models of Dynamic Multimedia Applications for Multiprocessor Systems, *Proc. of 4th IEEE Int. Work. on System-on-Chip for Real-Time Applications (SoCRT)*, ISBN 0-7695-2182-7, July 2004, pp. 206–209.

[90] M. Pastrnak, P. Poplavko, P.H.N. de With, and J.L. van Meerbergen, On Resource Estimation of MPEG-4 Video Decoding for A Multiprocessor Architecture, *Proc. of 4th Int. Symp. On Embedded Systems (PROGRESS)*, ISBN 90-73461-37-5, Oct. 2003, pp. 185–193.

[91] M. Pastrnak, P. Poplavko, P.H.N. de With, and J.L. van Meerbergen, Hierarchical QoS Concept for Multiprocessor System-on-chip, *Proc. of Work. On Resource Management for Media Processing in Networked Embedded Systems*, ISBN 90-386-0544-7, Mar. 2005, pp. 139–142.

[92] M. Pastrnak, P. Poplavko, P.H.N. de With, and J.L. van Meerbergen, Novel QoS Model for Mapping of MPEG-4 Coding onto MP-NoC, *Proc. of 9th IEEE Int. Symp. on Consumer Electronics (ISCE)*, ISBN 0-7803-8920-4, June 2005, pp. 93–98.

[93] A. Pearmain, J. Cosmas, A. Carvalho, and V. Typpi, The MoMuSys MPEG-4 Mobile Multimedia Terminal and Field Trials, *Proc. of ACTS Mobile Communications Summit 1999*, June 1999, pp. 741–746.

[94] F. Pereira and Touradj E., *The MPEG-4 Book*, Upper Saddle River, NJ: IMSC Press, July 2002.

[95] D. Pham, S. Asano, M. Bolliger, M.N. Day, *et al.*, The Design and Implementation of a First-Generation CELL Processor, *Proc. of IEEE Int. Solid-State Circuits Conf. (ISSCC)*, ISSN 0193-6530, Feb. 2005, pp. 134–135.

[96] P. Poplavko, T. Basten, M. Bekooij, J. van Meerbergen, and B. Mesman, Task-level timing models for guaranteed performance in multiprocessor networks-on-chip, *Proc. of Int. Conf. on Compilers, Architecture and Synthesis for Embedded Systems*, ISBN 1-58113-676-5, Oct. 2003, pp. 63–72.

[97] P. Poplavko and M. Pastrnak, Modeling Predictable Multiprocessor Performance for Video Decoding, *Proc. of Work. On the Design of Multimedia Architectures (MMA)*, ISBN 90-386-0822-5, Dec. 2003, pp. 133–136.

[98] P. Poplavko, M. Pastrnak, T. Basten, P.H.N. de With, and J.L. van Meerbergen, Mapping MPEG-4 Video Object Shape-Texture Decoding onto an Multiprocessor Network-on-Chip, *Proc. of 14th Int. Work. on Circuits, Integrated systems and Signal Processing (ProRisc)*, ISBN 90-73461-39-1, Nov. 2003, pp. 139–147.

[99] Secondlife project: http://secondlife.com/.

[100] A. Puri and A. Eleftheriadis, MPEG-4: an object-based multimedia coding standard supporting mobile applications, *Mobile Networks and Applications archive, Special issue: mobile multimedia communications*, Ser. 1, Vol. 3, ISSN 1383-469X, 1998, pp. 5–32.

[101] I.E.G. Richardson, *H.264 and MPEG-4 Video Compresion, Video Coding for Next-generation Multimedia*, John Wiley & Sons Ltd., Chichester, ISBN 0-470-84837-5, Sept. 2003.

[102] E. Rijpkema, K. G. W. Goosens, A. Radulescu, J. Dielissen, J. van Meerbergen, *et al.*, Trade offs in the design of a router with both guaranteed and best effort services for networks on chip, *Proc. of Design, Automation and Test in Europe Conference and Exhibition (DATE)*, ISBN 0-7695-1870-2, Mar. 2003, pp. 350–355.

[103] B. Sabata, S. Chatterjee, M. Davis, J. Sydir, and T. Lawrence, Taxanomy of QoS Specifications, *Proc. of the IEEE 3rd Int. Work. on Object-oriented Real time Dependable Sys. (WORDS '97)*, 1997, pp. 100–107.

[104] M. Schaar and H. Radha, A hybrid temporal-SNR fine-granular scalability for Internet video, *IEEE Trans. on Circuits and Systems for Video Technology*, Ser. 3, Vol. 11, ISSN 1051-8215, Mar. 2001, pp. 318–331.

[105] K. Seo, J. Ko, I. Ahn, *et al.*, An Intelligent Display Scheme of Soccer Video on Mobile Devices, *IEEE Trans. on Circuits and Systems for Video Tech.*, Ser. 10, Vol. 17, ISSN 1051-8215, Oct. 2007, pp. 1395–1401.

[106] S. Sriram and S. S. Bhattacharyyan, *Embedded Multiprocessors: Scheduling and Synchronization*, Marcel Dekker Inc., New York, ISBN 0-8247-9318-8, Mar. 2000.

[107] S. Stuijk, *Predictable Mapping of Streaming Applications on Multiprocessors*, PhD thesis, University of Technology Eindhoven, ISBN 978-90-386-1624-7, Oct. 2007.

[108] S. Stuijk and T. Basten, Analyzing concurrency in computational networks., *Proc. of MEMOCODE 2003, 1th Int. Conf. on Formal Methods and Models for Co-Design*, ISBN 0-7695-1923-7, June 2003, pp. 47–48.

[109] Y. Tan, P. Malani, Q. Qiu, *et al.*, Workload prediction and dynamic voltage scaling for MPEG decoding, *Proc. of the 2006 Conf. on Asia South Pac. Design Autom.*, ISBN 0-7803-9451-8, Jan. 2006, pp. 911–916.

[110] F. Thoen and F. Catthoor, *Modeling, Verification and Exploration of Task-level Concurency in Real-Time Embedded Systems*, Kluwer Academic, Boston, ISBN 0-7923-7737-0, Sept. 1999.

[111] D.S. Turaga, M. van der Schaar, and B. Pesquet-Popescu, Complexity scalable motion compensated wavelet video encoding, Ser. 8, Vol. 15, ISSN 1051-8215, Aug. 2005, pp. 982–993.

[112] M. van der Schaar-Mitrea, *System and Network Constrained Scalable Video Compression*, PhD thesis, TU Eindhoven, Dec. 2001.

[113] J. van Eijndhoven, J. Hoogerbrugge, M.N. Jayram, P. Stravers, and A. Terechko, Chapter 4: Cache-Coherent Heterogeneous Multiprocessing as Basis for Streaming Applications, in P. van der Stock (Ed.), *Dynamic and Robust Streaming in and between Connected Consumer-Electronic Devices*, Springer, ISBN 1-4020-3453-9, May 2005, pp. 81–108.

[114] B. Vermeulen, S. Oostdijk, and F. Bouwman, Test and debug strategy of the PNX8525 nexperia digital video platform system chip, *IEEE International Test Conference (ITC)*, 2001, pp. 121–131.

[115] J.A. Vijverberg, N.A.H.M. de Koning, Jungong Han, P.H.N. de With, and D. Cornelissen, High-Level Traffic-Violation Detection for Embedded Traffic Analysis, *Proc. of IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, ISSN 1520-6149, Apr. 2007, pp. II–793–II–796.

[116] M. Wan, H. Zhang, V. George, M. Benes, A. Abnous, *et al.*, Design methodology of a low-energy reconfigurable single-chip DSP systems, *Journal of VLSI Signal Processing*, Ser. 1, Vol. 28, ISSN 1573-109X, Jan. 2001, pp. 47–61.

[117] P. Wang, Y. Yemini, D. Florissi, and J. Zinky, A distributed resource controller for qos applications, *Proc. of IEEE/IFIP Network Operations and Management Symp.*, ISBN 0-7803-5927-5, Apr. 2000, pp. 143–156.

[118] Y. Watanabe, T. Yoshitake, K. Morioka, T. Hagiya, H. Kobayashi, *et al.*, Low power MPEG-4 ASP codec IP macro for high quality mobile video applications, *Proc. of IEEE Int. Conf. on Consumer Electronics (ICCE)*, ISBN 0-7803-8838-0, Jan. 2005, pp. 337–338.

[119] S. Wenger, Temporal scalability using P-pictures for low-latency applications, *Proc. of IEEE 2nd Workshop on Multimedia Signal Processing*, ISBN 0-7803-4919-9, Dec. 1998, pp. 559–564.

[120] T. Wiegand and G.J. Sullivan, The H.264/AVC Video Coding Standard [Standards in a Nutshell], *IEEE Signal Processing Magazine*, Ser. 2, Vol. 24, ISSN 1053-5888, Mar. 2007, pp. 148–153.

[121] D. Wilson and M. Ghanbri, Optimal DCT coefficient adjustment applied to MPEG-2 SNR scalability, ISBN 0-7803-3925-8, 1997, pp. 1664–1668.

[122] D. Yu and J.B. Ra, Fine spatial scalability in wavelet based image coding, *IEEE Int. Conf. on Image Processing*, ISSN 1522-4880, Sept. 2005, pp. II –862–5.

[123] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, RSVP: A New Resource ReSerVation Protocol, *IEEE Network*, Ser. 5, Vol. 7, ISSN 0890-8044, Sept. 1993, pp. 8–18.

[124] J.A. Zinky, D.E. Bakken, and R.E. Schantz, Architectural support for quality of service for CORBA objects, *Theory and Practice of Object Systems (TAPOS)*, Ser. 1, Vol. 3, ISSN 1074-3227, May 1997, pp. 55–73.

Summary

Performance and QoS-aware MPEG-4 video-object coding for multiprocessor architecture

The introduction of Arbitrary-Shaped (AS) Video Objects (VO) in the MPEG-4 coding standard has enabled various applications using both natural and synthetic composition of video scenes. The work presented in this thesis aims at realizing an embedded-systems design involving the mapping of this type of applications onto a multiprocessor platform, like Network-on-Chip (NoC). The research has focused on the upper design layers, dealing with the application and their control for an efficient execution. The aspects addressed for the mapping are performance modeling of the MPEG-4 decoding, granularity optimization of the algorithm, introduction of task-level scalability, and controlling the quality of the applications by a Quality-of-Service (QoS) manager.

The AS VO MPEG-4 decoding algorithm comprises of the conventional DCT coding techniques from MPEG-1/2 that are extended with the coding of object shapes and specific processing for the improvement of the picture quality of object borders, employing padding and block-based filtering. At the system level, the AS VO MPEG-4 coding allows the designer to think in individual planes and objects that together compose the scene. The target platform for such an application should be able to handle the features of MPEG-4 coding: the combination of high-level control-driven operations and streaming-oriented processing at the video-data level. The platform features a tile-based computing network, in which each tile is separated from the network by buffered communication. This allows multiple instantiation of object decoding, each having its own dynamic behavior.

The Synchronous Data Flow (SDF) graph is a traditional model for computation of multimedia applications mapped on the multiprocessor system. However, SDF cannot cope with the dynamic behavior of object-based video. Therefore, this research has extended SDF by a linear parametrical model of the required computation resources. The model is based on the coding parameters of the input stream (BAB-type of the block, number of non-transparent

sub-blocks, number of AC coefficients coded by an ESC code, etc.) and weighting coefficients depending on the target processor architecture. Similarly, thesis proposes a parametrical model for the communication resources. It was found that our obtained parametrical timing model has about 5% deviation from the real execution on an Æthereal NoC with ARM7 cores. Our comparison with the mostly used worst-case approach for communication resource allocation revealed that it reduces the required resources with a factor of 2.5.

For more efficient system control, the thesis presents a hierarchical Quality-of-Service (QoS) concept in combination with a scalable MPEG-4 decoder. To serve scalable execution, we have classified the tasks involved with the AS VO MPEG-4 decoding into two classes. The first class contains essential tasks that cannot be skipped, while the second class is filled with the enhancement functions. Scalability of AS VO MPEG-4 decoding was obtained by enabling/disabling optional functions of the non-essential tasks next to the essential tasks. The resource distribution is controlled by a hierarchical QoS management. This QoS is based on two QoS managers. In our experimental implementation, the Local QoS provides the estimation of the resource-usage of an application and monitors the real execution. The Global QoS selects the best quality-levels of the active applications and reserves resources for the application. The key contribution of our work on QoS is the design of a heuristic algorithm that searches suitable combinations of quality levels for individual jobs, so that a set of jobs can be mapped on the available resources.

In order to further improve the efficiency of the mapping, we have distinguished reservation-based QoS control and best-effort computing on top of it as an addition. This combination was studied for controlling the bandwidth of the communication resources. The reservation-based approach guarantees that the video object will be always decoded at least at the lowest quality level, while the best-effort computing improves the quality by using the resources as much as they are available, as controlled by the Global QoS. The complete system was experimentally verified with a network of eight ARM processor cores, using an MPEG-4 Video Object decoder at the ACE profile and at CCIR-601 resolution. The proposed framework showed that the adaptation at finer granularity, e.g. a VOP level within a GOV, significantly improve the image quality (provided that resources are constrained.

The mapping exploration of AS VO MPEG-4 decoding for execution on an NoC addresses a general case of running modern multimedia applications, because of the variability and dynamics of tasks. It has been shown that parametrical models help in planning the execution and QoS management and best-effort computing clearly improve the efficiency of multiple tasks executed in parallel.

Curriculum Vitae

Milan Paštrnák was born in Čadca, Slovakia, on May 14th, 1976. In 1994, he graduated from Turzovka Gymnázium, a comprehensive high school, in Slovakia. He obtained the M.Sc. degree in Information Systems at the University of Žilina, Slovakia, in 1999. In 2002, he received the Professional Doctorate in Engineering degree in Software Technology from Eindhoven University of Technology, in The Netherlands. Between 2002 and 2006, he was employed by LogicaCMG Nederland, where he performed his PhD research work.

Since 2007, he is with Philips Research Laboratories, in Eindhoven, The Netherlands, as Research Scientist. He is currently working on video content analysis for consumer electronics appliances. He received a Best Paper Award at the IEEE ISCE in 2006 for his work on QoS management on Multiprocessor NoC. His research interests are on the hardware-software co-design, design of multiprocessor systems, quality-of-service for multimedia systems, system-level design, and content analysis.