Video post processing architectures

Video Post Processing Architectures

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 8 mei 2008 om 16.00 uur

door

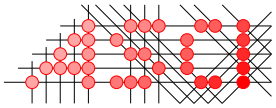
Aleksandar Berić

geboren te Belgrado, Servië

Dit proefschrift is goedgekeurd door de promotoren:

prof.dr.ir. G. de Haan
en
prof.dr.ir. J.L. van Meerbergen

Copromotor:
dr.ir. J.A.J. Leijten



This work was carried out in the ASCI graduate school.
ASCI dissertation series number 163.

CIP-DATA LIBRARY TECHNISCHE UNIVERSITEIT EINDHOVEN

Berić, Aleksandar

Video post processing architectures / by Aleksandar Berić. –
Eindhoven : Technische Universiteit Eindhoven, 2008.
Proefschrift. -ISBN 978-90-386-1844-9
NUR 959
Trefw.: digitale beeldverwerking / parallelle computersystemen / computerontwerp ;
microprocessoren / computergeheugens.
Subject headings: video signal processing / motion estimation / computer architecture /
memory architecture.

To Tijana, Luka, ...

Contents

Summary

Video Post Processing Architectures

The current IC technology has advanced to a level where millions of transistors integrated on a die of just a few square millimeters provide billions of operations per second. One of the current challenges is to increase the design efficiency by balancing the dedicated and more flexible (software programmable) hardware, and to increase the available off-chip memory bandwidth. These challenges are particularly important for the target application domain of this thesis, video post processing. Applications from video post processing are designed to adapt the stored or broadcasted video data to the individual display properties, the reception/viewing conditions and the taste of the viewer. These applications include de-interlacing, picture-rate up-conversion, spatial (resolution) scaling, noise reduction, sharpness and contrast enhancement, etc. Each of the functions in the domain can be implemented using different algorithms. This algorithmic diversity is an additional challenge of our domain.

We identify the memory subsystem as the key component that determines the compute performance and cost efficiency of the architecture. A cost effective memory subsystem, sufficiently flexible within the application domain is presented. The memory subsystem is based on scratchpad memories, which enable predictable performance, and the architecture is scalable to support a number of performance points. This memory subsystem is meant to be embedded in a VLIW-based ASIP with a vector instruction set, which is our architecture of choice. Such an architecture can exploit the instruction and data level parallelisms abundantly present in the video processing domain.

We propose a scratchpad architecture that enables one- and two-dimensional accesses to arbitrarily positioned blocks of data. These access patterns are needed in the application domain, typical examples are pixel filtering and motion estimation/compensation. To bridge the gap between the offered and requested off-chip memory bandwidth, we propose a scratchpad organization and addressing technique to minimize the off-chip memory

bandwidth. This concept enables a tradeoff between the scratchpad capacity and off-chip memory bandwidth. The proposed two techniques, for un-aligned access and bandwidth reduction, naturally map onto two levels of memory hierarchy. We show, however, that it is possible to apply both techniques to a memory subsystem based on just one level of memory hierarchy and list the number of advantages of such a concept. The effects to area, power, bandwidth requirement, software complexity and impact on algorithm have been analyzed.

The approach we adopt here is to fetch from a local memory subsystem relatively large groups of pixels, for example containing 16 or 32 pixels. The number of pixels equals the width of the datapath. After being fetched from the memory subsystem, these pixels are directly processed by the datapath that uses the instruction set with vector extensions. We identify two major algorithmic classes and show how they can be vectorized to match our architecture. Both algorithmic classes have similar requirements from the architecture point of view, un-aligned data accesses and 1D or 2D access patterns. The first class contains block-based motion estimation/compensation and the second one pixel-based content-adaptive filtering. They are generic enough to cover a range of applications with high quality and acceptable implementation cost. Our memory subsystem is benchmarked against other approaches using the proposed set of six criteria. Per individual criterion, it is at least equally good as any other evaluated solution performing on that criterion. Finally, we include a proof of concept. A processor that implements a high-quality motion estimation algorithm and processes HDTV material (1920*1080) in real-time, has been developed based on the concepts presented in this thesis. Comparison of the performance per square millimeter with the state-of-the-art media processor shows that our implementation is eight times more efficient.

Samenvatting

Architecturen voor Beeldsignaal Verbetering

De huidige IC technologie is uitgegroeid tot een niveau waar miljoenen transistoren biljoenen operaties per seconde uitvoeren op een geïntegreerd circuit van slechts enkele vierkante millimeters. Op dit moment zijn er twee belangrijke uitdagingen. De eerste betreft het verhogen van de ontwerpefficiëntie door het balanceren van functiespecifieke en flexibele, d.w.z. software programmeerbare, logische schakelingen. De tweede het verhogen van de beschikbare en benodigde bandbreedte naar het externe geheugen. Deze uitdagingen zijn met name belangrijk voor het applicatiedomein van deze thesis, beeldverbetering. Toepassingen van beeldverbetering zijn, en worden nog steeds, ontwikkeld om opgeslagen of uitgezonden videosignalen aan te passen aan de individuele beeldschermeigenschappen, de ontvangst- en weergavecondities, en de smaak van de kijker. Als voorbeelden noemen we de-interliniering, omzetting van de beeldherhalingsfrequentie, resolutievergroting, ruisfiltering, scherpte- en contrastverbetering, etc. Voor elk van deze functies in het applicatiedomein zijn realisaties met verschillende algoritmen beschikbaar. Naast deze algoritmediversiteit is ook de hoge vereiste bandbreedte naar het externe geheugen een grote uitdaging.

Wij identificeren het geheugensubsysteem als de belangrijkste component voor de prestaties en kostenefficiëntie van de architectuur. Een kostenefficiënt geheugensubsysteem wordt gepresenteerd dat voldoende flexibiliteit biedt binnen het applicatiedomein. Dit geheugensubsysteem is gebaseerd op scratchpadgeheugens, die voorspelbare prestaties laten zien, en een schaalbare architectuur toestaan om meerdere prestatiepunten te ondersteunen. De gekozen architectuur, waarin dit geheugensubsysteem is bedoeld opgenomen te worden, is een VLIW-gebaseerde ASIP met een vector instructieset. Een dergelijke architectuur is geschikt voor benutting van parallellisme op instructie- en dataniveau, welke veelvuldig aanwezig is in de beeldbewerkingapplicaties.

Wij hebben een scratchpadarchitectuur voorgesteld die één- en tweedimensionale toegang tot willekeurig geplaatste datablokken toelaat (unaligned access). Deze toegangspa-

tronen zijn nodig in het applicatiedomein, en typische voorbeelden zijn pixelfiltering en bewegingsschatting/compensatie. Om de kloof tussen aangeboden en gevraagde externe geheugenbandbreedte te overbruggen, hebben wij een scratchpadorganisatie en adresseringstechniek voorgesteld om de externe geheugenbandbreedte te minimaliseren. Dit concept staat een uitruil van scratchpadcapaciteit en externe geheugenbandbreedte toe. De twee voorgestelde technieken, voor un-aligned access en bandbreedtevermindering, zijn op een natuurlijke manier af te beelden op twee niveaus van geheugenhiërarchie. Wij tonen echter aan, dat het ook mogelijk is om de beide technieken toe te passen op een geheugensubsysteem dat op één enkel niveau van geheugenhiërarchie is gebaseerd. De voordelen van een dergelijk concept, en de gevolgen voor oppervlakte, dissipatie, bandbreedte-eisen, softwarecomplexiteit, alsmede het effect op de algoritmen worden geanalyseerd.

Onze benadering hier is om een grote groep van pixels uit het geheugensubsysteem te halen, bijvoorbeeld een groep van 16 of 32 pixels. Het aantal pixel is gelijk aan de breedte van het datapad. Nadat ze uit het geheugensubsysteem gehaald zijn, worden deze pixels direct verwerkt door het datapad, waarbij de instructieset met vectoruitbreidingen gebruikt wordt.

Wij identificeren twee belangrijke algoritmeklassen en tonen aan hoe zij gevectorizeerd kunnen worden om ze aan onze architectuur aan te passen. Beide algoritmeklassen hebben vergelijkbare vereisten vanuit het architectuurstandpunt, de unaligned data access en 1D of 2D toegangspatronen. De eerste klasse bevat blokgebaseerde bewegingsschatting/compensatie en tweede het data-afhankelijk filtreren. Deze twee klassen zijn generiek genoeg om een breed scala aan toepassingen met hoge kwaliteit en aanvaardbare implementatiekosten af te dekken.

We vergelijken ons geheugensubsysteem met andere oplossingen op basis van een voorgestelde reeks van zes criteria. Per individueel criterium blijkt onze oplossing beter of tenminste even goed te presteren als de beste van de alternatieven. Ook presenteren wij een voorbeeldimplementatie om onze concepten te bewijzen: Een processor voor hoge kwaliteit real-time bewegingschatting op HDTV materiaal (1920*1080), gebaseerd op de voorgestelde concepten. De vergelijking van de prestaties per vierkante millimeter met een recente referentie processor toont aan dat onze implementatie acht keer efficiënter is.

Acknowledgment

This PhD thesis is not the work of a single man. Many people contributed in many ways. I am thankful to Prof. Ralph Otten for providing me a PhD student position at the ICS group of the Electrical Engineering section of the TU/e and dr. Albert van der Werf for hosting me at the ESAS group of Philips Research.

I am thankful to Prof. Gerard de Haan for an opportunity to work on a challenging research topic of advanced video post processing. He helped me in the initial phases of my work to understand the basic algorithmic concepts and greatly improved my early publications. Any research field is broad and setting up the search direction is a great challenge. Prof. Jef van Meerbergen helped me to to find the direction of my research. The ideas proposed in our first joint research paper (Prorisc2002) form the skeleton of Chapter 4. Both of my professors greatly contributed in purification of the ideas and presentation clarity of this thesis. Other four members of the core committee, Prof. Jan Biemond, Prof. Henk Corporaal, Prof. Hartmut Schröder and dr. Jeroen Leijten provided me very good comments and helped to improve the quality of this thesis. I would like to thank dr. Ramanathan Sethuraman, for mentoring me on a day-to-day bases, for all the support, challenges, deadlines, interesting discussions and the CATDI project which shaped this thesis. Apart from Ramanathan, I am also indebted to my other CATDI team-mates, dr. Marc Duranton, Gerard Veldman, dr. Carlos Alba Pinto and Harm Peters. Marc was asking difficult questions about the first multi-core CATDI architectures and reviewed the first DMA specs (still alive). Gerard contributed to Section 4.4 and we also spent some nice time discussing custom-ops and debugging. During implementation of the ideas proposed in this thesis, my colleagues from Silicon Hive were always willing to help in scheduling, debugging, fixing the core dumps, adjusting the tool versions, etc. My student Radomir Jakovljević contributed with innovative implementation of the 3DRS algorithm, which is summarized in Appendix C.

Marja de Mol and Rian van Gaalen from the TU/e as well as Ann Brebels from Philips Research ensured that I felt very comfortable. Meng Zhao, Srinath Naidu, Vladimir Lale

Živković, Jürgen von Oerthel were one of many who took care that the lunch and coffee breaks are worth remembering. My friend Nebojša Fišeković introduced me to Philips basketball and tennis club, as well as to the secrets of ”klussen”.

Lastly, the VIP section. My family and in-laws in Serbia were a big support throughout this effort. I will always remember the help and love from my brother Andreja, mother Zora, father Branko and my father-in-law, Colonel Saplaić. Andreja also spent a few hours of his talent on embedding un-aligned 2D access in the Ring Nebula, which became the cover design of this thesis. I am thankful to many other relatives and friends for all their smiles, hospitality and many special moments we had together. And finally, daily life of my wife Biljana and our Tijana & Luka was most affected by this work, especially during the time of writing the thesis. To make things worse, writing had to go in parallel with a challenging job at Silicon Hive. Just saying thanks would not be appropriate for all the time I was stolen from them.

1

Introduction

REVOLUTIONS concern a drastic and far-reaching change in ways of thinking or behaving. The *digital revolution* brought the transition from analogue to digital signals, which by now has led to the almost complete domination of digital signal processing over analogue. Until the second half of the twentieth century, analogue signal processing was the only way to implement an application. The arguments in favor of digital, reproducibility and predictability, were not strong enough to outweigh the increased component count of digital. The digital revolution could happen only upon the cost reduction of digital implementation and the cost reduction required some fundamental research.

In 1946, J. W. Mauchly and J. P. Eckert correctly predicted the need for electronic computers and developed the ENIAC I (Electrical Numerical Integrator And Calculator) [1, 2]. This calculator occupied 167 m^2 of floor space, weighed 30 tons and consumed 160 kilowatts. Despite being large and clumsy, the ENIAC I was important enough to cause decades of discussions and patent (in)validations [3]. To arrive at the pocket version we know today, fundamental research was essential. A tiny invention with a big future, also known as the transistor, was the result of fundamental research at Bell Labs. Invented in December 1947 and published in July 1948 [4], the transistor marked the beginning of the digital era because of its potential for miniaturization that was impossible with the vacuum tubes they replaced. Although the Nobel Prize for physics in 1956 was awarded for this invention [5], the discussion who really invented it has not stopped even today [6].

After the invention of the transistor, a bit more than a decade was needed until another idea further pushed the wheel of digital revolution. Independently from eachother, J. S. Kilby, a debutant at Texas Instruments and R. Noyce from a startup called Fairchild Semiconductor, later co-founder of Intel, integrated a complete circuit containing a transistor and other components. Both are recognized as the inventors of the first integrated

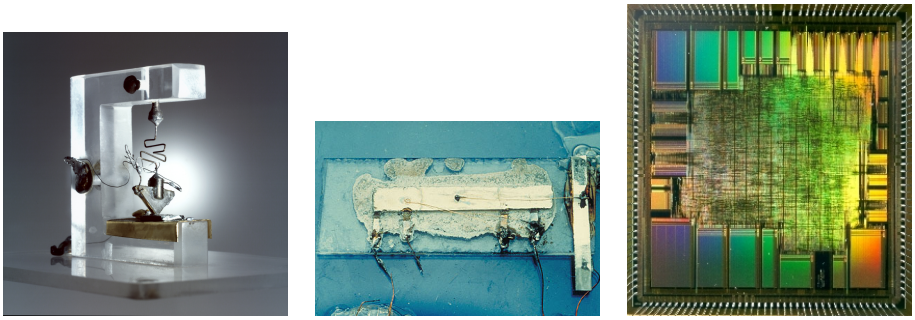


Figure 1.1: From a single transistor, to few components integrated and to millions of them. Courtesy of Bell Labs, Texas Instruments and Philips Semiconductors.

circuit (IC). Only a few years later, in April 1965, Gordon E. Moore, yet another Intel co-founder, formulated a self-fulfilling prophecy stating that the number of transistors on an IC will increase exponentially, doubling their number roughly every two years [7]. This initiated a race between companies to stay on Moore's curve and the process of miniaturization continues. The result is that today a transistor is probably the most massively manufactured electrical device while a typical IC contains millions of them.

Already in 1946, the transistor inventors predicted that their creation will find its use in many applications, including television. Even though the British Broadcasting Company (BBC) started the first regular black and white television broadcast in 1936, it took a few decades of underlying technology research and development until television reached popularity. At the beginning of the second half of the twentieth century, three broadcasting standards were developed, one in the USA (NTSC) and two in Europe (PAL and SECAM). By the year of 1970, regular color TV broadcast had started in most European countries. Initially, the TV merely played the role of status symbol, but soon it became an inseparable part of every home. Originally, its main purpose was viewing at distance, allowing millions of people to see what is happening in some other city, country, the other side of the globe, or even in space. Modern TV-sets offer much more than just a basic viewing at a distance. Large, bright screens with sharp pictures are becoming more and more interactive, they communicate with other devices, support a number of different input standards with compressed and uncompressed video streams, viewing multiple channels at the same time, etc. The latest generations influence even the ambient lighting to provide an immerse viewing experience. Each new generation was armed with new features enabled by the evolution of technological progress and thereby an increased number of transistors.

It took half a century to sufficiently improve the IC technology and develop the applications to arrive at today's high-end consumer television. Like all other developments, also this one was driven by application requirements that were always higher than the current machines were capable of. Consequently, the complexity of applications and ICs

Figure 1.2: Dramatic change of the consumer television system required half a century of research and development of IC technology and applications. Advanced digital video post processing applications are executed on multi-million transistor ICs enabling high quality pictures.

has significantly increased. One of the remaining challenges for the IC technology is to increase the design efficiency by balancing the dedicated and more flexible (software programmable) hardware, and to increase the offered off-chip memory bandwidth. Unlike the compute power, the offered off-chip memory bandwidth failed to follow Moore's law. Meanwhile, the screen resolutions and picture rates have been steadily increasing to a level where the bandwidth became the bottleneck. This gap, between the available compute resources and memory bandwidth, directly affects the application performance. This thesis proposes an architecture that increases the efficiency and lowers the bandwidth. The focus is given to the memory subsystem, which enables these benefits. The proposed memory organization enables predictable high-performance. Cost (area, power) is optimized and the off-chip memory bandwidth is kept to a minimum. Through a comparative study, we show the advantages compared to existing solutions.

1.1 Video post processing

The concepts proposed in this thesis are applicable to the application domain of video post processing. Video post processing is a pool of applications designed to adapt the stored or broadcasted video data to the individual display properties, reception and viewing conditions, and the taste of the viewer. These applications include de-interlacing, picture-rate up-conversion, spatial (resolution) scaling, noise reduction, sharpness and contrast enhancement, etc. Although this work focuses on video post processing, it appears that the proposed concepts are applicable even outside this target domain, particularly to video compression. Video compression uses basic algorithms, such as motion estimation, which are also part of the video post processing domain. However, video compression has not been analyzed in detail, and we therefore define video post processing as the scope of this thesis.

Television is one of the strongest drivers for digital video post processing. For decades,

Figure 1.3: A large number of devices, from portable to stationary, having different resolutions and refresh rates, emphasize the need for advanced video post processing.

all video material originated from cameras and video material existed and was broadcasted in just a few formats, PAL, SECAM and NTSC. The three formats were used in different parts of the world, almost without an overlap, so the need for conversion was limited.

At the broadcaster side as well as at the receiver end, the situation has dramatically changed since the first days of television. The diversity is huge now. Modern television sets display video material at various refresh rates that in general range from 50 to 120 Hz. At the source side, video material is recorded at 50 or 60 Hz, while the movie material is recorded at 24, 25 or 30 Hz. Further, video material is transmitted in an interlaced format meaning that odd and even lines of pixels alternately occur in successive pictures. Regarding display resolution, there are two main categories, SDTV (standard definition TV) and HDTV (high definition TV). SDTV assumes an interlaced format with resolution of 720*576 at 50Hz (PAL) or 720*480 at 60Hz (NTSC), while the HDTV standard includes three formats, 1920*1080 interlaced, 1920*1080 progressive and 1280*720 progressive. Clearly, conversion of digital signals from one format to another is necessary.

Television is a strong application driver for digital video post processing, but is not the only one. The digital revolution caused the appearance of a number of other sources of video material in addition to the already existing broadcasting material. This includes artificially created video content like cartoons, special film effects and computer graphics. Furthermore, there are a number of mobile devices that offer various video services. It is possible to follow the news, stock market report, results of sport events, or watch movies on a portable device like a laptop, PDA or a mobile phone. Watching recorded movie material in high quality is also possible by using specialized mobile displays. Figure 1.3 shows some of the devices that are capable of delivering video services.

In summary, different resolutions coupled with interlaced or progressive format and a variety of input and output refresh rates form a huge space of different combinations of input and output formats of video streams. Additionally, a video stream should be displayed in a high quality.

1.2 Domain-specific video processing

Through the process of miniaturization, the digital revolution enabled the implementation of millions of transistors onto a single die. Even though Moore's law might be unconstitutional [8], Gordon Moore correctly predicted this technology trend at the bare beginning of the miniaturization process [7]. His (improved) prediction was matching with the complexity curve of the Intel processors. Integrated circuits used in video processing have been following this trend as well. The recent situation is that millions of transistors form a compute resource pool providing billions of operations per second. The question is how efficiently those resources can be used. We shall recognize three major approaches.

1. *Application-Specific Integrated Circuits* (ASICs) are optimal in terms of area and power and the performance requirements are fully met. The biggest drawback is their very limited flexibility, if there is any flexibility at all. As the applications are becoming more demanding, the complexity of ASICs increases. Thereby, the time spent on hardware verification and debugging increases. Consequently, time-to-design increases and the percentage of first-time right designs drops. These are the key reasons why ASICs are becoming less advantageous and solutions like *General-purpose Programmable Processors* (GPPs) or *Application-domain Specific Instruction-set Processors* (ASIPs) that offer late modifications in software are gaining popularity.

2. GPPs offer a high degree of flexibility and usually have generic *function units* performing addition, subtraction, multiplication, etc. These function units are usually limited to 32 bits with the possibility to use 2-way or 4-way SIMD (Single Instruction, Multiple Data) or *vector* instructions. The SIMD, or vector, model offers computation on *partitioned* data and is one of the oldest models for parallel computation since the earliest ideas are almost half a century old [10]. This offers a way to parallelize the computation but GPPs can profit from it only to a limited extent

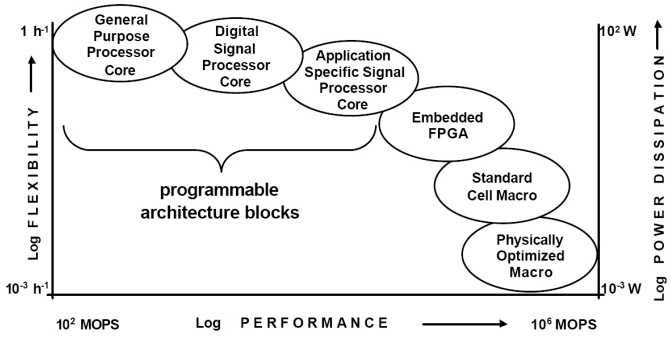


Figure 1.4: The tradeoff between flexibility, performance and power dissipation, source [9].

since the width of the datapath is limited. In order to increase the performance (expressed in terms of number of operations per second), some GPPs partially follow the approach from the ASIP world and use hardware accelerators such as specialized function units. Although they improve the performance, hardware accelerators impair two strong arguments of the GPPs, flexibility and software entry in a high-level language like C.

The usage of specialized hardware only reduces the performance problem. The other bottleneck of GPPs, the narrow datapath that limits the possibilities for parallel computation, remains. This bottleneck grows more significant for longer data words, e.g. when pixel data are moving from 8 to 10 or even 16 bits, which then limits the throughput. To increase the throughput (and therefore the performance), some GPPs use a Very Large Instruction Word (VLIW) architecture [11], which enables *Instruction Level Parallelism* (ILP) [12]. This partly solves the throughput problem for some applications. However, for the most demanding applications aiming at high quality at high resolutions, GPPs cannot cope with the required throughput and I/O data rates.

3. In the last few years, ASIPs have drawn a significant attention from both university and industry [9, 13–19]. Being tuned to a specific application domain they offer high performance suitable for very demanding applications and at the same time they are sufficiently flexible for the target domain. The ASIPs leverage the commonalities between applications, while bridging the gap between the ASICs and GPPs in terms of performance and flexibility (see Figure 1.4). ASIPs are designed to be flexible, but only within an application domain. They ideally offer performance, power and area that are comparable to ASICs. ASIP implementations are superior in terms of performance, power and area compared to GPPs for applications within their domain [9, 15]. These key features of ASIPs make them promising candidates for the next generations of video signal processing architectures.

1.3 Memory bandwidth challenges

The memory speed does not increase as fast as the compute power of processors [12]. This creates a gap between the offered memory bandwidth and available compute resources. This growth tendency gap is illustrated in Figure 1.5. It is interesting to dimension this gap in the context of the target application domain. We shall tackle two aspects, the bandwidth that is required by the processing element, and the memory bandwidth offered by the state-of-the-art memories, frequently used in today's System-on-Chips (SoCs). Additional question is whether the offered bandwidth is sufficient for current and future applications running on these SoCs.

The picture resolution and the refresh rate of displays have grown over time, resulting in higher bandwidth requirements. Additionally, to produce one pixel, algorithms refer-

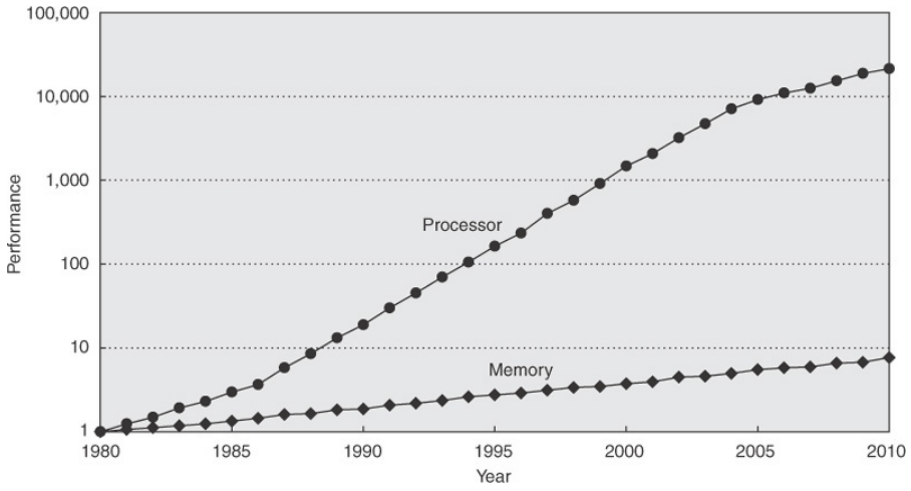


Figure 1.5: The gap between the memory and processor speeds. Source: [12].

ence pixels from few temporal instances and reference the same pixels multiple times. These are some of the main reasons for high bandwidth requirement towards the off-chip picture memory. Example applications are *de-interlacing* [20, 21] and *picture-rate up-conversion* [22]. They are based on *motion estimation* [23], which requires repetitive pixel access. Chapter 4 will show that the raw bandwidth requirement of an extremely efficient motion estimation algorithm performed at HDTV resolution (1920*1080 @ 60 Hz) used in picture-rate up-conversion is equal to 2.7 GB/s. The requested bandwidth can even be larger if the picture rate or resolution increase. In case of 120 Hz processing, the raw bandwidth requirement doubles to 5.4 GB/s. In case of quad HDTV resolution (3840*2160), this bandwidth is four times larger, 10.8 GB/s. This is a large bandwidth requirement, which requires adequate memory subsystem.

The above example discusses only a part of an application (picture-rate up-conversion). A typical SoC executes a chain of applications, such as video format conversion [24] or a Digital TV (DTV) application chain. The bandwidth of the off-chip memory is shared by a number of producers/consumers executing a chain of applications. This causes a lot of traffic to and from the off-chip memory making the requested bandwidth even higher. Let us analyze a typical DTV application chain in a bit more detail. One of the first steps is decoding the input video stream (MPEG2/H.264 are commonly used). The decompressed video signal goes through a sequence of steps performing format conversion and picture quality enhancement. Typically, they include de-interlacing, picture-rate conversion, spatial scaling, picture quality enhancement. These steps produce and consume the intermediate results (pictures), which are stored in the off-chip memory. Some recent (2007) papers estimate the combination of the H.264 decoding and motion compensated format conversion and quality enhancement to require 5.2 GB/s off-chip memory bandwidth (this estimate assumes an on-chip memory subsystem) [25]. In case of dual H.264

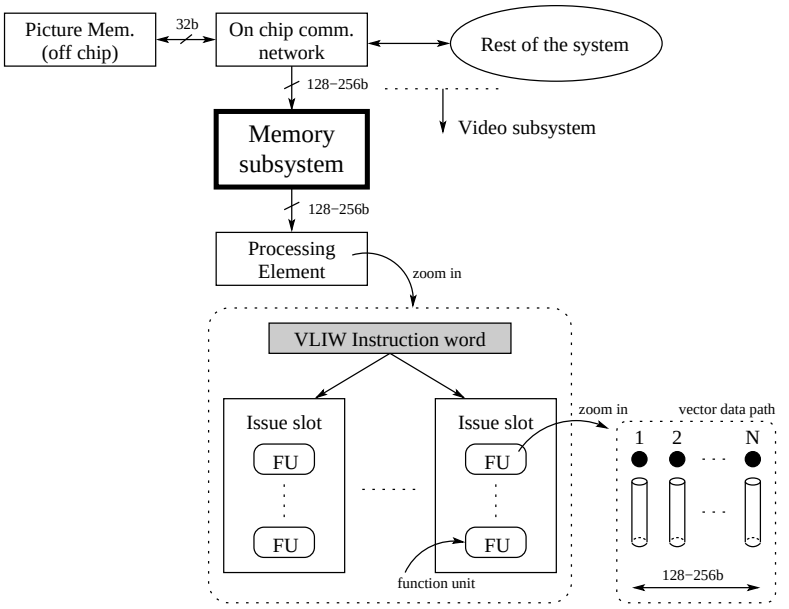


Figure 1.6: The processing template used in this work, a VLIW-based ASIP with vector function units. A customized memory subsystem is needed to provide the data to the function units. This thesis proposes one such customized memory subsystem.

stream, the bandwidth requirement is doubled. The off-chip memory used in today's System-on-Chips (SoCs) is usually realized as an SDRAM (Synchronous Dynamic Random Access Memory). A 32-bit DDR2/DDR3 SDRAM (Double Data Rate SDRAM) with a bus clocked at 400 MHz offers a peak memory bandwidth (without taking into account the high latency) of 3.2 GB/s. Assuming bandwidth efficiency of 80 % [25], the usable bandwidth drops to 2.56 GB/s. In a 64-bit version, the offered bandwidth is twice as large, which is still not enough. This emphasizes the importance of the memory subsystem.

1.4 The processing template

We have selected a VLIW-based ASIP with vector instruction set to be our processing platform. VLIW is an adequate template to exploit the ILP largely present in video processing applications. The application code is processed by the compiler off-line and all the parallelism available in the VLIW machine is utilized, taking into account any data dependencies [11, 26]. Video processing enables parallel computation, as groups of pixels having the same attributes can be identified and processed in the same way. To exploit that, the usage of a vector instruction set looks promising. The benefits of such a vector in-

struction set have already been reported [27–31]. Additionally, changing the vector length (for example, 8, 16 or 32 pixel-wide) enables a scalable solution, which can be applicable for different performance points. The vector instruction set may be generic, customized or mixed [31–33]. A generic instruction set offers the highest degree of flexibility, possibly bigger than needed by the application domain. A customized instruction set is more tuned towards a specific application domain and offers a higher performance. We do not analyze in depth the pros and cons of different instruction sets, but rather choose to use a generic instruction set for the sake of demonstration. By no means, does this decision limit the applicability of this work to only this instruction set.

To maximize the throughput, the presence of the selected processing platform that enables billions of operations per second is necessary, but not sufficient. A memory subsystem should limit the exposed bandwidth towards the off-chip memory and efficiently accommodate the specific access patterns of the application domain. The main focus of this thesis is on such a memory subsystem. The framework of this work is sketched in Figure 1.6.

1.5 This thesis

The topic of this thesis is the architectural research within the application domain of video post processing. The challenges are briefly summarized in the next subsection. Following, the list of main contributions is highlighted. An outline of this manuscript concludes this chapter.

1.5.1 Challenges

A large number of algorithms enabling the basic functions in the application domain have been reported in the literature and/or have already been realized in silicon. From a processing point of view, different algorithms use a different number of picture references, apply different filtering schemes, etc. At the application side, the picture resolutions and rates have been steadily increasing. This trend is not only present in television screens, but holds as well for other display devices including mobile phones. The algorithmic diversity coupled with high picture resolutions and rates imposes implementation challenges.

While the processor compute performance managed to stay on Moore's curve, the memory speed did not. The consequence is a discrepancy between the offered and requested memory bandwidth. At the processing element side, the bandwidth requirements are high. At the off-chip memory side, the available bandwidth is limited. The memory subsystem should bridge the gap between the offered and requested bandwidth.

In summary, the algorithmic diversity, high requested memory bandwidth at the processing element side and limited available bandwidth at the off-chip memory side represent the implementation challenges. The architecture should be cost-effective, flexible enough within the application domain, offer predictable performance and, finally, be scalable to enable implementation for different picture resolutions and rates.

1.5.2 Contributions

In this thesis, we propose an architecture to implement the algorithms from the video post processing application domain. This architecture enables low off-chip memory bandwidth and high processing element bandwidth (high performance). The architecture is based on a cost-effective memory subsystem, which is sufficiently flexible within the application domain. The performance is predictable and the architecture is scalable to support a number of performance points. The contributions are summarized in the following list.

1. We recognize two distinct algorithmic classes, the first enabling block-based motion estimation/compensation and the other pixel-based content-adaptive filtering. The list of algorithms within the application domain is impressive. The number of different classes should be limited to simplify the task of implementation. The selected two algorithmic classes are generic enough to cover a wide range of applications with high quality under acceptable implementation cost [Chapter 2].

2. In spite of operating on different data granularity (block- vs. pixel-based data), we identify implementation commonalities between the two algorithmic classes that allow a common architecture. We start from the C-description of the two classes and show how both of them can be mapped onto our architecture of choice using the vector instruction set, which enables parallel computation. To prove our concept, we use the generic vector instruction set [Chapter 3].

3. We propose a scratchpad architecture that enables one-dimensional (1D) and two-dimensional (2D) accesses to arbitrarily positioned blocks of data. Both algorithmic classes require these type of accesses. The 1D and 2D accesses are realized within the memory subsystem and release the datapath from this task [Section 4.4].

4. We propose a scratchpad organization and addressing technique to minimize the off-chip memory bandwidth, which also enables a tradeoff between the off-chip memory bandwidth and on-chip memory capacity. This technique is realized in software and enables a flexible implementation [Section 4.5].

5. We discuss the number of memory hierarchy levels. The proposed architecture, which enables 1D and 2D accesses to arbitrarily positioned blocks of data and the technique for bandwidth reduction, naturally maps onto two levels of memory hierarchy. We show, however, that it is possible to design a memory subsystem based on just one level of memory hierarchy using the proposed architecture and the bandwidth reduction technique. The advantages include reduced software complexity, which has an impact on the overall performance and cost, an improved algorithmic performance and, in most cases, a reduced power dissipation and area [Section 4.6].

6. We include a benchmark of our memory subsystem. In order to rank a memory subsystem, we proposed a set of six criteria: minimal off-chip memory bandwidth, predictability, high processing element bandwidth, flexibility, efficiency and scalability. Our memory subsystem is benchmarked with the existing solutions over the

proposed six criteria and proves to be better, on average, than any of the other solutions. Per individual criterion, it is better or equal than any other evaluated solution performing on that criterion [Section 4.7].

7. We include a proof of concept. A processor that implements a high-quality motion estimation algorithm [23] and processes HDTV material (1920*1080) in real-time, has been developed, synthesized and compared with the Trimedia TM3270 processor. Comparison of the performance per square millimeter shows that our implementation is eight times more efficient [Appendix C].

1.5.3 Outline

The remainder of this thesis is organized as follows. Chapter 2 presents the application domain of video post processing. Three pillars of the application domain are identified: Video format conversion, video enhancement and motion estimation. Different algorithms, i.e. different implementations of the applications within the domain are provided for each of the pillars. The number of different algorithms illustrates the diversity of the application domain. From that algorithmic diversity, we have selected two distinct algorithmic approaches (classes) as our major implementation target. The first class is block-based motion estimation/compensation and the second one is pixel-based content-adaptive filtering. Algorithms from these two classes enable a high picture quality under acceptable cost and together, they cover a wide range of applications.

Chapter 3 starts from the proposed classification of the algorithms. The chapter continues with a demonstration how the algorithms from both classes can be adapted to our architecture template, which exploits data level parallelism. Starting from a C-like description in a step-by-step fashion, we show how the algorithms from both classes can be vectorized. We conclude the chapter with the algorithmic requirements for the memory subsystem. In spite of their algorithmic diversity, our conclusions show that both algorithmic classes require access to an arbitrarily positioned group or block of pixels, and the block of accessed pixels is localized within a limited area.

Chapter 4 proposes a memory subsystem that is customized to the needs of the application domain. From the conclusions drawn so far, we identify two basic goals that a memory subsystem should fulfill, access to an arbitrarily positioned block of pixels and reduction of the off-chip memory bandwidth. Section 4.4 proposes a scratchpad architecture that enables one-dimensional and two-dimensional accesses to arbitrarily positioned blocks of pixels. Both access types are needed by the two algorithmic classes. Section 4.5 continues with a scratchpad organization and addressing technique that minimize the off-chip memory bandwidth. This proposal also enables a tradeoff between the off-chip memory bandwidth and required on-chip memory capacity. The important question on the number of memory hierarchy levels is addressed in Section 4.6. After benchmarking the proposed memory subsystem against the prior work in Section 4.7, we conclude the chapter in Section 4.8.

Chapter 5 concludes this thesis and presents some directions for future work.

2

Application domain: overview and analysis

THE goal of this thesis is to find an architecture that can efficiently deal with the entire application domain. This chapter provides an overview of the various methods and algorithms that are applied across the application domain. In order to assess the needs of these methods for architectural resources, we shall also provide an analysis of the application domain from an implementation point of view. In the conclusion of this chapter we shall use this overview and analysis to make the proper algorithmic choices that limit the achievable quality as little as possible.

To render a picture on a modern display, a number of processing steps take place. The picture might originate from a scene captured by the camera sensor, or it might be artificially generated. In this thesis, we assume that the picture is available in a digital format (which is nowadays most often the case) and the processing is performed in a digital domain. The sequence of digitalized pictures goes through a number of processing steps as indicated in Figure 2.1. The first step usually is noise reduction. Further, depending on the input format and the display format, each picture might be de-interlaced. In order to increase the picture rate, picture-rate up-conversion is applied. As the displays are becoming bigger, their resolution increases as well. At the same time, there is plenty of video material available in old (smaller) resolution. In order to display this material, spatial scaling must be applied. Further, there is a large number of algorithms used to enhance the quality of the rendered pictures. Examples are sharpness enhancement, gray-level rescaling, color correction, etc. Finally, motion estimation is sometimes used to enhance the quality of applications, such as de-interlacing or picture-rate up-conversion. As indicated in Figure 2.1, there are also algorithms that are applied for a specific type of displays. Even though Figure 2.1 suggests a specific order of these applications, they could be ordered differently, depending on the characteristics of the specific display, ap-

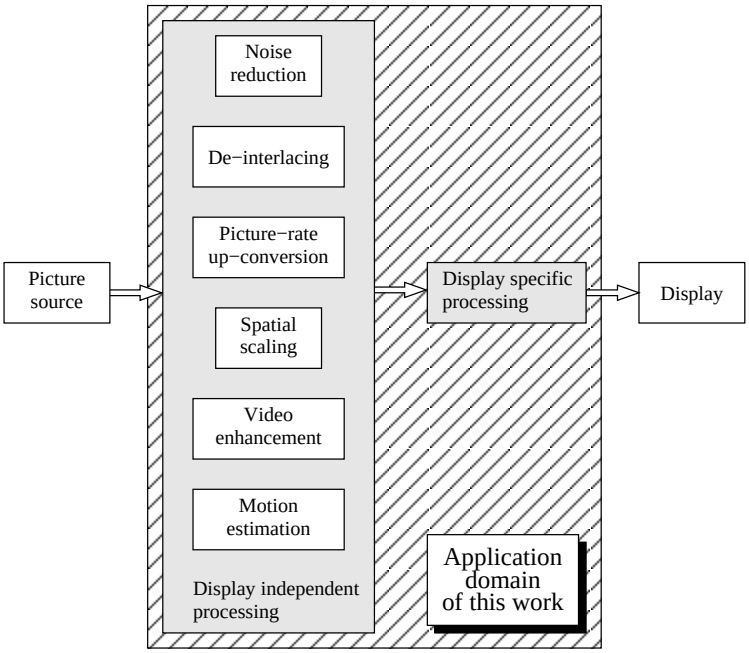


Figure 2.1: The video processing pipe with the application domain that this thesis targets. This thesis focuses to the display independent processing.

plied algorithms, the type of input material, etc.

There is a large number of algorithmic approaches to implement an application. Even though the basic functionality is satisfied, the difference between the performance and cost can be enormous. When describing a certain method, we shall indicate some of the cost indicators: the number of reference pictures used, number and position of pixels used in pixel interpolation and the computational complexity of an algorithm in general. The presentation of this chapter is organized in three pillars, *video format conversion*, *video enhancement*, *motion estimation*. The motivation for this is provided in the next section. The overview of the algorithms given in this chapter was inspired by [34].

This chapter will show that each application from the target domain can be implemented using different algorithms. The diversity of these algorithms is visible in many aspects. The presentation of each application will start from the basic algorithms that merely implement the requested function. The implementation cost of such algorithms is often low, but the achieved quality is low as well. After pointing to the drawbacks of these methods, the story continues with the improved algorithms, which eliminate some of the artifacts of the previous methods and increase the quality. Eventually, reaching the highest quality levels, the story will lead us to one of the two, or both, basic algorithm classes: motion estimation and content-adaptive filtering. This is certainly sure for the

most quality affecting applications from the domain, de-interlacing [21, 35–38], picture-rate up-conversion [22, 39–43] and spatial up-scaling [44, 45]. Based on the analysis of independent applications from the domain, Section 2.5 concludes that motion estimation and content-adaptive filtering are the two algorithmic classes that offer high quality. In addition, the content-adaptive algorithmic class is broad enough to include algorithmic approaches that obtain good quality even with fixed coefficients. Typically, such algorithms are found in the applications to enhance the quality of the rendered pictures.

In this and the following chapters we shall use mathematical equations as a tool to compactly define an algorithm. We denote the luminance value of the pixel located at position (x, y) in picture number n with $F_I(\vec{x}, n)$, where $\vec{x}$ is a compact way of writing (x, y). In the notation, we shall also use unit vectors, i.e. $\vec{u}_x \equiv (1, 0)$ and $\vec{u}_y \equiv (0, 1)$.

2.1 Three pillars of the application domain

This thesis identifies three pillars of the application domain:

- Video format conversion,
- Video enhancement,
- Motion estimation.

Video format conversion is an inevitable ingredient of the post processing application domain and represents our first pillar. A revolution in the display technology made video format conversion key to achieving a good motion portrayal [46]. A good motion portrayal is important in niche markets like 100 Hz television sets. There are plenty of input and output video formats that are currently in use and there is a clear need for conversion from one to another. To mention a few, there is film material recorded at 24 or 25 Hz, video material at 50 or 60 Hz, explosion of streaming videos on the Internet. Input and output video format do not only differ in picture rate but also at the scanning raster (interlaced vs. progressive), and picture resolution. The applications involved in video format conversion are de-interlacing, picture-rate up-conversion and spatial (resolution) scaling.

With state-of-the-art-displays, the quality of the output pictures is one of the key differentiators. Having brighter, crisper, more colorful pictures is on the wish list of any display-related company. Therefore, video enhancement is our second pillar. Key application drivers are noise reduction, sharpness, contrast and color enhancement.

Lastly, motion estimation has recently become a key enabler of many of the processing steps within the video chain. Motion compensated algorithms are usually the ones that provide the highest quality of pictures. The reason that motion estimation was not widely used in the past was its cost. Recently though, it was shown that motion estimation and compensation are feasible at acceptable implementation cost [23]. This fact firmly embedded motion compensated processing into the video chain and made our third pillar motion estimation.

2.2 Video format conversion

All three applications from the video format conversion pillar synthesize pixels at new spatio-temporal locations. The most simple methods are linear methods that lead to a poor quality of output pictures since video signals are in general not stationary signals. We begin this section starting from those linear methods, and then continue with non-linear methods that in general offer better quality under higher needs for computational resources.

2.2.1 De-interlacing

Interlace is a technique of alternatively transmitting odd and even lines of a picture. At picture number n, half of the picture (called field) containing the odd lines is transmitted and at picture number $n + 1$, the other field containing the even lines is transmitted. Interlaced transmission is illustrated in Figure 2.2.

Interlace has been invented in the early thirties. The credits for this invention typically go to R. C. Ballard and somewhat less often to F. Schroeter. The main advantage of this invention is reduction of channel bandwidth. Interlaced video namely reduces the signal bandwidth by a factor of two, for a given spatial resolution and refresh rate. For example, 1920*1080i HDTV signal with a 60 Hz field rate has a similar bandwidth to 1280*720p HDTV with a 60 Hz picture rate, but approximately twice the spatial resolution. Its main drawbacks are that it performs poorly on moving pictures, leading to saw tooth (combing) and other artifacts and that fine vertical detail is subject to flicker with two times lower frequency as the rest of the picture.

Contemporary displays use progressive scan and the need for interlace is lessened significantly with time. Currently, the transmission is mostly digital and initially there is a lot of bandwidth available [47]. However, in the process of analogue-to-digital TV set conversion, an increasing number of subscribers is asking for more advanced, bandwidth-hungry services like video-on-demand. The acceptance of the HDTV standard increased the pixel count 5 times (from SDTV (720*576i) to HDTV (1920*1080i)), which increases

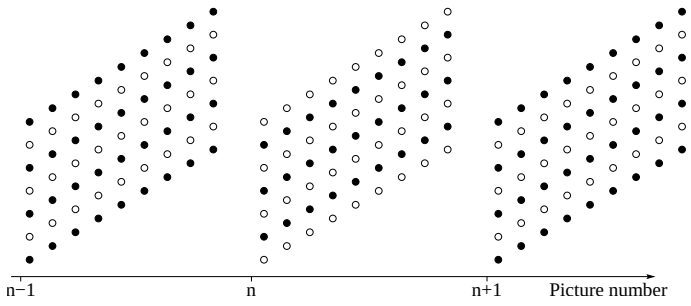


Figure 2.2: The basic principal of interlaced video. Only alternating pixel lines (marked as black circles) are transmitted.

the needed bandwidth. In conclusion, we cannot be sure that the bandwidth issues are solved for good. Furthermore, digital video broadcast already applies video compression to save bandwidth [48]. Thereby, the question to interlace or not to interlace is still quite open. In summary, interlace is the origin of years of research, legacy we have to deal with and possibly a technique that will continue to be used in the future as well.

De-interlacing is a technique that converts the interlaced input video material or *fields* to progressive video material or *frames*. Formally, the de-interlaced frame is defined with the following equation:

$$F_O(\vec{x}, n) = \begin{cases} F_I(\vec{x}, n), & y \bmod 2 = n \bmod 2 \\ F_{INT}(\vec{x}, n), & \text{otherwise} \end{cases} \tag{2.1}$$

where F_{INT} denotes the interpolated pixels.

De-interlacing is one of the major determinants of picture quality in a modern display processing chain. It takes one of the first positions within the video chain, before the picture-rate up-conversion. It is thereby important to minimize the artifacts that it produces since they propagate further and usually get amplified on their way to the display. Figure 2.3 illustrates the challenges of de-interlacing by showing some of the artifacts.

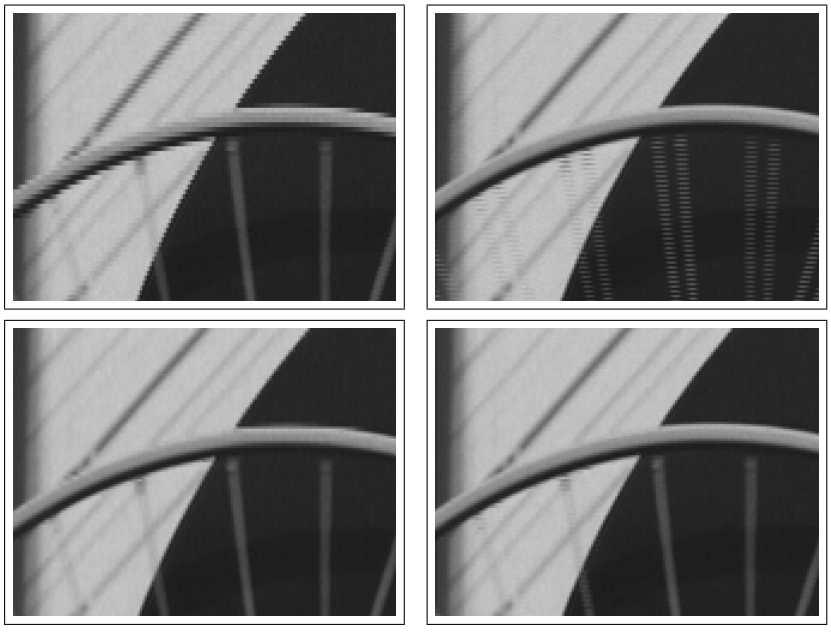


Figure 2.3: Different artifacts of de-interlacing. The picture in the top left corner was obtained by applying line repetition algorithm, top right field repetition (insertion), bottom left edge-dependent de-interlacing and bottom right motion compensated de-interlacing based on Generalized sampling theorem.

2.2.1.1 Linear methods

All linear methods are defined with the following equation:

$$\begin{array}{l} F_{INT}(\vec{x},n) = \sum_k F(\vec{x}+k\vec{u}_y, n+m)h(k,m), \\ \text{where} \quad k,m \in \mathbb{Z}, (k+m) \bmod 2 = 1 \end{array} \tag{2.2}$$

In the above formula $h(k,m)$ denotes the impulse response of the filter.

The two simplest and probably best known linear de-interlacing methods are line repetition (spatial de-interlacing) and field insertion (temporal de-interlacing) [49]. Line repetition is a spatial method that can be derived from Equation 2.2 when $h(k,0) = 1$ for $k = -1$ and $h(k,m) = 0$ otherwise. This is probably the cheapest de-interlacing method, which requires only one complete pixel-line and practically no pixel processing. The strong alias that results from this method (visible in Figure 2.3) can be reduced by increasing the order of interpolation. Increasing the order of interpolation by one leads to the line-averaging or popular Bob de-interlacing. The filter impulse response is defined with $h(k,0) = 0.5, k \in \{-1,1\}$ and $h(k,m) = 0$ otherwise. Bob de-interlacing still has modest memory capacity requirements and a simple 2-tap filtering, i.e. line averaging.

Due to the high temporal correlation of the incoming pictures, the next logical step would be to profit from it. Field repetition or field insertion (also known as Weave in the PC community) is the simplest linear temporal de-interlacing method. Its impulse response results when selecting $h(0,-1) = 1$ and $h(k,m) = 0$ otherwise. This method provides the perfect result in case of stationary video sequence but quite annoying artifacts in the case of motion as can be seen in Figure 2.3.

Field insertion does require one field reference that makes it memory-wise much more expensive compared to its spatial counter part. Better (and more costly) results can be obtained if longer FIR filtering is applied. The number of filter taps defines the complexity of implementation and the memory requirements. This makes the Weave de-interlacing implementation-wise unattractive as the number of used field references is equal to the number of filter taps.

Vertical-temporal filtering is a linear filter that combines spatial and temporal information available in the sequence of pictures. This filter is designed such that only the higher frequencies are used from the neighboring fields. An example realization is given in the following equation:

$$h_{VT}(k,m) = \begin{cases} 1,8,8,1, & k \in \{-3,-1,1,3\} \wedge m = 0 \\ -5,10,5, & k \in \{-2,0,2\} \wedge m = -1 \\ 0, & \text{otherwise} \end{cases} \tag{2.3}$$

As can be seen from the above equation, this method is computationally-wise and memory capacity-wise more expensive than just a combination of the Bob and Weave de-interlacers, it requires one field reference and a 7-tap filtering.

2.2.1.2 Non-linear methods

The next step that improves the quality is to take the motion information into account when filtering the pixels. This can be realized through methods that are motion or detail adaptive. Motion adaptive algorithms position themselves in between non-motion compensated algorithms and motion compensated ones, in terms of cost and performance. In general, motion detectors compute the difference between (parts of) two field references. This difference is usually low-pass filtered and rectified to increase the reliability of the signal [34]. The motion detector is used to switch or fade between two or more processing modes. As we have seen from the weave and bob de-interlacers, a de-interlacer might be optimized for certain type of video material, e.g. involving a lot of motion or mainly stationary. Bock [50] proposed a version of this reasoning that provides fading between these two modes of operation according to the next equation:

$$F_{BOCK}(\vec{x},n) = \begin{cases} F_I(\vec{x},n), & y \bmod 2 = n \bmod 2 \\ \alpha F_{ST}(\vec{x},n) + (1-\alpha)F_{MOT}(\vec{x},n), & \text{otherwise} \end{cases} \tag{2.4}$$

In this equation, α is the fading factor, provided by the motion detector, F_{ST} and F_{MOT} are the functions describing the static and moving picture parts, respectively. Apart from motion detector, this method requires computation of both, F_{st} and F_{mot} functions. Mixing of the two outputs is also required.

An extension to Bock's idea is to fade between more than two interpolators. Filliman et al. [51] designed an IC used in television that implements this idea. The de-interlaced output has two sources, the high-frequency component and the low-frequency component of the input signal. The fading (again, controlled by the motion detector) is applied to the low frequency component.

$$F_{FIL}(\vec{x},n) = \begin{cases} F_I(\vec{x},n), & y \bmod 2 = n \bmod 2 \\ F_{HF}(\vec{x}+\vec{u}_y,n) + \alpha F_{AV}(\vec{x},n) + & \\ (1-\alpha)F_{LF}(\vec{x},n-1), & \text{otherwise} \end{cases} \tag{2.5}$$

where $F_{AV} = \frac{1}{2}(F_{LF}(\vec{x}-\vec{u}_y,n) + F_{LF}(\vec{x}+\vec{u}_y,n))$.

The motion detector proposed by Filliman et al. uses the field difference to determine the amount of motion. This implies rather straightforward and simple processing with modest cost and the usage of two field references. The presence of motion is quantized using eight levels (three bits). Apart from the motion detector, this approach implies the splitting of the input signal into the high and low frequency parts, HF and LF respectively. The LF part of the signal is calculated for the previous field using the FIR filter of the sixth order. The HF part is computed as the difference between the original signal and the LF part. Thereby, two field references are required in this part of the algorithm, which is matched with the needs of the motion detector. To compute the F_{AV}, a simple vertical filter (averaging) was applied on the pixels from two vertically neighboring pixel-lines. Mixing is performed in two stages, as defined by Equation 2.5. In the first stage, mixing of the LF components take place, similar to Bock's approach. In the second stage, the already mixed LF component is combined with the HF component.

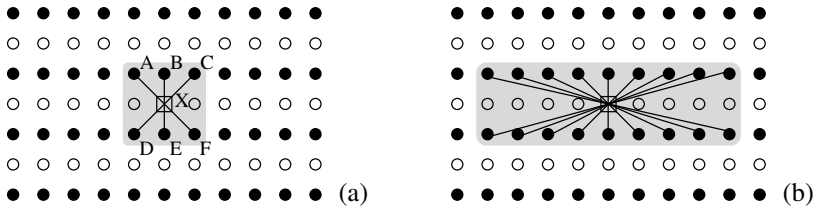


Figure 2.4: Two edge-dependent de-interlacing algorithms. The difference is in the aperture, i.e. the number of possible edges they can distinguish. The strongest edge is determined and the interpolation is performed along that edge. Pixels indicated as black circles are existing pixels from the current field and the white ones are to be interpolated. Pixel marked with square is the one currently being interpolated.

The edge-dependent de-interlacers (EDDI) perform directional interpolation. We illustrate its principals on the simple example published by Doyle et al. [52]. This method distinguishes between three edges, 45°, 90° and 135° by using the aperture of 3*2 pixels. The aperture and notion of particular pixels is illustrated in Figure 2.4a. Using the notation from the same figure, we present the interpolation equation of the missing pixel X.

$$X = \begin{cases} X_A, & (\mid A-F \mid < \mid C-D \mid) \wedge (\mid A-F \mid < \mid B-E \mid) \\ X_B, & (\mid C-D \mid < \mid A-F \mid) \wedge (\mid C-D \mid < \mid B-E \mid) \\ X_C, & \text{otherwise} \end{cases} \tag{2.6}$$

where $X_A = (A+F)/2, X_B = (B+E)/2, X_A = (C+D)/2$ and the pixels used in formula are the ones as indicated in Figure 2.4a. In the improved version, X_B is replaced by the vertical temporal median filter such as:

$$F_{VTM} = \begin{cases} F_I(\vec{x}, n), & y \bmod 2 = n \bmod 2 \\ \text{med}(F_I(\vec{x} - \vec{u}_y, n), F_I(\vec{x} + \vec{u}_y, n), F_I(\vec{x}, n-1)), & \text{otherwise} \end{cases} \tag{2.7}$$

Where the med operator performs the three-tap median. This is the simplest form of the vertical temporal median filter. It can be shown that this method automatically switches to the intra/inter field interpolation on a pixel-basis. Vertical temporal median filtering was also used in a commercially available chip [53]. In order to enhance the quality, more angles can be taken into account. In this case, the solution becomes more expensive since the aperture has increased and the number of operations needed to arrive at the strongest edge increases accordingly. One such approach (illustrated in Figure 2.4b) has been published in [54].

In order to estimate the cost of the EDDI-class algorithms, we first note that they are spatial algorithms (if the vertical temporal filtering is not applied). The support they require usually fits in two pixel-lines. The horizontal aperture determines the quality and linearly increases the need for HW resources. Per edge, three absolute differences need to be calculated, two comparisons and one logical operation need to be performed. The interpolation is usually a simple two-pixel averaging performed along the strongest edge. In the improved version, which uses vertical temporal median filter such as the one indicated by Equation 2.7, the algorithm requires an additional field reference.

2.2.1.3 Motion compensated methods

The idea behind motion compensated de-interlacing is to improve the correlation between pixels used in filtering. With application of motion estimation, the filtering is performed along the motion trajectory. Thereby, for many non-motion compensated de-interlacing methods, its motion compensated counter part exist as well. Replacing the pixels $F_I(\vec{x}, n)$ by $F_I(\vec{x} + m\vec{D}(\vec{x}, n), n + m)$ converts a non-motion compensated method into the motion compensated (MC) one. For example, MC field repetition, MC field averaging, MC vertical temporal filtering, MC median filtering and others are known de-interlacing algorithms. For these methods, we must assume the existence of motion vectors without explaining how we obtain those. Subsection 2.4 will provide insight into the most popular motion estimation techniques. There we shall also asses the cost associated with motion estimation.

This subsection discusses the particular methods that cannot be directly derived from the non-MC algorithms. The fundamental problem here arises when the motion vector does not point to an existing pixel on the interlaced grid. In the horizontal domain, this is not a problem since we can apply the sampling rate conversion theory. In the vertical domain, the theory cannot be applied since the demands of the sampling theorem are not satisfied.

To cope with this problem, we could neglect the theory and interpolate even in the vertical domain [35]. Woods et al. [36] improve this approach by extending the motion vector to the pre-previous field. They check then if the vector points to the vicinity of an existing pixel in either of these two fields. Only if this is not the case, spatial interpolation in the previous field is performed. The drawback is that the algorithm assumes linear motion vector field over the two-field period, which is not always the case. The HW costs are increased since the third field reference is needed as well.

Another approach is to use the previously *de-interlaced* frame in the interpolation and directly apply the sampling rate conversion theory [55]. Since this approach is recursive, the errors from the previously de-interlaced frame propagate. This is the biggest drawback of this method. To prevent error propagation, methods have been proposed [55, 56]. Median filtering such as in the following equation proved to provide good results.

$$F_O(\vec{x}, n) = \text{med} \begin{cases} F_O(\vec{x} - \vec{D}(\vec{x}, n), n - 1), \\ F_I(\vec{x} - \vec{u}_y, n), \\ F_I(\vec{x} + \vec{u}_y, n), \end{cases} \tag{2.8}$$

The median filter used in the above method can cause alias. To improve on that, Adaptive-Recursive (AR) de-interlacing was proposed in [57]. AR de-interlacing is defined with the following equation:

$$F_O(\vec{x}, n) = \begin{cases} kF_I(\vec{x}, n) + (1 - k)F_O(\vec{x} - \vec{D}(\vec{x}, n), n - 1), & (y{+}n) \bmod 2 = 0 \\ pF_A(\vec{x}, n) + (1 - p)F_O(\vec{x} - \vec{D}(\vec{x}, n), n - 1), & \text{otherwise} \end{cases} \tag{2.9}$$

where k and p are adaptive parameters and $F_A(\vec{x}, n)$ is the output of any (preferably simple) de-interlacing algorithm. Reference [57] explains how to calculate p and k.

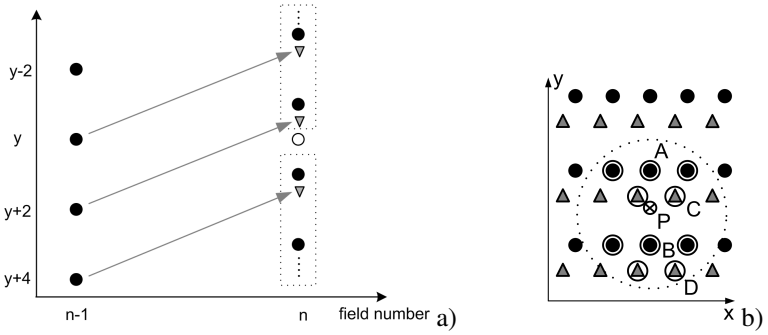


Figure 2.5: Picture (a) illustrates the essence of the 1D GST based de-interlacing. Picture (b) shows the filter support used in 2D GST based de-interlacing. In order to de-interlace the missing pixels (white circles), two sets of pixels are used: the original pixels from the picture number *n* (black circles) and the motion compensated ones taken from the picture number *n-1* (triangles).

We conclude the de-interlacing section with the algorithm based on the Generalized Sampling Theorem. According to the sampling theorem, a bandwidth-limited signal with a maximum frequency of $0.5f_s$, can exactly be reconstructed after sampling at a frequency higher than f_s (Nyquist criterion). In 1956, Yen [58] proposed a generalization of the sampling theorem (GST), proving that a signal with a bandwidth of $0.5f_s$ can be reconstructed from *S* independent sets of samples, all obtained by sampling the signal at f_s/S with a phase shift. Put in the context of video processing, Delogne et. al have proposed the solution for de-interlacing based on the GST [37, 38]. As shown in Figure 2.5a for de-interlacing, the first of the two required independent sets of pixels is obtained by shifting the pixels from the previous field (picture number *n-1*) over the motion vector towards the current picture *n*. The second set contains pixels from the current field. This filter does not use previously interpolated field, which prevents error propagation. The filter coefficients depend on the fractional values of the vertical component of the motion vector.

The robustness of this method can be improved if the outliers in the motion vector field are replaced by the more probable values. The most common method to that is the median filtering. Bellers and de Haan [59] show that the median filtering degrades the quality of the motion vector field too much in the areas with correct motion vectors. The authors proposed to apply the modification only in the areas where motion vectors contain close to critical velocity.

A further improvement has been published by Ciuhu and de Haan in [21]. This algorithm computes the de-interlaced pixel *twice*. A pixel is interpolated using the previous and the current fields as well as using the current and the next field. Theoretically, these two values should be identical. If their difference is higher than certain threshold, this is the case of unreliable motion vector candidate. In such a case a fall back scenario can be

applied.

The 2D generalized sampling theorem based de-interlacing [21] uses the 2-D pattern of pixels used in GST-based de-interlacing. Ten pixels contributing to de-interlacing are selected from the circular 2-D neighborhood of the pixel being interpolated (see Figure 2.5b). Taking into account the discussion about the robustness of the motion vector field from the previous paragraph, we conclude that the 2D GST based de-interlacing is a three-field algorithm where the motion vector candidates are used to fetch motion compensated pixels from the previous field and the next field in order to estimate the motion for the current field. Using the best-matching motion vector candidate, the missing pixels are interpolated using the 10-tap filter.

Apart from the fact that the algorithm uses three picture references, we note that this is a compute intensive algorithm. For each motion vector candidate, a 10-tap filtering has to be performed twice, for previous-current and for the current-next input field combination. During the de-interlacing, the complete process is repeated for the best motion vector candidate. Thereby the dominant part of the algorithm is pixel interpolation. More information about the computational complexity of this algorithm can be found in [60]. The author made an effort to simplify this algorithm. The interested reader may find more detailed information in [20].

2.2.2 Picture-rate up-conversion

Modern television sets display video stream at rates that range from 50 to 120 Hz, while the source picture-rate can be 50 or 60 Hz for video material and 24, 25 or 30 Hz for film material. Clearly, a high quality conversion of signals from one format to another is of great importance. In order to generate new pictures (see Figure 2.6), we need to perform a *temporal* interpolation. Directly interpolating pictures according to the rules of the sampling theorem [61] does not produce good results unless the temporal interpolation is performed along the motion trajectory [34]. The human visual system (HVS) is capable of tracking a moving object. In the case of motion, the HVS compensates for the motion,

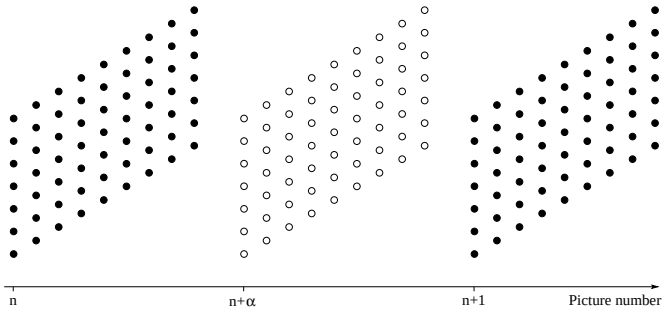


Figure 2.6: The problem statement of picture-rate up-conversion. Pixels marked as black circles are transmitted. Pixels marked as white (empty) circles have to be interpolated.

creating a stationary object at the retina. Therefore, if the temporal interpolation is not performed along the motion trajectory, echos or aliases of the moving object appear at the retina. This either leads to motion judder or blur [34].

There is no upper limit to the temporal frequencies in video that the HVS can resolve because the eye transforms them to low frequencies. Without an upper limit it is impossible to design an interpolating low-pass filter that does not cause blur. However, there is a limit to the maximum perceivable temporal frequency *at the retina*. For the average viewer, it is about 50 Hz (this value depends on the brightness of the display and the viewing angle) [34]. This implies that it is possible to interpolate in the temporal dimension *only after compensating for motion*.

The classical approaches neglect this aspect of temporal interpolation and therefore produce poor results with respect to the motion portrayal. We start this section with these simple algorithms, show their drawbacks and present more advanced methods that perform the interpolation along the motion trajectory.

2.2.2.1 Linear methods

All linear methods for up-converting a video sequence by a factor of k/l, where k and l are natural numbers, can be defined with the following set of equations [34]. We shall denote the input samples with n and the output with $n\prime$. The up-conversion with a factor of k starts with the appropriate zero-stuffing. The zero-stuffed signal F_z is defined with the following equation:

$$F_z(\vec{x}, \tfrac{a}{k}) = \begin{cases} F_I(\vec{x}, n), & \tfrac{a}{k}\text{=1, 2, 3, } \ldots \\ 0, & \text{otherwise} \end{cases} \tag{2.10}$$

followed by the low-pass filtering:

$$F_{LP}(\vec{x}, a) = \sum_m F_z(\vec{x}, a+m)h(m), m \in \mathbb{Z} \tag{2.11}$$

In the above formula, $h(m)$ denotes the impulse response of the temporal filter. The last step is the decimation with a factor l:

$$F_O(\vec{x}, n\prime) = F_{LP}(\vec{x}, la) \tag{2.12}$$

Changing k, l and $h(m)$ leads to various linear up-conversion techniques including the simplest one, picture repetition. The impulse response is defined with the following equation:

$$h_{REP}(m) = \begin{cases} 1, & m \in \{-1, 0\} \\ 0, & \text{otherwise} \end{cases} \tag{2.13}$$

The picture repetition algorithm practically does not require any processing and it needs one picture reference. Surprisingly enough, this approach is actually also used in professional equipment for displaying film material on a 50 or 60Hz television. Since film is usually recorded with a 24 Hz camera, in order to be displayed on a 50Hz television, two

Figure 2.7: Different artifacts of picture-rate up-conversion. From top-left to bottom-right: Non motion compensated picture averaging, motion compensated picture averaging, static median filtering, dynamic median filtering, cascaded median and the central weighted median filtering.

steps are applied. The first step is to accelerate the material to 25 Hz and the second one is to display each picture twice (this method is also called 2-2 pull down). In the case of 60Hz television, pictures are repeated two or three times, alternating (2-3 pull down). By doing this, the achieved rate is exactly 60 fps (frames per second), or 60Hz. Picture repetition has also been used in the early ICs for flicker-free television [53].

In order to produce better results, we return to the sampling theory. As the theory prescribes, after band limiting and zero-stuffing the signal, we apply low-pass filtering in which the empty pixels get meaningful values. Unlike in the picture repetition algorithm, we should take infinitely many picture references and a low-pass filter should have infinitely many taps in order to arrive at the ideal impulse response of the filter, the sync function. Practical implementation aspects impose limitations to the number of used picture references and filter design, which leads to feasible filters. Here we exemplify it with the simple weighted picture averaging that uses only two picture references:

$$F_{AVG}(\vec{x}, n+\alpha) = (1-\alpha)F_I(\vec{x}, n) + \alpha F_I(\vec{x}, n+1), \quad 0 \leq \alpha \leq 1 \tag{2.14}$$

This linear method uses coefficients that are reversely proportional to the temporal distance between the original pictures and the interpolated one. In the case of picture rate doubling (counterpart of the 2-2 pull down for example), $\alpha = 1/2$, the impulse filter response is defined with:

$$h_{AVG}(m) = \begin{cases} 1, & m = 0 \\ 1/2, & m \in \{-1, 1\} \\ 0, & \text{otherwise} \end{cases} \tag{2.15}$$

Compared to the coefficients of the impulse filter of the picture repetition, we see that the number of non-zero coefficients has increased. This means that the number of echos in the interpolated picture has also increased, which results in a blurred picture. The blurring with visible echos is illustrated in Figure 2.7.

The reason why this blurring occurs we have to seek in the way the interpolation was performed. In this case, it was performed along the temporal and not along the motion axis. Therefore, using more picture references does not help, it will only increase the number of echos. In order to achieve good results, filtering must be performed along the motion trajectory. This implies the usage of motion-compensated techniques.

2.2.2.2 Motion compensated methods

The simple picture-rate up-conversion algorithms like picture repetition or picture averaging produce visual artifacts like motion judder and blur while motion compensated algorithms enhance the up-conversion quality. Picture-rate up-conversion is independent from the motion estimation algorithm used. In this subsection, we assume the existence of motion estimator and leave the analysis of various estimators for Subsection 2.4.

After motion estimation is performed, to every pixel identified with spatial position $\vec{x}$ and temporal position *n*, a displacement vector $\vec{D}(\vec{x}, n)$ has been assigned. Based on the

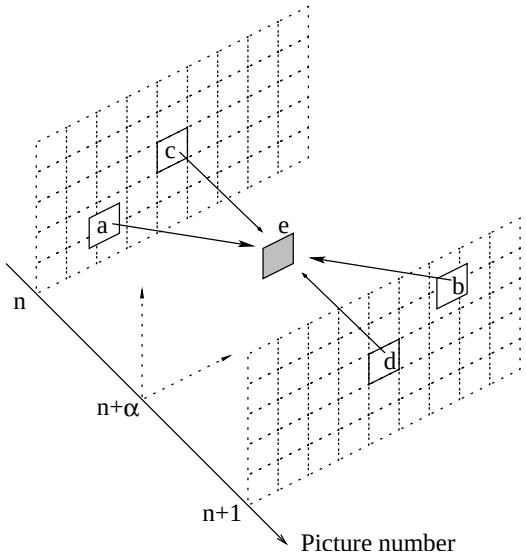


Figure 2.8: The pixels from the previous *n* and current *n+1* picture, accessed in the motion compensated up-conversion. 'a' and 'b' denote the pixels compensated with the current motion vector candidate while 'c' and 'd' are non-motion compensated positions in the previous and current picture.

motion vector field calculated at the temporal position $n + \alpha, 0 \leq \alpha \leq 1$ as well as the luminance values of the pixels available at the picture numbers *n* and *n+1*, new pixels can be interpolated at picture $n + \alpha$. Figure 2.8 illustrates the process of motion compensated creation of the pixel 'e' in the interpolated picture (picture number $n + \alpha$). This picture is applicable for all the methods to follow.

All linear non-motion compensated methods discussed in this subsection have their motion compensated counterpart. We begin discussing those methods and afterwards continue with the non-linear MC methods, which bring further performance improvement.

2.2.2.3 Linear motion compensated methods

Using motion estimation, the corresponding pixels on the motion trajectory in the previous and current picture are determined (motion compensated pixels 'a' and 'b', respectively). The simplest motion compensated algorithm called motion compensated pixel repetition (MCPR) is based on the first order linear interpolation and it uses only one of the two mentioned pixels (pixel 'a' or 'b'). It is defined with:

$$F_{MCPR}(\vec{x}, n + \alpha) = F_I(\vec{x} - \alpha\vec{D}, n), \quad 0 \leq \alpha \leq 1 \tag{2.16}$$

The next improvement is to use higher order interpolation linearly combining pixels from successive pictures. This method is called motion compensated pixel averaging (MCPA)

and averages motion compensated pixels 'a' and 'b'. Formally, MCPA is defined with:

$$\begin{aligned} &F_{MCPA}(\vec{x}, n+\alpha) = \\ &1/2\left\{F_I(\vec{x}-\alpha\vec{D}, n) + F_I(\vec{x}+(1-\alpha)\vec{D}, n+1)\right\}, \\ &0 \leq \alpha \leq 1 \end{aligned} \tag{2.17}$$

Compared to MCPA, MCPR has the modest advantage of using just one picture reference. This advantage is limited since it is only applicable in case of an external motion estimator, which is often not the case. MCPR practically does not require any processing. MCPA requires a bit more processing, i.e. a simple averaging at pixel rate. For both methods, the dominant HW cost goes to fetching the right pixels pointed out by the motion vector with possible application of the scaling factor α.

Thanks to the application of motion compensation, which enables interpolation along the motion trajectory, the blurring in moving picture parts is drastically reduced compared to the non-motion compensated variants of these algorithms. Figure 2.7 illustrates significant quality improvement that MCPA brings compared to its non motion compensated variant. However, the problems that all motion compensated methods face are the errors in the motion vector field. These errors cause the wrong pixels to be fetched and used in the interpolation. For example, if we look at the compensation of a video sequence that contains a subtitle, the non-zero motion vectors might also penetrate into the regions with subtitles. This leads to the artifacts as illustrated in Figure 2.7.

2.2.2.4 Non-linear motion compensated methods

The way to tackle this problem is to generate a number of likely pixel candidates for temporal interpolation and remove the outliers by using the order statistical filtering. In the literature, we find a number of solutions [40–43, 62–65]. We distinguish two major options among the non-linear motion compensated methods.

According to the first option, the promising candidates can be pointed out by the best-matching motion vector or a zero vector (more conservative approach). Actually, the more complete pixel candidate set consists of the pixels pointed out by all the motion vectors from the evaluation set (including the zero candidate) [39]. The order statistical filter will then select one pixel from this relatively large and complete pixel candidate set.

The second option would be to choose from the list of small position variations centered around the motion compensated pixel [64, 65]. The non motion compensated variant of this approach would be the list of variations centered around the zero vector [64]. In the coming text, we will cover both options in more detail.

The first option: In order to find an outlier, at least three candidates are required such that a majority prevails and eliminates the outlier. There are four immediate possibilities for these candidates, MC compensated pixels from the previous ('a') or the next ('b') picture or the non-MC pixels from the previous ('c') or the next ('d') picture. These four candidates provide the bases for a couple of non-linear methods that we will present in

the following paragraphs.

In order to cope with the previously mentioned subtitle artifacts, a conservative 3-tap median filter which has two non-MC compensated inputs and one MC input has been proposed [43]. This filter, called the *static median filter*, is defined with the following equation:

$$\begin{aligned} &F_{MCSTA}(\vec{x}, n+\alpha) = \\ &\text{med } \{F_I(\vec{x}, n),\ F_I(\vec{x}, n+1),\ F_{MCAVG}(\vec{x}, n+\alpha)\}; \\ &0 \leq \alpha \leq 1 \end{aligned} \tag{2.18}$$

As the number of non-motion compensated pixels outnumbers the motion compensated pixels (only one), it is clear that this filter is a conservative one that switches to one of the non-motion compensated pixels in the case of a spurious motion vector. Its biggest problem occurs with detailed moving picture parts (like textures, or very thin objects) where the difference between non-MC pixels is very large causing that the filter output frequently switches among the three inputs. The performance of the static median filter is illustrated in Figure 2.7. The subtitle is practically artifact free but the leaves of the palm-tree (detailed texture) show prominent artifacts as a result of too-conservative median filtering.

To address the mentioned problem, another 3-tap median filter has been proposed, where the number of MC pixels outnumbers the number of non-motion compensated ones. This filter is called *dynamic median* filter [39, 40]. The dynamic median filter is defined with the following equation:

$$\begin{aligned} &F_{MCDYN}(\vec{x}, n+\alpha) = \\ &\text{med } \{F_I(\vec{x}-\alpha\vec{D}, n), F_I(\vec{x}+(1-\alpha)\vec{D}, n+1), F_{AVG}(\vec{x}, n+\alpha)\}; \\ &0 \leq \alpha \leq 1 \end{aligned} \tag{2.19}$$

where F_{AVG} is the non-motion compensated picture average defined with Equation 2.14. In case of the accurate motion vector, the MC pixels will be ranked next to each other and the output of the filter will be one of them. This is the reason why this filter performs better than the static median in case of reliable motion vector and improvements are visible in the case of input material is detailed texture. If the motion vector is not reliable, the idea is that the MC pixels will be significantly different such that the non-motion compensated average sample will be in the middle and therefore be selected as the output of the median filter. Sometimes this does not work, as illustrated in Figure 2.7. Contrary to the static median filter the subtitle suffers again from artifacts while the detailed texture is correctly interpolated (since the correct velocity has been detected by the estimator).

To asses the HW cost of these two median based filters, we note that additional to the cost of MCPA, these methods also require fetching the non-motion compensated pixels from the pre-determined positions. Further, as a final step, the 3-tap median is calculated. Clearly, the HW cost for the median based filtering is somewhat higher than the cost of MCPR and MCPA.

An alternative to the median filtering is mixing the MC averaging and temporal averaging [41]. The mixing is controlled by the local reliability of the motion vector field,

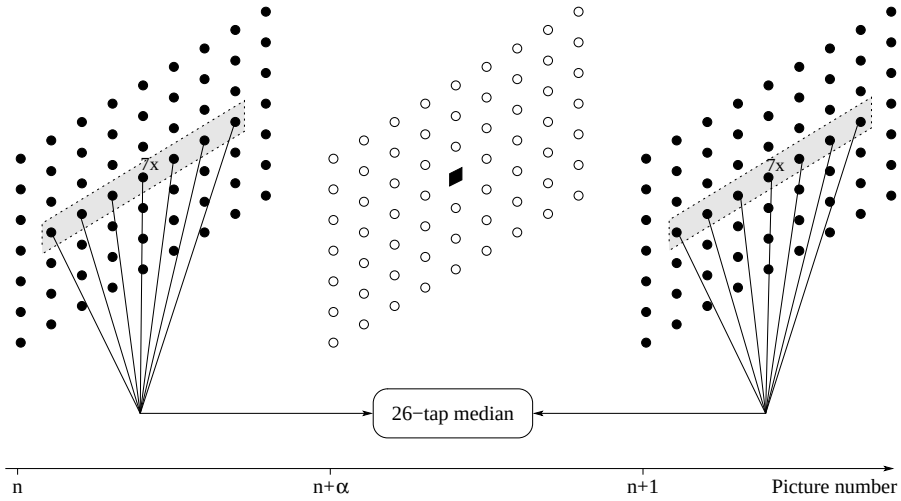


Figure 2.9: Illustration of the algorithm published in [64] that uses weighted median filter. Pixel marked with square is currently being interpolated.

denoted with k as defined by the following equation.

$$F_{MIX}(\vec{x}, n+\alpha) = (1-k)F_{AVG} + kF_{MCPA} \tag{2.20}$$

As proposed in [41], the reliability k is controlled with two differences, between the motion compensated pixels and the second one between the non-motion compensated pixels. Depending on the control, the performance resembles the static or dynamic median filter.

The static median filter performs good in the stationary areas, but not in the case of detailed textures. Dynamic median performs much better in the latter case assuming that the motion vector is correct. To benefit from both the static and the dynamic median filtering, they can be combined in order to arrive at more robust up-conversion [42]. A practical implementation, involving two-stage median filtering, called cascaded median has been published in [43]. Cascaded median is defined with the following equation:

$$F_{CMED} = \text{ med } \{F_{MCDYN}, F_{MCSTA}, F_{MIX}\} \tag{2.21}$$

This algorithm requires calculation of all three used methods and performs additional three-tap median. Figure 2.7 illustrates the benefits of this method that joins the good points of static and dynamic median filtering.

The second option: As we mentioned in the introduction of the non-linear MC methods, an alternative to the aforementioned methods is a median filter, using input pixels centered around the motion compensated pixels. By doing that, we can correct small errors in the motion vector field. In the case where motion estimation is not applied, the same holds with the only difference that the median filter is centered around the zero

vector. Blume et al. [64] extend this idea and propose the so-called Central Weighted Median (CWM) filter. In the weighted median filter, some input samples are repeated multiple times, thereby giving those samples higher specific weight. The median filter used in [64] has 26 taps and the highest weights get pixels originating from the same spatial location and two temporally neighboring locations as the pixel currently being interpolated. If with $F_{CWM}(\vec{x}, n+\alpha)$ we denote the pixel being interpolated, the pixels having highest weights are $F_I(\vec{x}, n)$ and $F_I(\vec{x}, n+1)$, where F_I denotes the luminance values of the input pixels. Figure 2.9 further clarifies this method.

To clarify the benefits of the weighted median filtering, we first observe that the simple (unweighted) median filter is already sufficient in interpolation of a moving edge. The displacement of the edge in a picture period should not exceed the aperture of the median filter. The main drawback of the simple median filter is the degradation of fine details in case of stationary sequence while the central weighting can preserve those.

The CWM filter is good for sequences with very limited or no motion (motion which does not go beyond the limits of the filter support). However, as Figure 2.7 illustrates, it is still not good enough for areas with motion. As we mentioned before, centering the CWM filter around the best-matching motion vector significantly improves the performance. In this case, the *motion compensated* CWM (MCCWM) method should indeed be able to correct small errors in the motion vector field. The details about MCCWM approach can be found in the work of Franzen [65].

Even though this method uses a modest filter support, its large 26-tap median filter is its main drawback. In spite of the methods for efficient median filter implementation [65], a 26-tap median filter is still much more expensive than a 3-tap median filter. Further, any change in the filter support (for example, adding more pixels) changes the median filter tap count which directly impacts the implementation.

The CWM summarized here has an aperture that extends only in the horizontal domain. A 2D central weighted median has also been published by Blume [64]. The typical aperture is cross-shaped (no diagonal neighbors).

2.2.2.5 Occlusion-aware up-conversion

In the previous motion-compensated methods, we have mentioned the problem of inaccuracies of the motion vector field. All the mentioned methods use a single motion vector field computed based on two successive pictures. By doing so, we are unable to cope with the fundamental problem of occlusions [22]. The brief description of the occlusion problem is that in the occluded areas, we are unable to determine whether we should compensate using the foreground or background motion vector and whether we should use pixels from the previous, or from the next picture. The motion estimator typically shows a tendency to extend the foreground vectors to the background causing the so-called *halo effect* [66]. We will address this problem in Subsection 2.4.

Apart from having three picture references and performing the motion estimation on two picture pairs, the computations on the motion vector field are quite intense. A detailed cost analysis of this algorithm is available in [67].

2.2.3 Spatial Scaling

Spatial scaling and resolution up-conversion make the third major function member of the video format conversion pillar. Recently, the need for high-quality up-scaling is on the rise, mostly thanks to the boom of HDTV. There is enormous amount of video material available in lower resolutions like SDTV (PAL and NTSC). Further, a vast majority of broadcasters still emit their program in SDTV resolution. At the same time, people are abandoning their CRT displays and going for large panels with increased resolution. The need for high-quality up-scaling is evident. As in the previous sections, we will begin this section with classical linear theory applied to the spatial up-scaling problem.

2.2.3.1 Linear up-scaling

In order to scale the input signal with a rational factor $\frac{u}{d}, u, d \in \mathbb{N}$, the signal first needs to be up-sampled by a factor of u and then down-sampled by a factor of d. In order to up-sample the signal by a factor of u, two steps are needed [68, 69]. The first step is zero stuffing to arrive at the new sampling rate (according to the up-sampling factor u). The second step is interpolating low-pass filtering that brings the meaningful values to the zero samples. It should also suppress all the frequencies from the spectrum of the output signal that are above the Nyquist frequency. We illustrate this low-pass filtering on the example of the 1D video signal, e.g. horizontal or vertical dimension. Formally, the up-sampled, continuous signal is defined as the convolution of the sampled input video signal and the continuous impulse response of the filter:

$$F(x) = \sum_k x(k)h(x-k) \tag{2.22}$$

According to the theory, for the ideal suppression of any frequency above the Nyquist rate, the continuous impulse response $h(k)$ should be the sinc function. The sinc function however uses infinite number of pixels. For practical usage, we apply a windowing function to limit the filtering support or opt for the filters with simpler impulse responses (which inevitably leads to alias). In the literature, we find the following popular practical choices. The impulse response of the zero-order filter is defined with the following equation:

$$h_0(x) = \begin{cases} 1, & -0.5 < x \leq 0.5 \\ 0, & \text{otherwise} \end{cases} \tag{2.23}$$

Better quality is achieved by using the first order filter whose impulse response is obtained by convolution of the aforementioned zero-order impulse response:

$$h_1(x) = h_0(x)h_0(x) = \begin{cases} 1- \mid x \mid, & \mid x \mid \leq 0.5 \\ 0, & \text{otherwise} \end{cases} \tag{2.24}$$

This filter is in the 2D/3D world known as the bilinear/trilinear interpolation. Its coefficients are reversely proportional to the distance between the pixel being interpolated and its neighboring pixels.

Finally, the signal $F(x)$ is down-sampled with a factor of d. Down-sampling begins with the low-pass filtering. The frequency spectrum of the input (up-sampled) signal has

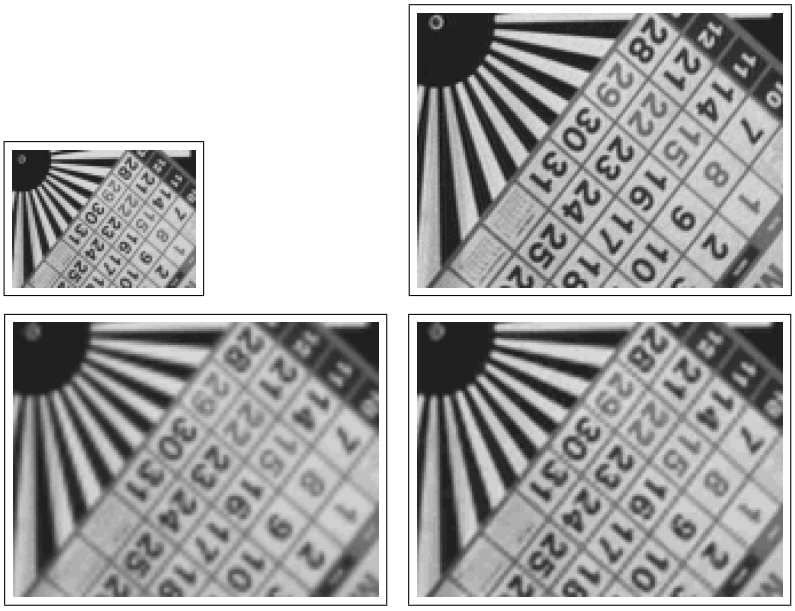


Figure 2.10: Different artifacts of spatial up-scaling and resolution up-conversion. From top-left to bottom-right: The down-scaled version of the original picture (drawn to scale), the original picture, up-scaled using bi-linear interpolation, up-scaled using advanced resolution up-conversion.

to be limited to match the new (lower) sampling rate. The lower sampling rate implies lower maximal representable spatial frequency. Thereby, limiting the spectrum to the half of the new sampling rate (Nyquist frequency) prevents the alias in the down-sampled picture.

The method that is more frequently applied for scaling with a rational factor in the television applications is the poly-phase filtering [68, 69]. Poly-phase filtering is filtering that merges the up-sampling and down-sampling and the thrown away samples are not computed. This filtering is performed such that the coefficients depend on the position (or the phase) of the output pixel. By doing so, the coefficient set is optimized for each phase of the output pixel such that only the multiplications with the non-zero coefficients are computed. Clearly, this method offers a tradeoff between number of computations and additional control. The control can be realized by using the LUT holding a number of coefficient sets. A practical implementation (IC) that uses 24-tap poly-phase scaling has recently been reported for scaling even beyond HDTV resolution (1920*1080) [70].

2.2.3.2 Resolution up-conversion

Looking at the frequency side, linear up-scaling methods do not extend the high frequency part of the spectrum since the resolution has not increased. In this subsection, we shall discuss some advanced resolution up-conversion algorithms that go beyond the linear

theory and introduce new spectral components [44, 71–75]. Figure 2.10 illustrates the quality improvement brought by the advanced resolution up-conversion method.

In general, we distinguish two categories of the resolution up-conversion algorithms. The algorithms from the first category, optimize objective performance (usually expressed using the mean square error, MSE) aiming at picture restoration. Algorithms from the second category have the subjective performance as the target. We shall provide four algorithms from the first category (Kondo et al. [44], Atkins et al. [71], Plaziac [72], Li and Orchard [73]) and a combination of two algorithms from the second category [74, 75].

Methods that aim at picture restoration

In the method proposed by Kondo et al. [44], the coefficients used in the interpolation depend on the local picture content. Based on the large amounts of video material, picture content is classified and for each class, different set of coefficients is available. This method requires the availability of both original and up-scaled video material. Based on this material, the optimal filter coefficients are determined by means of minimal mean square error (MSE) criterion. This is extremely compute intensive method that fortunately has to be performed only once, offline.

The algorithm consists of two steps, classification and filtering. Both steps use the same aperture of 3*3 pixels, illustrated in Figure 2.11a. Classification uses the so-called Adaptive Dynamic Range Coding (ADRC) algorithm that drastically reduces the number of total classes [76]. In the ADRC approach, each pixel in the aperture is encoded with a single bit. If we denote this bit with Q, the encoding is defined with the following equation:

$$Q = \lfloor (F_{IP} - F_{MIN})/((F_{MAX} - F_{MIN})/2) \rfloor \tag{2.25}$$

where F_{MIN} denotes the minimal, and F_{MAX} the maximal value in the aperture, F_{IP} denotes the original, input pixel and $\lfloor \cdot \rfloor$ denotes the floor operation. By repeating the previous equation for each of the nine pixels within the aperture, we obtain the 9-bit class identifier. The second step, filtering is applied based on the same filter support as the support used in the classification process, 3*3 pixels. The coefficients depend on the x and y coordinates of the appropriate input pixels and the determined class c.

To analyze the computational complexity of this algorithm, we begin with remark that it is spatial-only and pixel-based which means that each pixel is treated differently, depending on its own value and its 8 nearest neighbors. The classification step starts with determining the minimal and the maximal value within the 3*3 pixel aperture. The division from equation 2.25 can be substituted by a comparison, which reduces the computational complexity. If $F_{IP} - F_{MIN} > (F_{MAX} - F_{MIN})/2$ then the result is 1, otherwise 0. This comparison is repeated for all nine pixels within the aperture and 9-bit value is produced. This value is actually an address of the LUT where the filter coefficients are stored. The second step, a 9-tap filtering, is performed per each output pixel. All the coefficients used in the interpolation depend on the local picture content. Note that this method requires the LUT with 2^9 entries, each entry holding one filter coefficient. Such a large LUT increases the cost of implementation. Recently, methods have been proposed

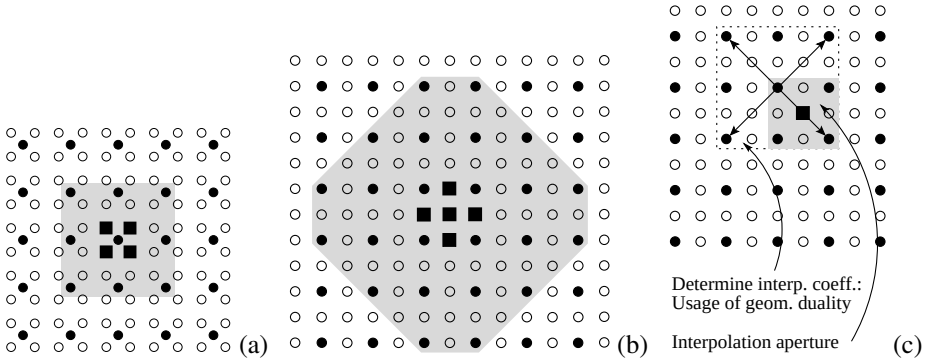


Figure 2.11: The apertures used in various resolution up-conversion algorithms. Picture (a) illustrates a 3*3 support used in the Kondo method [44]. Picture (b) illustrates the aperture used in neural network proposed by Plaziac [72]. Picture (c) illustrates the algorithm proposed in [73] that performs the training of the coefficients on-the-fly. As before, the black pixels in the shaded area mark the existing pixels used in the interpolation while the square-shaped black pixels are currently being interpolated.

to reduce the size of the LUT under minor picture quality loss [77, 78]. The LUT can be reduced by two orders of magnitude, which leads to cost-effective implementation of this method.

A method also involving classification and aiming at minimal MSE has been proposed by Atkins et al. [71]. According to this method, the output is computed as the weighted sum of the outputs of number of linear filters based on classification. The filter aperture is identical to the one from Kondo, 3*3 pixels, although the classification algorithm is more complex [71, 79]. The Expectation Maximization algorithm [79] provides a probability that a vector belongs to certain class. The number of classes that this algorithm uses is fixed and is around 100.

Analyzing the computational complexity of this algorithm we first note that its advantage compared to Kondo's approach is that the number of classes is fixed (it depends on the aperture in Kondo's method). However, the number of classes is not lower in case the mentioned class number reduction methods are applied to Kondo's method. Thereby, the LUT is larger. Even bigger drawback of this approach is the computational complexity. In order to compute the output pixel, all the linear filters (reflecting all 100 classes) have to be computed followed by the weighted summing. In the Kondo's approach, after the classification, only one filtering has to be performed. Finally, the weighting factor is a continuous function. Coarser quantization might lead to the performance degradation.

Another example of a method that needs a training, is the neural network based up-scaling. In the neural network approach, the video material is implicitly classified in the process of learning (training). Plaziac [72] proposes a neural network for de-interlacing and spatial up-scaling. This network has multiple inputs and outputs and multi-cell hidden layer. The output of this non-linear approach is defined with the following set of two

equations:

$$\vec{h} = tanh(\vec{w}_0\vec{F}_I + \vec{b}_0) \tag{2.26}$$

where the $tanh$ is a non-linear function, w_0 is the weight matrix between the hidden layer and the N input pixels F_I and b_0 is the bias vector for the hidden layer. The output pixels of the neural network, F_{NN} are described with the following equation:

$$\vec{F}_{NN} = (\vec{w}_1\vec{h} + \vec{b}_1) \tag{2.27}$$

where w_1 is the weight matrix between the output pixels and the hidden and b_1 is the bias vector for the output layer. The w_0, w_1, b_0 and b_1 parameters of the network have been determined in an off-line training process.

Based on the above equations, Plaziac proposed the method for spatial up-scaling that uses one picture reference. The up-scaling neural network has 24 input pixels, 16 cells in hidden layer and produces 5 output pixels. The aperture used here has a diamond shape as illustrated in Figure 2.11b. However, the network is flexible, meaning that the number of inputs can be changed. In spite of using such a large number of input pixels, more than one output pixel are produced. Since the computations can be shared, the overall computation effort per produced output pixel can be reduced. However, the computational requirements of this method are quite high due to equations 2.26 and 2.27. They involve matrix multiplication and the number of input pixels and the number of cells in the hidden layer are high.

Contrary to the above methods, the training can also be performed on-line. Li and Orchard [73] propose one such method. The interpolation coefficients are determined based on the minimal MSE algorithm executed on the original pixel grid. Compared to all previously mentioned non-linear algorithms, the number of classes is not limited. There are two major disadvantages of this method, the increased amount of calculations (which might impair the real-time implementation) and that no original up-scaled picture data were available for training (which might lead to the quality degradation).

This method can up-scale the input picture by the factor of 2^n, $n \in \mathbb{N}$, horizontally and vertically. Assuming an up-scaling factor of 2, horizontally and vertically, we shall illustrate and analyze the proposed method. Following the authors' notation, with $X_{i,j}$ we denote the original pixels and with $Y_{2i+k,2j+l}, 0 \leq k,l \leq 1$ the up-scaled ones. Fourth order interpolation is used to limit the operation count (see also Figure 2.11c) :

$$Y_{2i+1,2j+1} = \sum_{k=0}^{1}\sum_{l=0}^{1} \alpha_{2k+l} Y_{2(i+k),2(j+l)} \tag{2.28}$$

where $Y_{2(i+k),2(j+l)} = X_{i+k,j+l}$. The interpolating α coefficients are determined in an on-the-fly process using the feature of geometrical duality. They are determined based on the local window of $M * M$ original pixels. More details can be found in the cited paper, but for the purpose of this analysis it is enough to mention that the authors have chosen that $M = 8$. Geometrical duality (see also Figure 2.11c) basically means that the relationship between the diagonal neighbors from the original grid is the same as the

relationship between the interpolated pixel and its nearest diagonal neighbors from the original grid.

Analyzing the computational complexity of the mentioned approach, first we note that the algorithm requires only one picture reference and that interpolation is of the fourth order. However, as the authors themselves also conclude, the real computational complexity does not lie in the filtering itself but rather in determining the values of the coefficients. All four coefficients depend on the content of the local window of $M * M$ pixels. If the dimension of the local window is $M = 8$, as much as 1300 multiplications are needed to compute four α coefficients. To relax the computational requirements the authors go for a hybrid solution where only pixels that are near the edge are computed in the described manner and the rest are computed by using the simple bilinear interpolation. Whether a pixel is considered to be close to an edge or not is determined by comparing its local activity (local variance estimated from its four nearest neighbors) to a certain threshold. According to the authors, even though this reduction assumes computing the local activity per pixel, they achieved a speed-up by a factor of 7-20.

Methods that aim at subjective performance

All the non-linear methods mentioned so far minimize the objective error, expressed in MSE. Finally, we shall report on a method that optimizes the subjective performance. In the introduction part of this subsection we have mentioned that the drawback of the linear methods is that the high-frequency part of the spectrum is not extended. To improve the linearly up-scaled pictures, and add new spectral components, the methods described below use methods such as luminance transient improvement, LTI and peaking. The LTI and peaking will be described in more detail in Section 2.3.

Bellers [74] proposed to use peaking (filter that boosts high frequencies) on textured areas and luminance transient improvement, LTI (on edges) to improve the up-scaled pictures. Since LTI is a 1D technique, the authors propose a straightforward way to extend it in the 2D space - LTI is first applied in the vertical direction and then in horizontal. To achieve better results, in [75] was proposed to apply the LTI perpendicular to the edge. This involves detection of the edge orientation (implemented using Sobel operators) and rotation of the line along which the LTI operates. In order to reduce the complexity, an alternative edge orientation dependent LTI technique has been proposed. According to that proposal, a horizontal and vertical LTI are calculated in parallel. These two results are combined into a weighted sum, where the weights are controlled by the edge orientation detector. This approach was shown to give the best price-performance [34].

2.3 Video enhancement

The goal of video enhancement is to enhance the *subjective* (perceived) picture quality. As subjective quality may differ from person to person, many of the techniques summarized in this section should be controlled by the end user. This poses an important design constraint demanding a high degree of flexibility, additional to the high performance.

We divide the topic of this section into four parts, noise and coding artifacts reduction, sharpness enhancement, contrast enhancement and color reproduction enhancement.

2.3.1 Noise reduction

The need for noise reduction originates from the analogue world where the illumination of the scene may be insufficient, the transmission channel may not be properly dimensioned, receiving antennas can be too small, cables may be degraded over time, etc. In a television system, the typical entry points of the noise are in the picture sensor and/or in the transmission channel. The noise reduction problem statement is to find an estimate of the input video sequence, which has been corrupted by noise.

In general, noise and artifact reduction assumes that the signal is large compared to the artifact and the correlation between neighboring pixels in a picture sequence is high. This assumption leads to an estimate of the uncorrupted pixel by combining likely identically valued neighboring pixels. Denoting the pixels in the support with $S(\vec{x}, n)$), we summarize that it is possible (see also Chapter 3 of [34]):

• To use all the pixels in the neighborhood (standard neighborhood), $N_s(\vec{x}, n) = S(\vec{x}, n)$

• To only combine a predetermined fraction of the pixels in the support choosing the ones that differ least from the current pixel (K-nearest neighborhood). Formally, to define the k-nearest neighborhood, we start from the ordered support S_O containing s pixels:

$$\vec{S}_O(\vec{x}, n) = \{F_i \in S(\vec{x}, n) \mid F_1, F_2, \ldots F_s\} \tag{2.29}$$

where

$$\mid F_i - F(\vec{x}_c, n) \mid \leq \mid F_{i-1} - F(\vec{x}_c, n) \mid, i = 2, \ldots, s \tag{2.30}$$

where $F(\vec{x}_c, n)$ is the luminance value of the central pixel from the filter support. Using the above two equations, we define the K-nearest neighborhood as the subset of the ordered support S_O:

$$\vec{N}_{\text{KNN}}(\vec{x}, n) = \{F_i \in S_O(\vec{x}, n) \mid F_1, F_2, \ldots F_{K+1}\} \tag{2.31}$$

• To only combine pixels that differ less than a threshold value, sigma (σ), from each other. The sigma nearest neighborhood is defined with the following equation:

$$\vec{N}_{\text{SNN}}(\vec{x}, n) = \{F_i \in S(\vec{x}, n) \mid \sigma \geq \mid F_i - F(\vec{x}_c, n) \mid\} \tag{2.32}$$

• To combine those symmetrically distributed pixels (with respect to the central pixel), that are closest to the central pixel.

$$\begin{aligned} &\vec{N}_{\text{SYMNN}}(\vec{x}, n) = \{F(\vec{x}_c + \vec{k}, n) \in S(\vec{x}, n) \mid \\ &\mid F(\vec{x}_c + \vec{k}, n) - F(\vec{x}_c, n) \mid \leq \mid F(\vec{x}_c - \vec{k}, n) - F(\vec{x}_c, n) \mid\} \end{aligned} \tag{2.33}$$

• To only combine pixels if there is no edge between them (motion-adaptive, and edge-adaptive filtering, general term: edge-preserving smoothing [80]) or to prescribe a

Figure 2.12: Different types of noise. Top left corner shows the original Lena picture, top right the same picture corrupted with additive white Gaussian noise, bottom row left/right shows the effects of the clamp/shot noise, respectively.

specific combination of pixels depending on a local picture classification [81, 82]. For coding artifact reduction, the location of the pixel relative to the encoding block-grid may be used to affect the combination of pixels [83, 84].

From this general concept, many variants can be derived. First, the neighboring pixels that may be combined, the so-called filter support, can be 1, 2, or 3-dimensional and, either spatial, temporal, or spatio-temporal. Second, the combination can be a plain averaging, a weighted averaging, or a non-linear combination like rank-order filtering [85].

To further complicate the application domain, we recognize that some input pixels may have been filtered before (recursive filtering) [86–88], pixels in the support taken from neighboring pictures may be compensated for motion, i.e. taken from a shifted location depending on the local speed [89], while the noise smoothing may be in a transformed, e.g. wavelet, domain [90]. Finally, it is a common practice to globally adapt the parameters of the noise/artifact reduction filter to the estimated noise/artifact level that has to be, either estimated [91, 92], or communicated from the noise source (e.g. the

encoder).

To better understand the principles and requirements of the algorithms for noise reduction, in the following text we will report on a few published methods that utilize some of the previously briefly introduced features. Before we proceed, we visualize the challenges of the noise and coding artifact reduction in Figure 2.12.

Spatial noise reduction neglects temporal correlation between successive pictures in the video sequence and only combines pixels from the same picture. The advantage of this spatial filtering is its cost-effectiveness - there is no need for picture memories. Another advantage is the minimal delay caused by this function. For some equipment, the video signal must not be too much delayed (for example to prevent loss of the lip-synchronization) and in such cases spatial filtering might be preferred.

To exemplify spatial processing and some of the peculiarities mentioned above like adaptive weights and recursive processing, we refer to the filter realized in silicon (see Chapter 3 of [34]). This filter is actually related to the "sigma filter" [86] and the gradient inverse filter [93]. It is purely spatial (using just one picture reference) and utilizes a modest filter support consisting of two pixel-lines only. The pixels used in the recursive path are located one pixel-line above the line that is currently being interpolated. The recursiveness applied here does not hamper a parallel implementation since, theoretically, one complete pixel-line can be computed in parallel (there is no data dependency at pixel level). Eventual pipelined execution has to be performed within a line and not across lines. The weights of this filter are monotonously decreasing with increase of the absolute difference of the output and input luminance function. To make it simple for HW implementation, this monotonously decreasing function is piece-wise linear and has only three segments with the remark that in the two outer segments the coefficients are either one or zero.

Spatial-only filters neglect temporal information and introduce temporal artifacts like abrupt changes in the pixel values and oversmoothing [94]. In case of usage of (spatio-)temporal filters we profit from the temporal redundancy of the video signal while paying the price in the memory capacity and delay.

Drewery et al. [91] from BBC research did some initial research of the temporal filtering of noise. Application of temporal filtering requires at least one picture delay, which implies the usage of large memory elements capable of storing one or more picture references.

3D spatio-temporal can also profit from the recursive filtering [87, 88]. An alternative approach that ignores the correlation between pixels exist and it is based on ordered statistics filtering. We exemplify this approach by looking into the 1D temporal and 3D spatio-temporal solution. The simple and cheap temporal 1D solution has been proposed by Huang an Hsu [95]. For video sequences corrupted by Gaussian noise the average value approximates the maximum likelihood estimate. However, video signal is in general not stationary and thereby local distributions are not Gaussian. Therefore, a median filter should provide a better estimate of the original signal.

Alp et al. [85] use two-stage median based spatio-temporal 3D filter and references

Figure 2.13: Illustration of the coding artifacts. The left picture shows the blockiness artifacts and the right one shows the ringing artifact (picture showing ringing artifact is taken from http://www.gisdevelopment.net).

three pictures. They propose two algorithms, both being two stage median filter based, which is the only processing performed. Both proposed algorithms, called 3D planar filter (P3D) and 3D multilevel filter (ML3D) are of similar complexity of the first stage (three times 5-tap median versus two times 7-tap median) and have identical complexity of the second stage (3-tap median). The filter support used in the first stage of the P3D algorithm has the shape of the cross (for all three median operations). The filter support used in the first stage of the ML3D algorithm is different for the two performed operations. One uses the quincunx shaped filter support and the other one the shape of the cross.

2.3.2 Coding artifact reduction

The call for coding artifact reduction started with the introduction of digital transmission and storage. Theoretically, digital transmission can be lossless, but in practice, the available channel bandwidth makes lossy compression necessary, which results in visible artifacts [96] (see also Figure 2.13). Block-based Discrete Cosine Transformation (DCT) has been widely used for picture and video compression. To obtain a reasonable compression ratio, coarse quantization of DCT coefficients is necessary. As a result, the compressed pictures may exhibit coding artifacts and the perceptual picture quality is degraded.

The most annoying coding artifacts are blocking, ringing, blurring and mosquito noise [34]. All these artifacts are a consequence of omitting the high frequencies by truncating the DCT coefficients. Since the blocking artifact is the most visible one [97], we will here focus solely on this artifact. In the literature, very many approaches for blocking artifact reduction have been published [81, 83, 84, 98–109]. Among them, we can distinguish two main classes, the methods that operate in the *spatial* domain and the methods defined in the *frequency* domain. The methods in the spatial domain are more popular as they operate directly on the received spatial data (pixels) [81, 83, 84, 98, 99, 103–105, 110]. In other words, they do not require access to the DCT coefficients, which are usually not

available for post processing.

2.3.2.1 Artifact reduction in the spatial domain

Due to the horizontal and vertical "edges" or lines demarking the block grid, there are additional high frequencies in the spectrum of the decoded video signal. The logical remedy would be application of the low-pass filtering to suppress those frequencies [84, 110].

• Jarske et al. propose the Gaussian low-pass filter with a high-pass frequency emphasis [110]. A similar method, also based on Gaussian filtering, uses a 3*3 aperture applied only to the pixels that are located along the block boundaries [84]. Gaussian low-pass filtering computes a weighted average of the pixel within the support, in which the weights decrease with distance from the support center. The general assumption is that pictures typically vary slowly over space, so near pixels are likely to have similar values. It is further assumed that the noise values that corrupt these nearby pixels are mutually less correlated than the signal values, so noise is averaged away while signal is preserved. These assumptions however fail at edges, which are consequently blurred by low-pass filtering. The concluding general drawback of such algorithms is the loss of high frequencies (excessive blurring) [83]. To overcome this drawback, adaptive de-blocking algorithms have been proposed.

• In general, adaptive filtering [81, 98, 99] requires a classification step (usually based on the mean or variance statistics) followed by the linear or nonlinear filtering. For example, Ramamurthi et al. [83] use these two mentioned statistics to determine whether a block is a monotone or contains an edge. Edge blocks have higher variance. If it is concluded that a block is a monotone one, a 2D filtering is applied, otherwise 1D directional filtering is applied.

• Bilateral filtering [98] is a simple non-linear technique to remove the picture noise while preserving edges. The bilateral filter adjusts its coefficients to the geometric closeness *and* to the photometric similarity of the pixels. Bilateral filtering is a nonlinear operation because the weights depend on picture intensity. Regarding computational complexity, in addition to the filtering, analysis of the local content is required.

• Sobel filters are frequently used in the literature for edge detection [81, 100–102]. The example of usage of adaptive filtering in combination with Sobel kernels can be found in the algorithm proposed by Lee et al. [81]. The picture is classified into two areas, an edge and a monotone area using Sobel filters. After the edge information has been computed, two steps follow, 1D directional filter and adaptive 2D low-pass filter. This directional 1D filtering is performed on edges (to reduce the staircase artifacts) along eight directions, $0°$, $\pm 45°$, $\pm 90°$, $\pm 135°$ and $180°$. The second step, adaptive 2D low-pass filtering uses a 5*5 pixel support, excluding the corner pixels. The weights of this filter depend on whether pixels in the support are classified as edges or not. For simplicity reasons we omit here the explanation of weights computations, those details can be found in the mentioned publication, [81].

The algorithm is a 1D/2D spatial using just one picture reference. It actually consists

of four steps, two in the classification of edges and two in filtering. Two steps in classification assume computation of two Sobel-based weighted averaging. The Sobel filter support can be optimized to 3*2 pixels per edge and because the coefficients are fixed to 1, 2, 1, the multiplications can be avoided. The classification step also assumes computation of the mean and variance values. The filtering applied is pixel-based, conditional, and can be 1D (3-tap directional filtering) or 2D adaptive low-pass filtering using 21 taps in the worst case.

• Further quality improvement of the adaptive algorithms can be achieved if the algorithm in addition to the blockiness reduction may enhance details and preserve the existing edges. One such algorithm that applies blockiness reduction in combination with sharpness enhancement can be found in [99]. The authors use already mentioned content adaptive filtering with number of coefficient classes obtained from the off-line training process [44, 76]. Since the coefficients depend on the training material, if the reference pictures are satisfactory sharp, the algorithm can be optimized for both functions, blockiness reduction and sharpness enhancement. The authors conclude significant quality improvements with preservation of edges and other picture detail.

• Finally, another collection of methods for blockiness reduction pertains to the technique called the Projection Onto Convex Sets or POCS. POCS is effective in eliminating blocking artifacts but less practical for real time applications, since the iterative procedure adopted increases the computation complexity [111]. The quantization function applied during encoding is an injective function, i.e. has no inverse function. Youla et al. introduced the POCS to the field of picture processing [112]. Let A denote the set of all possible solutions that can result from the input encoded video signal. Let B denote the set of all solutions (sequences) without artifacts. Our solution lies in the set $A \cap B$. The constraint sets used in reducing the aforementioned set intersection is the problem of the POCS method. Some of the popular constraints like the *intensity* or *smoothness* constraint as well as other information about the POCS technique can be found in [103–106].

2.3.2.2 Artifact reduction in the frequency domain

The argument in favor of artifact reduction in the frequency domain might be that the spatial methods bring in excessive blurring [107]. As we have seen in the previous paragraphs, many spatial methods apply low-pass filtering and as such, result in unnecessary blurring of the picture. We have also presented a few advanced methods that successively deal with this in the spatial domain, and here we will register a few of them that operate in the frequency domain.

Relatively few approaches in the literature have tackled the problem of blockiness reduction in the frequency domain, including [102, 106–109]. The methods that operate in the frequency domain can either use coefficients available in the bitstream, or re-compute DCT coefficients in post processing. The problem of the DCT coefficients re-computation, apart from the extra cost, is the need to locate the original block grid. At the end of this subsection, we will report on a method, [106] that belongs to that class.

• A method [108] is recommended in the JPEG standard to suppress the blocking

discontinuities in smooth areas of the picture. It uses the dc values from current and neighboring blocks for interpolating the first few ac coefficients into the current block, by fitting the pixel values in a quadratic polynomial.

• Tan and Ghanbari propose the method to exploit the periodic nature of blocking artifacts [102]. The harmonics generated by the regular lattice pattern can be measured easily in the frequency domain, and give vital information of blockiness estimation. The amplitude of the harmonics is proportional to the degree of blockiness, while the phase of the harmonics can be used to verify that the harmonics are not due to contextual details in the picture. Examining both the amplitude and phase information of the harmonics leads to an accurate blockiness detector that needs no reference pictures.

• Minami and Zakhor in [107] present an approach based on the empirical observation that the quantization of the DCT coefficients of two neighboring blocks increases the expected value of the Mean Squared Difference of Slope (MSDS) between the neighboring boundary pixels. The MSDS is the criterion proposed by the authors. Therefore, among all the possible inverse quantized coefficients, the set that minimizes this MSDS is the most likely to decrease the blocking effect. The authors further propose a method to minimize the MSDS. This minimization can be formulated as a compute-intensive Quadratic Programming (QP) problem.

• The smoothing constraint, part of the POCS technique can also be applied in the frequency domain. Paek proposed to remove the high frequencies in the DCT domain by looking into the 8-point and 16-point DCTs [106]. Therefore, an 8-point DCTs are calculated on two horizontally neighboring blocks. In addition, one 16-point DCT is calculated on a joint block. If a high frequency is found in the result of the 16-point DCT and not in the 8-point DCTS, the authors conclude that this high frequency is a consequence of the block-boundary. They remove this frequency component by zeroing the appropriate coefficient(s) and performing the IDCT.

2.3.3 Sharpness enhancement

We can distinguish two classes of algorithms that try to enhance the subjective perception of sharpness. Algorithms from the first class apply non-linear methods to increase the steepness of the edges in the picture (edge enhancement methods). Algorithms from the second class increase high or middle frequency components by using small FIR filters (linear peaking methods). Peaking can also be dynamic (see for example [113] and silicon realization [114]). In the following paragraphs, we tackle these methods in more detail.

Shrinking or compressing the edges is a non-linear operation, which produces the impression of sharper picture. This technique is called the luminance/chrominance transient improvement (LTI/CTI). To achieve this, two mainstream methods are applied, the first one uses a tapped delay line and the second is basically a clipped peaking filter.

The first method computes the second derivative of the input video signal and the found amplitude indicates the position of the pixel which replaces the current pixel. This method requires the signals in continuous form or the sufficiently high sampling

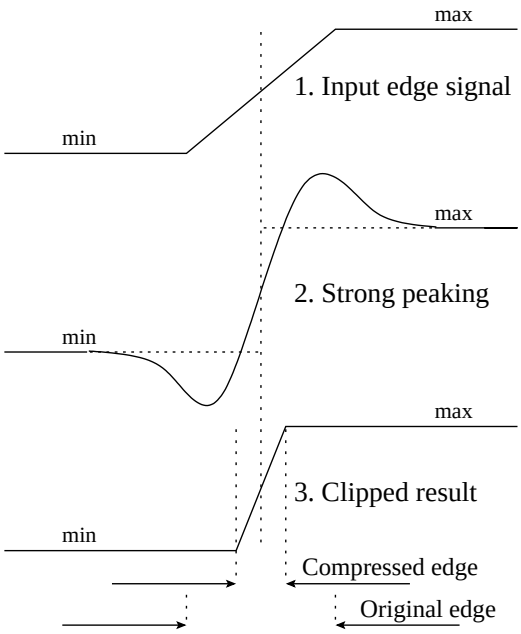


Figure 2.14: The LTI method consisting of successive peaking and clipping. The original edge is located, peaking is applied with higher overshoot followed by the clipping to the local minimal (min) and local maximal (max) values.

frequency (accuracy of determining the zero-crossings must be good enough).

There are a few drawbacks of this method. Some edges that should remain unchanged will become so sharp that they will look unnatural. If the transient has a long duration, it is unclear where is the center. Further, the first and the second derivation are very sensitive to the noise in the signal. This is especially visible in case of small second derivations. Transient improvement as a non-linear operation introduces new harmonics into the signal and thereby alias in pictures.

The second method is a clipped peaking filter. Figure 2.14 illustrates the concept of edge compression. To address the issue of sensitivity to the noise of the LTI/CTI, the peaked signal can be clipped. After the initial peaking has been performed, the results are clipped to the minimal and maximal value found in the local neighborhood [115]. In order to achieve a steeper edge, the applied peaking has higher overshoot than normally used. This can be done since the output of the peaking will be clipped afterwards. The algorithm is sketched in Figure 2.14. The authors recommend application of one more peaking after the clipping. This filter is a 21-tap 2D peaking filter using 5*5 filter support. A similar filter is applied for the first peaking step.

A method similar to the previously analyzed one has been proposed and realized in ICs [113, 114]. This method is used for both LTI and CTI. The authors also use a peaked version of the original signal as a reference. This signal is then clipped to the local min-

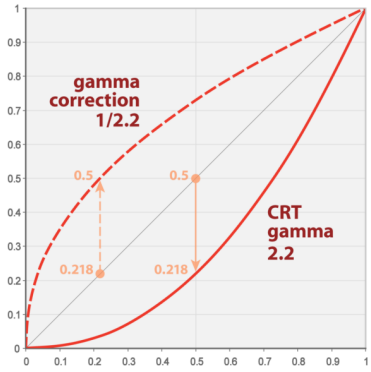


Figure 2.15: The process of gamma correction for CRT display. The graph shows a CRT's nonlinear transfer function, light intensity as a function of input voltage, and applied correction. Source: Wikipedia.

imum and maximum. Over- and undershoots are thereby avoided, allowing the method described to be used not only as a luminance transition improvement, but also as a chrominance transition improvement. The authors reason that the over- and under-shoots in the chrominance part are quite irritating for the HVS (human visual system). Since the over- and under-shoots are avoided by the clipping, they apply this method directly for CTI as well. The detection of the local maximum and minimum uses a small horizontal support of 4 pixels. The local extremum detection is performed using multiplexing logic. The second step (the clipping) is realized as a simple 3-tap median with the following inputs: local minimum, current pixel and the local maximum.

2.3.4 Contrast enhancement

Contrast enhancement is a non-linear technique in which gray-level of the picture is rescaled. There are three major methods for contrast enhancement, gamma correction, automatic black and soft-clipping. Histogram modification includes all of these.

2.3.4.1 Gamma correction

Displays in general convert a video signal into the light in a non-linear way. Looking at the cathode ray tube (CRT) displays for example, we observe that the electron gun is a non-linear device where the intensity of the light I is exponentially related to the input voltage V_S, as described with the following equation.

$$I = const V_S^{\gamma} \tag{2.34}$$

where γ (gamma) is a measure of the nonlinearity of a display device. Each display has its own value of gamma, for a CRT display, it is about 2.2. The effect of this non-linearity for CRT is illustrated in Figure 2.15. In order to get linear relation between the applied

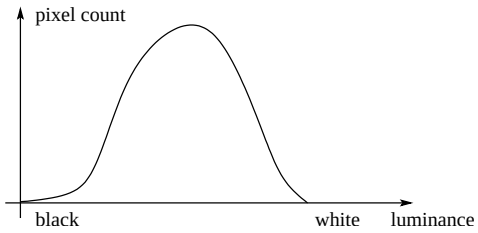


Figure 2.16: Illustration of the histogram. Even though the pixel count is plotted here as a continuous curve, it is often realized as a discrete function computed over several intervals (assuming implementation in the digital domain).

stimulus and perceived light intensity, we must apply inverse function to the video signal before sending it to the display.

The optimal gamma correction setting depends on the taste of the user, ambient light and other conditions. Thereby, this setting must be left under user control or adjusted automatically. Gamma control is one of the basic settings of a display and is available in various ICs [116–118].

2.3.4.2 Automatic black control

To enhance the contrast, there is a need to emphasize the black parts of the screen (since the human eye is more sensitive to changes in dark areas than in light areas). There are three major methods of black control, auto pedestal, black restore and black stretch (ranked from the simpler to more complex).

• Auto pedestal measures the darkest pixel in the picture and then subtracts this value from the entire picture. As a result, the smallest luminance values become truly black. The obvious disadvantage of this simple method is global brightness reduction.

• Black restore slightly improves the performance of the auto pedestal since white remains white. This is achieved with a transfer curve with a gain greater than unity.

• Black stretch is a more advanced method with a three-segment-linear characteristics. Light-gray pixels are unchanged by the black stretch algorithm due to the non-linearity of the transfer curve. Skin tone also look more natural as the parts of the picture having skin tone colors are not darkened.

2.3.4.3 Histogram modification

Histogram modification practically contains all the previously mentioned methods for video enhancement. As its name says, the essence of this technique is to calculate the histogram of the picture, i.e. the distribution of luminance values in the picture. Based on this (statistical) data, the values of some pixels are modified such that e.g. contrast is improved. As in the majority of similar techniques, the risk is that the artist did not actually want any modification or ”improvement” of the picture. Another negative aspect

is that if the noise was present in the picture, it will probably be emphasized. Figure 2.16 illustrates the histogram.

More information about physical implementation of the histogram modification can be found in the IC datasheets, e.g. [119]. This particular IC keeps track of 32 different gray levels. The major functions that it performs are the black stretch and the white stretch.

2.3.5 Colour reproduction enhancement

With the television standard evolution from NTSC to PAL and SECAM and with newer display technologies, the primary colors have changed significantly. In spite of improving the power efficiency, the new colors are less saturated, which implies that the color gamut of the new TV set is smaller than the old TV set. There are four major methods for color enhancement, linear color space conversion, green enhancement, skin-tone correction and white adjustment.

• The solution to this problem starts from converting one set of R, G and B components to another, R', G' and B'. To realize that, one can use a matrix multiplication such as:

$$\begin{bmatrix} R' \\ G' \\ B' \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} . \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{2.35}$$

However, matrix multiplication is a fully linear operation and we have seen in the previous paragraphs that video signal suffers severe non-linearity on its way through the chain of video applications (for example gamma correction). The best achievable optimization with matrix multiplication is to aim at three corrected color points. The usual choice falls on the green, skin-tone and white colors.

• In principle, the technique for green enhancement is quite simple. The color detector senses the green-like areas in the picture and for those areas, the saturation is increased. Thereby, more vivid colors are obtained.

• The human eye is especially sensitive for the skin tone and thereby, dynamic skin tone correction can be applied. As a negative consequence, not only human skin is "corrected" but everything that looks like the human skin is modified.

• Depending on the region of the World and the personal consumer taste, the different color temperature of the white color might be preferred. As a result, not only the white point are modified, but all the other weakly saturated colors as well. To prevent the negative influence of this to the skin tone, a detector is used to find the areas with little saturation and allow the mentioned change to happen only in those areas. Skin tone is saturated enough such that it does not pass through the detector's filter and remains unchanged.

2.4 Motion estimation

Motion estimation is one of the key enabling algorithms for modern video processing. Initially, the implementation demands were too high, partly because of immaturity of architectures that lacked performance and partly because the motion estimation algorithms were indeed unnecessary computational intense. With the waves of digital revolution, processing architectures significantly gained processing power and flexibility, while at the same time a lot of research went into simplifying motion estimation algorithms. The result is that motion estimation is now embedded in all major video compression standards and in various high end video post processing integrated circuits [89, 120–126].

The evolution of motion estimation (ME) algorithms starts with the pixel-based algorithms [127], continues via block-based [23] to object-based ME algorithms [128]. The high compute demands of pixel-based algorithms limited their usage to academia only. Block-based estimators are performing on groups of pixels (blocks), meaning that a single, best matching motion vector is associated to a *block* of pixels. They are much more noise-robust compared to the pixel-based ones and, due to their block property, much more attractive for silicon implementation. The third class of motion estimation that goes one step further, is called object-based motion estimation aiming to a single motion model description for each object. By far, the most popular ones are the block-based motion estimation algorithms since they offer the best ratio between the performance (picture quality) and compute intensity. Apart from a few words about object-based estimators, we shall devote this section completely to the block-based ME.

A number of block-based motion estimation algorithms exist. Block-based ME algorithms or block-matchers divide the picture into blocks of pixels $B(\vec{X})$ with center $\vec{X}$ and assign to all pixels of every block at picture number n a displacement vector, $\vec{D}(\vec{X}, n)$, selected from a candidate set, $CS^{\max}$, that limits the possible output vectors to a search space or search area, SA.

2.4.1 The full search algorithm

Probably, the best known motion estimation algorithm is the full search block-matcher (FSBM) where within a certain search area and accuracy, all possible motion vectors are evaluated. Formally, the full-pel accurate candidate set of the FSBM is defined with the following equation:

$$CS^{\max} = \left\{ \vec{C} | -N \leq C_x \leq +N, -M \leq C_y \leq +M, C_x, C_y \in \mathbb{Z} \right\}, \tag{2.36}$$

where *N* and *M* represent the horizontal and vertical dimensions of the search area, respectively.

The advantage of the FSBM is a good access regularity, which offers a number of possibilities for parallel implementation. The FSBM has the following three major drawbacks:

- A high computational complexity,

- Its motion vectors have a poor relation with the *true motion* of the objects, and,
- It cannot deal with the occluded areas.

In the following three subsections, we register the ways to remedy these three drawbacks.

2.4.2 Complexity reduction

Assuming that FSBM operates on a modest SA of 72*56 pixels, the number of evaluations results in more than 3000 motion vector candidate evaluations per block. Assuming a sum-of-absolute-differences (SAD) as an evaluation criterion (will be explained shortly after) and the dimensions of the SAD window of 8*8 pixels, almost 600 KOPs are needed to evaluate all the candidates of a block of 8*8 pixels[1]. Such a high operation count disables an efficient real-time implementation. To reduce the complexity, we have four options available: pixel sub-sampling, block sub-sampling, complexity reduction of the cost function and efficient search strategies.

2.4.2.1 Pixel and block sub-sampling

Pixel sub-sampling reduces the number of pixels participating in the SAD calculation. Usually, the sub-sampling can go up to a factor of four [120, 129]. Block sub-sampling means that the SAD computations are not performed for some blocks [130]. Instead, motion vectors are computed based on their neighbors in the motion vector field. Pixel and block sub-sampling reduce complexity by few times, which is still not enough for efficient real-time implementation.

2.4.2.2 Cost function reduction

Motion vectors are evaluated according to a certain cost function or criterion. Three major criteria have been proposed: The sum-of-absolute-differences (SAD), the mean squared error (MSE) and the normalized cross correlation function (NCCF) criterion, which is only meaningful if the match error is computed in the Fourier domain. By far the most widely used criterion is the SAD criterion, which offers a good price/performance ratio [34]. The absolute difference is found between the respective blocks of pixels originating from two picture references distanced by p pictures and displaced by the motion vector candidate $\vec{C}$ being evaluated. The motion vector candidate yielding the minimal SAD value is selected as the winner of the process. Formally, the match error for candidate $\vec{C}$ according to the SAD criterion is defined with the following equation:

$$SAD(\vec{C}, \vec{X}, n) = \sum_{\vec{x} \in B(\vec{X})} |F(\vec{x}, n) - F(\vec{x} - \vec{C}, n - p)| \tag{2.37}$$

[1] As 1OP, we consider addition, subtraction and absolute difference. Each of these three contribute by roughly 200KOPs. For simplicity reasons we have omitted the overheads such as the comparisons of the SADs and motion vector generation.

2.4.2.3 Efficient search strategies

Iterative algorithms such as the three-step search [131], logarithmic search [132] or one-at-a-time-search (OTS) [133] offer more significant complexity reduction. The first step of these algorithms is to obtain the best matching motion vector on the coarse pixel grid. For each next step, a new search is performed on a finer grid, centralized around the pixel where the best motion vector from the previous step points to.

The three mentioned algorithms on the average evaluate up to a few dozens of motion vectors, which is roughly two orders of magnitude lower than the FSBM. This does enable a real-time implementation. However, from the quality point of view, they do not find the best matching motion vector in the SA, they just find the local minimum that might not be the global one. Further, the problem of motion portrayal is still not solved. A few methods discussed in the further text have been proposed for this purpose.

2.4.3 True motion

As a result of the FSBM, the vector that yields a minimal SAD value is selected. The techniques that we registered in the previous subsection reduce the number of computation but the problem of motion portrayal is still not solved. All these methods (including the FSBM) aim only for the best match in terms of the applied cost function (error criterion) but do not take into account the real motion velocities of the objects in the screen. In this subsection, we report on some of the methods that take the motion portrayal into account.

2.4.3.1 Motion vector field post processing

Post-processing should preferably not create any new motion vector (the vector that did not result in the estimation process). Median filtering is frequently used (see also Subsection 3.3 of [134]). Reuter [135] proposed to use a 2D median filter with a support of 5 horizontally and 3 vertically, centered around the motion vector being filtered. This technique uses relatively large number of taps that might have impacts to the HW implementation. The advantage of this method is that it is performed on the motion vector field grid, rather than on the pixel grid, which significantly reduces the number of elements on which the computation has to be performed.

Another median-based method that improves the quality of the motion vector field is block erosion. Block erosion reduces the granularity of the motion vectors associated to blocks. For example, if the generated motion vector field is of 8*8 granularity, eroding it might lead to 4*4, 2*2 or even 1*1 blocks, each having its own motion vector associated to. There are few methods used here, but the most popular one (since it does not introduce new values) is the one based on the median filtering of the current vector and its spatial neighbors. Block erosion requires small filter support containing the nearest motion vector neighbors. From the processing point of view, it requires 3-tap median filtering, with the only peculiarity that it operates on *two independent* motion vector components. Block

erosion makes the task of motion compensation more compute intensive since it reduces the granularity of processing.

In some occasions, the motion in an entire picture can be described with a very limited parametric model. For example, zooming with the camera will generate motion vectors that linearly change with the spatial position. Panning, tilting or traveling with a camera, on the other hand, will generate a uniform motion vector field for the entire picture. An example of the parametric model can be found in [136]. Based on the derived parametric model, an additional motion vector candidate is added to the candidate set. Global parametric motion model assumes computation on the generated motion vector field. This processing is performed once per picture reference (after the motion vector field has already been generated). It is thereby not compute intensive although it requires the memory capacity capable for storing of one motion vector field. It does not require eroded motion vector field, it operates on the original motion vector grid.

2.4.3.2 Hierarchical algorithms

Hierarchical motion estimators construct the motion vector field in a few iterations, starting from the strongly pre-filtered and sub-sampled picture and ending with the highest original resolution grid [137]. The result from the lower resolution is used as the initialization for a more accurate estimate at the next sub-band, which contains higher frequencies. The consistent motion vector field is realized through this process of initialization of motion estimators with a global estimate.

A method related to the hierarchical motion estimation is phase plane correlation. Phase plane correlation is initially introduced in the field of astronomy [138], and later was successfully applied in studio picture-rate conversion [139] and consists of two steps. The first step operates in the frequency domain on fairly large blocks (e.g. 64*64 pixels) and produces a limited number of motion vector candidates (usually less than 10), which is used by the second step. The first step starts by computing the 2D Discrete Fourier Transform (DFT) on the blocks originating from the current and the previous picture reference. Afterwards, the Cross-Power Spectrum (CSP) is computed. The second step operates in the spatial domain on much smaller block grid (ranging from 1*1 to 8*8 pixels) and uses the candidate set produced by the first step.

To analyze the computational complexity we start with remark that the algorithm uses two picture references and the number of candidates and processing steps is known in advance. The peculiarity of this algorithm is that the first step is performed in the frequency domain, which requires computation of the 2D DFT and IDFT, multiplication in the frequency domain and normalization. In the spatial domain, a motion estimation algorithm is performed. The number of evaluations per block is dictated by the first step. This offers a possibility for a tradeoff since the number of evaluations can be specified apriori, thereby limiting the compute requirements of the second step.

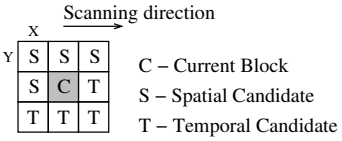


Figure 2.17: The background of the 3DRS block matcher: The blocks marked with *S* provide the spatial prediction candidates, those with *T* provide the temporal ones, while *C* identifies the current block.

2.4.3.3 Maximum-a-posteriori methods

In statistics, the method of maximum a posteriori (MAP, or posterior mode) estimation can be used to obtain a point estimate of an unobserved quantity on the basis of empirical data. It requires at least two probability distribution models, the conditional probability of the observed picture intensity given the motion field and the a-priori probability of the motion vectors. In the context of motion estimation, the MAP method aims to find the motion vector field which maximizes the posterior distribution. More details about the maximum-a-posteriori methods can be found in [140]. Example of the MAP estimation is the optical flow, which is based on the assumption that the picture intensity remains constant along the motion trajectory. Even the 3DRS (explained in the further text) can be viewed as an MAP method [140].

2.4.3.4 The 3DRS motion estimation algorithm

Another way to arrive at economical ME algorithm, which features the true motion is to use the recursive approach. Such a solution, called the 3-dimensional recursive search (3DRS) has been proposed [23, 129]. Economical attractiveness comes from the low candidate count and the true motion comes from the recursive candidate selection that originates from the spatio-temporal 3D neighborhood. The 3DRS reduces the candidate set of an FSBM based on the following two assumptions:

1. Objects within a picture are larger than blocks, and
2. Objects have inertia.

The implication of the first assumption is that evaluation of all possible vectors within the search area denoted with $CS^{\max}$ is not necessary, as a candidate set that contains result vectors taken from the neighbors (marked with *S* and *T* in figure 2.17) should be sufficient:

$$CS(\vec{X}, n) = \left\{ \vec{C} \in CS^{\max} | \vec{C} = \vec{D}(\vec{X} + \begin{bmatrix} iX \\ jY \end{bmatrix}, n) \right\}, i, j = -1, 0, 1 \qquad (2.38)$$

where, *X/Y* are the block width/height.

The initialization problem (all the vectors are zero) and convergence problem (also after the scene change) are tackled by adding a (pseudo) random update vector to one of the spatial neighbors and using it as an additional candidate. This update vector is chosen

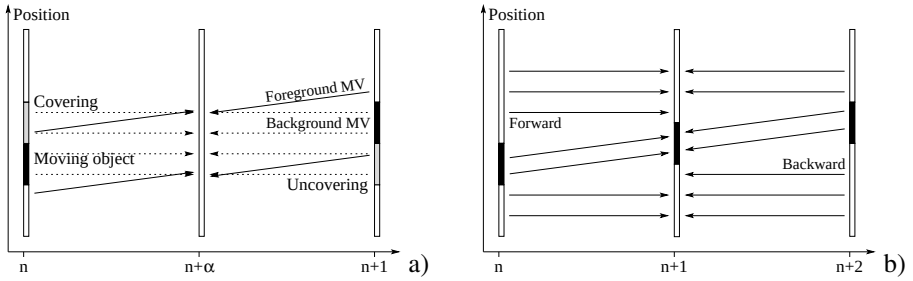


Figure 2.18: Picture a) illustrates the limitation of the two picture reference up-conversion. The foreground vector often provides the best match in the occluded background (gray-shaded). Picture b) shows that three picture-reference up-conversion enables the selection of the proper motion vector even in the occluded areas.

cyclically from a predefined update set, such as:

$$US_i = \left\{ \begin{array}{cccc} \vec{y}_u, & -\vec{y}_u, & \vec{x}_u, & -\vec{x}_u, \\ 2\vec{y}_u, & -2\vec{y}_u, & 3\vec{x}_u, & -3\vec{x}_u \end{array} \right\}, \text{where } \vec{x}_u = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \text{and } \vec{y}_u = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad (2.39)$$

Result vectors can be sub-pixel accurate if the update set also contains fractional updates. Different update strategies are also possible (see for example [120]).

The 3DRS block matcher uses two picture references. In the case of quarter-pel accurate vectors, the bilinear interpolation is applied to interpolate the missing pixels. The bilinear interpolation uses four nearest neighbors of the pixel being interpolated. The usage of pixel sub-sampling within the block is also optional to reduce the operation count. To further reduce the operation count, the author proposed a technique to reduce the already low number of motion vector candidates that 3DRS uses [141].

The recursive nature of the algorithm makes difficult parallel evaluation of two blocks. The possibility of pipelined execution is still left open. The horizontally neighboring spatial motion vector candidate is evaluated as part of the candidate set of the current block. Thereby, the evaluation of *this particular candidate* can start only *after* the evaluation of the previous block has been completed. However, the evaluation of all the other motion vector candidates can start even before the evaluation of the previous block is completed. This conclusion is valid for one complete block-line.

2.4.4 Occlusion-aware estimation

All the previously mentioned methods assume that the motion estimation has been performed on two picture references. The biggest problem of these methods is to estimate the motion between two pictures for the occluded parts, i.e. those parts of the background that are being covered or uncovered. Figure 2.18a illustrates that problem. This problem is present since only two pictures are used in motion estimation and in occlusion areas, no proper match can be found If the third, i.e. next, picture is used in addition to the current and previous, then the correct motion vectors can be found even in the occluded areas. The motion estimation that uses three picture references is illustrated in Figure 2.18b.

The first motion vector field is computed backward using the current (n) and the previous ($n-1$) pictures as references and the second one is computed forward using the current (n) and the next ($n+1$) pictures as references. Both motion vector fields are valid at the current picture (n). This is actually the problem since we are interpolating a picture located *in between* two pictures, e.g. ($n+\alpha$) and thereby we need a motion vector field valid at time instance ($n+\alpha$). Vector field re-timing [22] addresses this issue.

Even after the re-timing is performed, in order to properly perform the up-conversion, the occlusion type has to be determined. Two types of occlusions exist, the covering and uncovering. In the case of covering, we use previous picture for interpolation while in the case of uncovering, the next one. More data on the topic of occlusion-aware motion estimation involving advanced motion vector field processing can be found in [22].

Looking at the HW aspects of this method, we first note that this is the only motion estimation algorithm that uses three picture references. Another peculiarity is that the motion estimation is performed *twice* per original picture reference (forward and backward). Apart from having three picture references and performing the motion estimation twice, the computations on the motion vector field are quite intense. Analyzing the complete picture-rate up-conversion algorithm, in [67] it was concluded that the motion vector field post processing takes 88% of the total operation count.

2.4.5 Object-based motion estimation

The usage of block-based motion estimation results in the blocking artifacts since the boundaries of the real objects in the scene in general do not correspond to the boundaries of the blocks used in the estimation. Object-based motion estimation (OBME) was introduced as a method to overcome this difficulty. In spite of being a promising technique, OBME until now did not gain any significant popularity, possibly due to already satisfactory results of block-based estimators. Since there are only a few references of the OBME realizations [142, 143], we limit our scope here to only briefly mention the very basics of the OBME, without going into any detailed analysis. In general, we distinguish between the bottom-up and top-down OBME methods.

Bottom-up methods (see for example [136]) start from the picture segmentation based on the previously calculated motion vector field. The regions that feature the same or similar motion are grouped together. These large regions are then treated independently. Top-down methods are based on the changed/unchanged rule (see for example [142]). The two consecutive pictures are compared: When two corresponding motion compensated (blocks of) pixels in these two pictures are significantly different, these pixels are marked as changed. The new motion model is calculated for these portions of the picture and the process is repeated until there are no positions marked as changed.

Regarding the complexity of the OBME, we could refer to the relative complexity comparison of the OBME and the 3DRS [143]. The authors conclude that their OBME implementation requires 10% more operations while maintaining the same level of quality. The additional benefit that the OBME brings is the segmentation map that could potentially be used by some other application.

2.5 Conclusions

Decades of video post processing research resulted in an impressive stack of different algorithms. Each of the functions in the domain can be implemented using different algorithms. The diversity of these algorithms is visible in many aspects. Algorithms can be based on spatial, temporal and spatio-temporal processing, which can be recursive, content-adaptive and linear as well as non-linear.

Our goal is to identify algorithmic approaches that are generic enough to cover a wide range of applications with high quality and acceptable implementation cost. The number of different approaches should be limited to simplify the task of implementation. We have selected two distinct approaches, the first enabling block-based motion estimation/compensation and the other pixel-based content-adaptive filtering.

Having in mind everything said in this chapter, block-based motion estimation such as the 3DRS was an obvious choice. Indeed, it is cost-effective and enables a substantially better quality than other approaches throughout the application domain. Per application, we find many algorithms that use block-based motion estimation and compensation. De-interlacing [21, 35–38], picture-rate up-conversion [22, 39–43] and noise reduction [87, 89] are good examples of applications that profit from motion-compensated processing. Our application domain requires true-motion, so support for the 3DRS motion estimator was a logical choice. Finally, motion estimation and compensation are essential ingredients of all major video compression standards. Even though video compression was not within the immediate scope of this thesis, supporting motion estimation and compensation certainly extends the coverage of this work beyond the target application domain.

Content-adaptive filtering has been selected as the second distinct approach for two reasons. The first reason is that for a number of applications, it enables high quality of output pictures. The recently proposed class-reduction methods [77, 78] make it suitable for cost-effective implementation. Examples include de-interlacing, up-scaling [44, 45], coding artifact removal and sharpness enhancement [82, 99]. The second reason is that content-adaptive filtering is generic and also covers algorithms that obtain good quality even with fixed coefficients. This is a very large group of applications based on FIR filtering or matrix multiplication. Occasionally, the coefficients depend on a single or a few conditions and in such cases, we can think of having just a few coefficients classes. Most of the algorithms that require one, or a few, coefficient classes are found within the video enhancement pillar, e.g. [100, 101, 113, 115, 116, 118, 119].

3

Application Domain Processing

IN the previous chapter, a wide variety of applications was discussed which all belong to the target domain. Our goal is to define an architecture that combines efficiency with programmability in order to deal with the different applications. A three-step approach is adopted. The first step is algorithm development. The functional requirements at the application level (the *what* description) are translated into a high-level language such as ”C” representing an implementation at the algorithmic level (the *how* description). An application can be implemented by many different algorithms possibly offering a variety of quality levels. In a second step, the SIMD form of parallelism is introduced for performance reasons and the ”C”-like description is vectorized. A common vector instruction set is defined. In a third step, a datapath is defined and the requirements for the memory subsystem are derived.

This chapter describes the gradual convergence to a single architecture and consists of three sections. The first section discusses the algorithm step. The algorithms from the application domain belong to one of the two algorithmic classes, the pixel-based or the block-based class. As concluded in the previous chapter, there are algorithms from both classes that offer acceptable picture quality, i.e. the content-adaptive algorithms as representatives of the first class, and, motion-compensated algorithms representing the second. The second section shows how the algorithms of both classes can be vectorized resulting in a common vector instruction set. The principle is illustrated on most important kernels taken from class representatives. Lastly, the third section concludes the chapter and presents the directions for future work.

3.1 The two algorithmic classes

The algorithms from the application domain can be classified in two algorithmic groups or classes, the pixel-based or the block-based class. An application might have a tendency or preference towards one of the two mentioned algorithmic classes. Another possibility is that an application can be implemented using both algorithmic classes. All the possibilities are illustrated in Figure 3.1. Typical examples of block-based applications are picture-rate up-conversion and de-interlacing. These high-quality applications use block-based motion estimation and compensation as the basic algorithms. Typical examples of pixel-based applications are de-interlacing, spatial up-scaling, sharpness enhancement, histogram and gamma correction. To stay as generic as possible, we allow that the coefficients depend on the local content, for example, as described in the T. Kondo method [44, 76].

Our goal is to cover the algorithms from both classes. Not all the algorithms are directly supported, for example, recursive pixel-based filtering. According to our approach, a number of input and output pixels are processed in parallel. This cannot be directly done in case of recursive pixel-based filtering, because of the data dependencies between the output pixels. One of the methods for avoiding the data dependencies could be to compute partial products separately in parallel. Only the final product and sum would be computed in sequence. We did not explore these possibilities since the supported algorithms from both classes ensure sufficient quality of output pictures. The above does not affect the recursive motion estimation algorithms such as the 3DRS since the data dependencies exist at the block level only (and not at the pixel level). This enables parallel computation of pixels pertaining to a block.

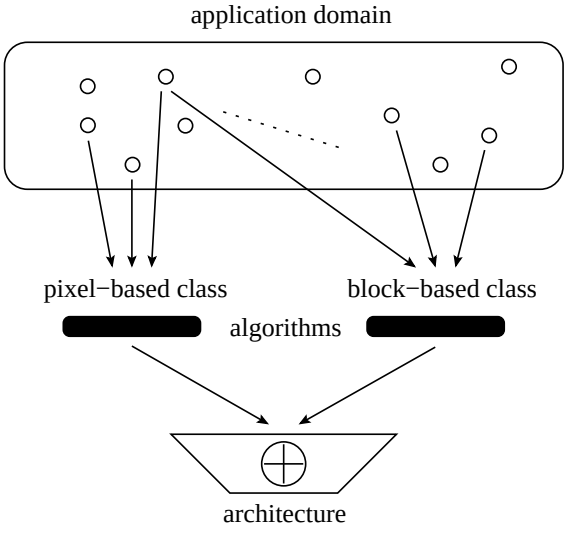


Figure 3.1: Different scenarios during mapping applications to two algorithmic classes. An application can be mapped to one of the two classes or to both.

De-interlacing is an example that can be mapped onto both classes. The block-based variant is based on the Generalized Sampling Theorem and uses block-based motion estimation/compensation [21]. Pixel-based de-interlacing is edge-dependent [54].

First we analyze the characteristics of both classes followed by a comparison. The characteristics of each class will be analyzed using a representative algorithm. The inevitable ingredient of block-based algorithms is motion estimation algorithm, such as the 3DRS. This is why motion estimation is selected to be the representative of this class. For pixel-based algorithms, filtering is performed in most of the cases. Since it is compute-intensive, in many cases defining the performance, we select it as a representative of the pixel-based class. As mentioned before, the coefficients used in filtering may depend on the local pixel content.

3.1.1 Algorithm characterization

In order to characterize the two algorithmic classes, they will be analyzed from two angles, access pattern (position of the pixels that are used in the processing) and the type of the processing involved. The next two subsections address pixel-based and block-based algorithms, respectively. Finally, the differences and commonalities between these two algorithmic classes will be determined.

3.1.2 Pixel-based algorithms

Filtering is part of most of the pixel-based algorithms. Depending on the number of used filter taps, it can be compute intensive requiring a large number of multiplications and additions per pixel that influences the cost. A typical scenario is that a central pixel is interpolated using its neighboring pixels. As mentioned before, the coefficients used in interpolation may depend on the local picture content. This is taken into account in all the illustrations and pseudo-code snapshots related to the pixel-based class. The characteristics of pixel-based algorithms, are summarized by the following pseudo-code.

Pixel-based algorithms: (e.g. filtering)

```
for (each interpolated pixel I ∈ picture) {
   I = 0 ;
   for (each pixel C_i ∈ support(I)) {
      I = I + c_i(I)C_i ;
   }
   normalize(I) ;
}
```

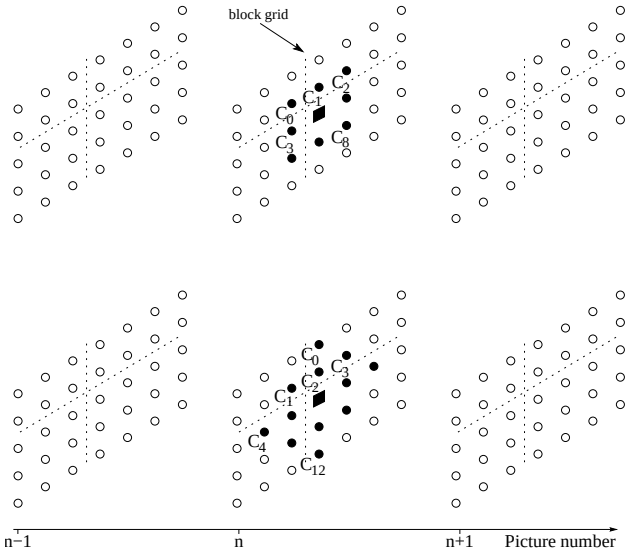


Figure 3.2: Some of the supports used in pixel-based processing. Most of the pixels do not pertain to the block-grid. The filter supports illustrated here are used in content-adaptive spatial up-scaling and sharpness enhancement. Black pixels participate in the interpolation and the pixel being interpolated is marked with a black square.

To characterize pixel-based algorithms, we first look into the access patterns. The access patterns depend on the distribution of pixels in the filter support. Various supports are used, for example, a square, a rectangle (e.g. 3*2, 3*3 pixels) or a diamond (e.g. involving 13 pixels) [23, 45, 52, 82]. Figure 3.2 provides an illustration. An imaginary block-grid is overlaid on this figure. In case a group of pixels is stored at one memory location, the memory address of that pixel group coincides with the block grid. In the above figure, we have assumed that the interpolated pixel is located at the block grid, which in general is not the case. As we can see from this figure, even in this case, the locations of the majority of the pixels from the filter support do not coincide with the block grid. This poses a challenge for retrieval of such pixels from the memory subsystem. We shall discuss this in greater detail in Section 3.2 that introduces vectorization.

To conclude, the processing consists of n-tap filtering that is performed per output pixel. The number of taps, n, is equal to the number of pixels in the interpolation support. As mentioned before, the coefficients used in the interpolation may depend on the local picture content. Lastly, all the pixels within the support are spatially localized. This locality of reference can thereby be used.

3.1.3 Block-based algorithms

The characteristics of block-based processing is that a picture is split into blocks of pixels and the processing is performed in a block-by-block fashion. Motion estimation and compensation are the only block-based basic algorithms from the domain that ensure high quality. Block-based motion-compensated (MC) algorithms compare blocks of pixels from at least two picture references. Here, we call them the previous and the current picture reference. The position of the currently processed block (located in the current picture reference, time instance n) pertains to the block-grid. The position of the block in the previous picture reference, time instance n, is determined by the motion vector, which can point at any pixel within the search range (search area). Since the motion vector pertains only to the pixel grid, the position of the referenced block does not pertain to the block-grid in general. Additional aspect of the pixel fetching applied here is that the locations on the pixel-grid are known only at run-time, since a motion vector candidate is a run-time variable. Lastly, the motion vector candidates are limited within the boundaries of the Search Area (SA), as mentioned in Section 2.4.

The processing is typically block-comparison. In the common case of sub-pel accurate motion vectors, pixel interpolation is used as well. For each motion vector candidate, a difference between a block from the current and previous picture reference is computed using an error criterion. The most commonly used error criterion is SAD (sum-of-absolute-differences), illustrated in Figure 3.3. SAD requires pixels from at least two picture references, and possibly more [22, 67]. If the pixel grid is interlaced, the com-

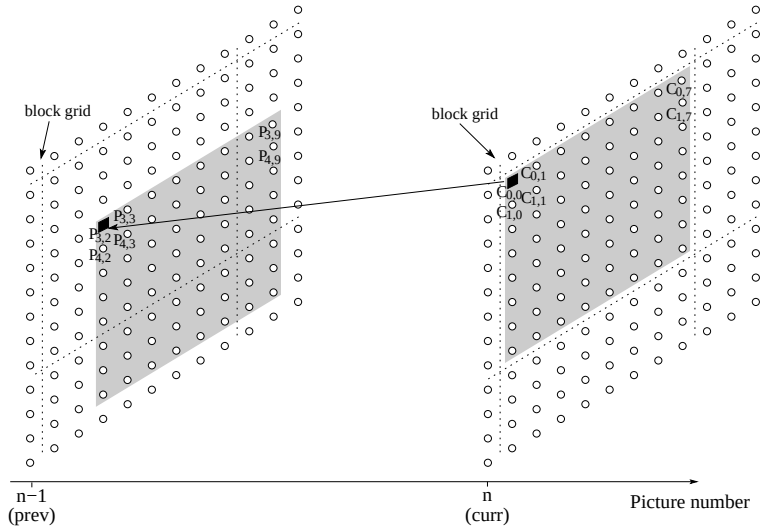


Figure 3.3: The access pattern used in motion estimation. The pixels that participate in SAD calculation are shaded. The block at time instance n (current picture reference) pertains to the block grid while the block at time instance $n-1$ (previous picture reference) does not. The index $(0,0)$ of the top-left pixel at the previous picture reference is chosen for clarity.

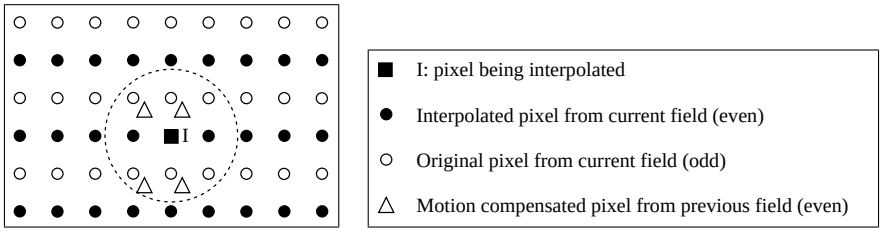


Figure 3.4: The circular support of the interpolation on the interlaced grid that involves pixels from two field references (spatio-temporal filtering). This is used in high-quality motion-compensated Generalized sampling theorem based de-interlacing.

putation of SAD might become more complex than bi-linear interpolation. An example is high-quality 2D Generalized Sampling Theorem (2D GST) based de-interlacing. This block-based de-interlacing algorithm uses motion estimation and compensation.

Much like other motion-compensated algorithms, in the process of pixel interpolation, this algorithm involves pixels from two fields, the original pixels from the current field and the motion-compensated pixels from the previous field [21, 60]. Taking into account pixels from both references, the filter support has a circular shape and consists of ten pixels. This is shown in Figure 3.4. It is possible to split this circular two-reference-field support into two single-reference-field supports. The first support consists of 3*2 pixels original pixels from the current field. In Figure 3.4, these pixels are marked as white circles located within the dotted circle. The second support consists of 2*2 motion compensated pixels originating from the previous field. These pixels are marked as white triangles. Thereby this circular support is equivalent to the union of the two rectangular-shaped supports originating from two reference pictures. Without loosing generality, in the further text, we continue with the motion estimation that uses rectangular support for interpolation. Typical example is bi-linear interpolation that uses rectangular 2*2 pixel support. The characteristics of this algorithmic class are summarized in the following pseudo-code.

Block-based algorithms: (e.g. motion estimation)

```
for (each block B(X⃗) ∈ picture) {
   for (each candidate MVC⃗) {
      CB = fetch_aligned (B(X⃗), n) ; // current block
      PB = fetch_unaligned (B(X⃗ + MVC⃗), n − 1) ; // previous block
      SAD = block_compare (CB, PB) ;
      compute_min (SAD) ;
   }
}
```

3.1.4 Comparison

The main difference between the algorithms from these two classes is that pixel-based algorithms treat each pixel differently while in the case of the block-based ones, a group of pixels is treated in the same way. Other differences, valid in a typical case, include the following. Block-based algorithms impose data dependencies between successive blocks while pixel-based algorithms usually do not (at the pixel level). Block-based algorithms basically compare rectangular blocks of data while pixel-based algorithms involve filter supports of different shapes.

The text provided in the above subsections showed that the two algorithmic classes have differences in the access pattern and processing. Because of those differences, the transformations to introduce vector type parallelism are different. But, both classes have also commonalities. Both of them require access to the pixels that do not pertain to the block grid. Additionally, accessed pixels are always localized that opens the possibility for the usage of the locality of reference.

3.2 Vectorization of the algorithms

In this thesis, we use vector operations to process a number of pixels in parallel. For the sake of demonstration, the used instruction set is generic. By using generic instruction set, the software can be written regardless of any algorithmic peculiarity such as the number of pixels in the support, the shape of the support, etc. A generic vector instruction set has already been proposed and used [27–31] and in this section we show that this concept can be applied to our two algorithmic classes. To further increase the performance, custom-ops may be added to the instruction set. Within this application domain, this has already been proposed in [32, 33]. In general, vector instruction set offers a number of advantages. Let us mention a few of them.

The architecture is parametric and can be tuned for different performance nodes. The vector length is equal to the number of pixels processed in parallel. This is just an architecture parameter selected according to the specified performance requirement. This parametric approach offers a generic and flexible solution for different performance nodes. This means that even for different vector dimensions, the software (especially the kernels) does not require major modifications. Usually, the parts of software that need to be modified are the parts that process the borders of the picture (padding, clipping, etc.).

In order to eliminate the need for major software modifications, the software must be written in such a way that the used vector length is independent from the parameters of the application. For example, in the case of the pixel-based class, the shape of the filtering support and the number of pixels in the support should not affect the software. The vector instruction set enables that, but requires rewriting an algorithm in a vector form, or algorithm vectorization.

Vectorization is a set of (loop) transformations. In case of our two algorithmic classes, the applied transformation is loop tiling and unrolling. In this section, we show how

loop unrolling can be applied to vectorize the two algorithmic classes. For each of the classes, we begin with the generic recipe indicating which loops should be unrolled. This is followed by an example. In order to present the vectorization, we need a number of definitions that are provided in the next subsection.

3.2.1 Definition of a Block-Of-Interest (BOI) and access types

In block-based algorithms, e.g. motion estimation and compensation, a picture is split into blocks and the block grid (coarser than the pixel grid) can be defined at the picture level. Usually, a block is defined as a group of 8*8 pixels [32, 124]. From the architecture point of view, in a common case, a vector contains 8, 16, 32 or 64 pixels. The question is how to organize the pixels within a vector. Because of the block constraints, the vector in general needs to be two-dimensional. The typical vector dimensions are 8*1, 8*2, 8*4, 8*8 pixels. We will devote some additional attention to this in Subsection 3.2.3. In pixel-based algorithms, there is no block constraint and pixels can be grouped and stored in a one-dimensional vector. The vector consists of a number of pixels organized in one line, typically, 8*1, 16*1, 32*1, 64*1 pixels.

The datapath accesses pixels from the (local) storage. The accessed group of pixels we call the *Block-Of-Interest (BOI)*. This BOI is stored in a vector and processed by the datapath. Following from our discussion in the previous paragraphs, a BOI can be one- or two-dimensional (1D or 2D BOI). Thereby, in this thesis, we use rectangular access patterns. We distinguish between two types of accesses, *aligned* and *un-aligned*, which are defined as follows. The access is aligned if it is possible to arrange one or more tiled BOIs such that they fully coincide with the block-grid. Opposite, if no tiled arrangement can be found, the access is un-aligned. Figure 3.5 clarifies the definition.

According to our approach, a memory subsystem should provide to the datapath only the pixels that belong to a BOI, regardless of the type of the access. The pixels that belong to a BOI, are directly processed by the datapath and we call them the *useful* pixels. This approach omits the shuffling of the data within the datapath. Thereby, a BOI is defined as

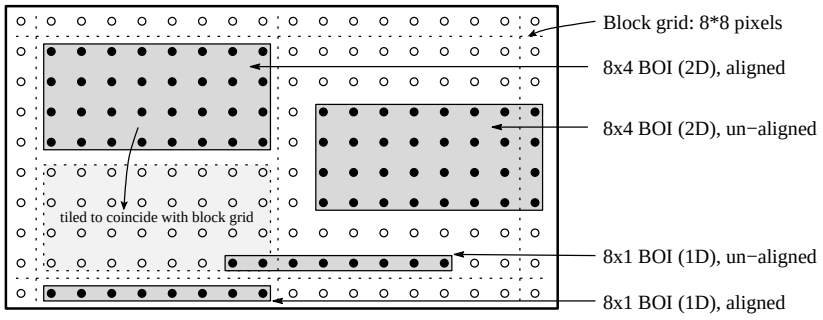


Figure 3.5: The difference between aligned and un-aligned types of accesses. The illustration is provided for the two common cases: 1D and 2D rectangular BOI.

a group of useful pixels provided by the memory subsystem and directly processed by the datapath. In Figure 3.5, the useful pixels are marked as black circles, which are part of a BOI.

The part of an algorithm that has the most influence on the performance is the kernel. This piece of code is executed the largest number of times. Here we demonstrate the vectorization of kernels on one representative example per algorithmic class. The example kernels include bi-linear interpolation used in block-based motion estimation and compensation [129], and a combination of content-adaptive spatial up-scaling and sharpness enhancement [45] used in the pixel-based class.

3.2.2 Vectorization of pixel-based algorithms

This algorithmic class does not impose any restriction on the dimensions or the aspect ratio of the processed block of pixels. The only constraint is that the number of pixels within such a block is equal to the vector size. When vectorizing this algorithmic class, we have a freedom to select the aspect ratio of the processed block of pixels, such that is most suitable for implementation.

One of the most important issues of vectorization of the pixel-based class is the support of various shapes of the filter supports. For example, a representative of this class, a content-adaptive de-blocking algorithm [82], uses a 13-pixel diamond filter support to interpolate the central pixel. The most obvious way of addressing this filter support would be to fetch those 13 pixels and perform the interpolation. However, as mentioned before, other filter supports containing different number of pixels are also present within this class (for example, a square support containing 9 pixels). Following the idea of fetching all the pixels from the support implies that the memory subsystem would have to support a number of filter supports each possibly containing a different number of pixels. The same holds for the datapath. Clearly, such an architecture would neither be efficient nor generic.

We opt for an approach where a vector consists of a number of pixels arranged in one horizontal line, and thus is one-dimensional. Other arrangements are also possible. However, the simplest way of vectorizing this algorithmic class is the case when vectors contain 1D blocks of pixels. The number of vectors participating in interpolation is equal to the number of pixels in the filter support. The process of vectorization is illustrated using the following three snapshots of a generic pseudo-code.

For the sake of clarity, we repeat the pseudo-code of a starting point, being a content-adaptive filtering as a representative of the pixel-based class.

Pixel-based algorithms: (e.g. filtering)

```
for (each interpolated pixel I ∈ picture) {   LOOP TILING (1D)
   I = 0 ;
   for (each pixel C_i ∈ support(I)) {
      C_i = fetch_unaligned (support(I), i) ;
      I = I + c_i(I)C_i ;
   }
   normalize(I) ;
 }
```

We begin with 1D loop tiling, applied to the outer-most pixel loop. The tile size is equal to the vector dimension. The result is indicated in the following pseudo-code.

Pixel-based algorithms: Intermediate step

```
for (each interpolated vector Ī ∈ picture) {
   for (each interpolated pixel I ∈ Ī) {   LOOP UNROLLING
      I = 0 ;
      for (each pixel C_i ∈ support(I)) {
         C_i = fetch_unaligned (support(I), i) ;
         I = I + c_i(I)C_i ;
      }
   normalize(I) ;
   }
 }
```

The next step is completely unrolling the new loop. All the pixels from the unrolled loop are stored in one vector. The pseudo-code can now be rewritten in a vector form:

Pixel-based algorithms: Vectorized

```
for (each interpolated vector Ī ∈ picture) {
   Ī = 0̄ ;
   for (each vector C̄_i ∈ vectorized_support(Ī) {
      C̄_i = fetch_unaligned (vectorized_support(Ī), i) ;
      Ī = Ī + c_i(Ī)C̄_i ;
    }
   normalize (Ī) ;
}
```

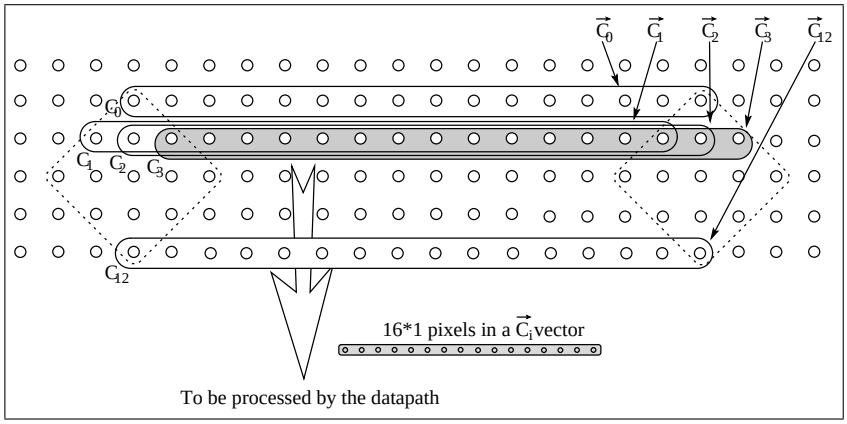


Figure 3.6: The vectorization of a pixel-based spatial up-scaling algorithm. In this example, the datapath processes 16 pixels in parallel and the dimensions of the fetched block are 16*1 pixels.

The vectorization of the pixel-based class can be illustrated on the example of the de-blocking algorithm using the diamond shaped support containing 13 pixels [82]. Each pixel within the support marks the starting pixel of a vector. Therefore, 13 vectors of input pixels are used in the interpolation. As a result, the interpolated vector is computed. Naturally, the width of this vector is the same as the width of any of the 13 input vectors. There is one additional peculiarity related to this example. This pixel-based algorithm is content-adaptive, meaning that each pixel belongs to a different class and is interpolated using a different set of coefficients. The consequence of this is that a multiplication of a vector by a vector is used. In pixel-based algorithms that are not content-adaptive, a multiplication of a vector by a scalar is used. Cost (power, area) of a multiplication by a vector and by a scalar does not significantly differ. However, here we do not tackle the memory subsystem needed to deliver coefficients to the vector multiplier. In order to interpolate a pixel marked with $I_{i,j}$, the following operation is performed:

$$I = \sum_{i=0}^{N_{TAPS}-1} c_i(I)C_i \quad \text{where} \quad 0 \leq i \leq 15 \text{ and } N_{TAPS} = 13 \tag{3.1}$$

As we have mentioned before, our pixel-based class does not contain algorithms that are recursive. Thereby, there are no data-dependencies between the interpolated pixels and couple of them may be interpolated at the same time. The immediate consequence is that there are practically no limitations in organization of pixels within a vector (within the limits of picture). In other words, pixels may be organized in a 1D or a 2D block of data of any aspect ratio as long as the total number of pixels equals the vector length. For a memory sub-system and a programmer, 1D is typically an easier scenario, and here we use it as well. Since we are using a vector length of 16, a vector contains 16*1 pixels organized in one horizontal pixel-line. This is illustrated in Figure 3.6. Using generic

units, we realize the mentioned interpolation through the set of commands given in the following pseudo-code.

```
for (each interpolated vector I⃗ ∈ picture) {
   I⃗ = 0⃗ ;
   for (i = 0; i < N_TAPS; i++) {
      I⃗ = I⃗ + c⃗_i(I⃗)C⃗_i ;
   }
   normalize (I⃗) ;
}
```

3.2.3 Vectorization of block-based algorithms

In case of block-based algorithms, the algorithmic constraints are stronger compared to the pixel-based class. The block-based class of algorithms enables that each pixel from the block is treated in the same way, which seems to offer a direct way to vectorize an algorithm. However, the block dimensions impose some additional constraints. The width of a block is a relatively small number, typically 8 or 16 pixels. If the vector length is larger than the width of the block, the vector contains the pixels that span over multiple pixel lines, i.e. the data are two-dimensional. To illustrate this issue, we refer to the common case of SAD computation. If bi-linear interpolation is not used, the typical case of SAD window is 8*8 pixels. In such a case, the processed block can be organized as a group of 8*1, 8*2, 8*4 or 8*8 pixels, depending on the selected vector length.

The problem of vectorization of such an algorithm occurs if bi-linear interpolation is used, which is the common case, for example if motion vectors are quarter-pel accurate. In such a case, in order to produce 8*8 interpolated pixels to be used in SAD calculation, in total, 9*9 pixels need to be fetched from the memory subsystem. The dimensions of the fetched block and the block of pixels that are interpolated differ by one in both directions. This is illustrated in Figure 3.7. The following paragraphs show how block-

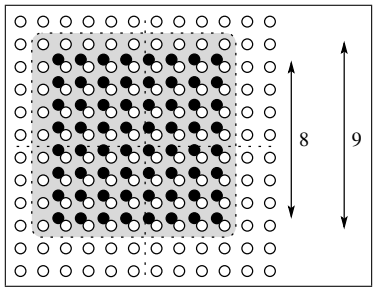


Figure 3.7: Bi-linear interpolation applied to a block of 9*9 pixels. The resulting block contains 8*8 pixels.

based algorithms, which also use interpolation, can be vectorized.

We assume that a vector contains a two-dimensional group of pixels. The number of input vectors used in the computation of one output vector equals the number of pixels in the filter support. For example, in the case of bilinear interpolation, four vectors are fetched for each interpolation. The process of vectorization of the block-based class is illustrated in the following three pseudo-code snapshots. As in the case of the pixel-based algorithms, we begin with the repetition of the pseudo-code of the block-based filtering algorithm.

Block-based algorithms: (e.g. motion estimation)

```
for (each block B(X⃗) ∈ picture) {
   for (each candidate MVC⃗) {
      SAD = 0 ;
      for (each pixel pair (C,P) ∈ (CB,PB)) {   LOOP TILING (2D)
         C = fetch_aligned (B(X⃗),n) ; // curr
         P = 0 ; // prev
         for (each pixel P_{i,j} ∈ 2*2 support) {
            P_{i,j} = fetch_unaligned (B(X⃗) + MVC⃗, n − 1) ;
            P = P + c_{i,j}(MVC⃗)P_{i,j} ;
         }
         SAD = SAD + |C − P| ;
      }
      compute_min (SAD) ;
   }
}
```

The support in the above pseudo-code has a shape of 2*2 pixels. The coefficients depend on the fractional part of the motion vector. For each of the four pixels in the support, the coefficients are constant for the entire block. The locations of pixels used in the computation, P and C pixels, are also indicated in Figure 3.3. When vectorizing this algorithmic class, we must take into account some data dependencies. Motion estimation can be recursive, an example is the 3DRS. In such a case, there are data dependencies when processing the neighboring blocks. In particular, a strong data dependency exists when processing the horizontally neighboring blocks. Thereby, we leave the motion vector loop intact and unroll the loop nested within it [1].

The intermediate phase in vectorization of the block-based class and the loop being unrolled is indicated in the following pseudo-code snapshot.

[1] The motion vector candidate loop can be unrolled as well. This can be done to a limited extent since the data dependencies do not exist for all the candidates.

Block-based algorithms: Intermediate step

```
for (each block B(X⃗) ∈ picture) {
   for (each candidate MVC⃗) {
      SAD = 0 ;
      for (each vector pair (C⃗, P⃗) ∈ (CB, PB))
         for (each pixel pair (C, P) ∈ (C⃗, P⃗)) {   LOOP UNROLLING
            C = fetch_aligned (B(X⃗), n) ; // curr
            P = 0 ; // prev
            for (each pixel P_{i,j} ∈ 2*2 support) {
               P_{i,j} = fetch_unaligned (B(X⃗) + MVC⃗, n − 1) ;
               P = P + c_{i,j} P_{i,j} ;
               SAD = SAD + |C − P| ;
            }
         }
      compute_min (SAD) ;
   }
}
```

If all the pixels from the unrolled loop are stored in one vector, the pseudo-code can be rewritten in a vector form. We keep indices to mark the vectors used in interpolation. The approach we adopt here is that four vectors are fetched, where the "top-left" pixel of a vector marks one of the four pixels participating in the interpolation. Thereby, the indices i and j take the following values: $(i, j) \in (0, 0), (0, 1), (1, 0), (1, 1)$.

Block-based algorithms: Vectorized

```
for (each block B(X⃗) ∈ picture) {
   for (each candidate MVC⃗) {
      SAD = 0 ;
      for (each vector pair (C⃗, P⃗) ∈ (CB, PB) {
         C⃗ = fetch_aligned (CB, n) ; // curr
         P⃗ = 0⃗ ; // prev
         for (each vector P⃗_{i,j} ∈ vectorized 2*2 support) {
            P⃗_{i,j} = fetch_unaligned (PB + MVC⃗, n − 1) ;
            P⃗ = P⃗ + c_{i,j} P⃗_{i,j} ;
         }
         SAD = SAD + |C⃗ − P⃗| ;
      }
      compute_min (SAD) ;
   }
}
```

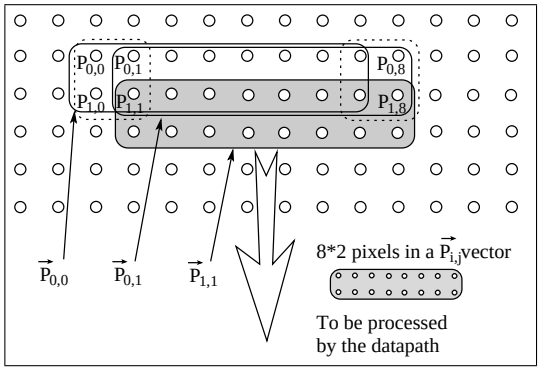


Figure 3.8: The vectorized bi-linear interpolation. Pixels marked with *P* are the original pixels participating in the interpolation. Pixels $P_{0,0}, P_{0,1}, P_{1,0}, P_{1,1}$ mark the positions of the top-left pixels of the corresponding vectors of 8*2 pixels.

Let us illustrate the above on the example of bi-linear interpolation and SAD computed on blocks of 8*8 pixels. The vector length is set to 16. To accommodate the horizontal SAD dimension, the vector is organized as a 2D block of data consisting of 8*2 pixels. Figure 3.8 illustrates the computation of bi-linear interpolation in a vector form. Four input vectors, each consisting of 8*2 pixels, are used in the interpolation of one output 8*2 pixel vector (the output is not indicated in figure). In order to interpolate any of the indicated 16 pixels, the following operation is performed:

$$\begin{aligned} &I_{i,j} = \\ &(c_0 P_{i,j} + c_1 P_{i,j+1} + c_2 P_{i+1,j} + c_3 P_{i+1,j+1})/c_4 \\ &\text{where} \\ &0 \leq i \leq 1 \text{ and } 0 \leq j \leq 7 \end{aligned} \tag{3.2}$$

where c_0, c_1, c_2, c_3 stand for four coefficients proportional to the fractional parts of motion vector. This formula can be vectorized if the pixels are grouped according to the generic recipe provided above and according to Figure 3.8. The following pseudo-code illustrates Equation 3.2 in a vector form.

$\vec{V}_{temp0} = c_0 \vec{P}_{0,0}$; $\vec{V}_{temp1} = c_1 \vec{P}_{0,1}$;
$\vec{V}_{temp2} = c_2 \vec{P}_{1,0}$; $\vec{V}_{temp3} = c_3 \vec{P}_{1,1}$;
$\vec{I} = (\vec{V}_{temp0} + \vec{V}_{temp1}) + (\vec{V}_{temp2} + \vec{V}_{temp3})$;
$\vec{I} = \vec{I} >> c$;
where,
$\vec{P}_{0,0} = \{P_{0,0}, \ldots, P_{0,7}, P_{1,0}, \ldots, P_{1,7}\}$;
$\vec{P}_{0,1} = \{P_{0,1}, \ldots, P_{0,8}, P_{1,1}, \ldots, P_{1,8}\}$;
$\vec{P}_{1,0} = \{P_{1,0}, \ldots, P_{1,7}, P_{2,0}, \ldots, P_{2,7}\}$;
$\vec{P}_{1,1} = \{P_{1,1}, \ldots, P_{1,8}, P_{2,1}, \ldots, P_{2,8}\}$;

In this case, three basic operations are used to perform the bilinear interpolation, multiplication of a vector by a scalar, vector addition and shifting (possibly with the usage of rounding) of a vector by a scalar. The factor used in shifting depends on the magnitude of the coefficients. Since we are operating on 16 pixels in parallel and the block dimension is horizontally 8 pixels, the memory subsystem has to output four blocks of 8*2 pixels, $\vec{P}_{0,0}$, $\vec{P}_{0,1}$, $\vec{P}_{1,0}$ and $\vec{P}_{1,1}$ to interpolate a block of 8*2 pixels, $\vec{I}$.

3.3 Conclusions

From the architecture point of view, we have adopted the following classification of the algorithms: Block-based and pixel-based. Representative algorithms for the first class are motion estimation and compensation. For the second class it is pixel-based filtering where the coefficients may be content dependent. From a quality point of view, the two processing classes offer acceptable quality levels of the output pictures. From an architecture point of view, they have similar requirements.

Our architecture of choice is a VLIW-based ASIP with a vector instruction set. Such an architecture is suitable for exploitation of the instruction and data level parallelism largely present in video processing applications. For the sake of demonstration, the used instruction set is generic. By using generic vector operations, the software can be written regardless of any algorithmic peculiarity such as the number of pixels in the support, the shape of the support, etc. The addition of customized instructions remains an option to further improve the performance.

The approach we adopt here is to fetch relatively large groups of pixels, for example containing 16 or 32 pixels from a local memory subsystem. The number of pixels equals the width of the datapath. After being fetched from the memory subsystem, these pixels are directly processed by the datapath that uses the vector instruction set. We have shown how both the algorithmic classes can be vectorized to match our architecture. Vectorization assumes that the pixels to be processed are fetched directly from the memory subsystem without the need for reshuffling and discarding some of the pixels within the datapath. This is enabled if un-aligned access to a pixel storage is possible. In the case of block-based algorithms, algorithmic constraints impose un-aligned access to a two-dimensional block of pixels. The requirements of pixel-based algorithms are not so strict, i.e. a block of fetched pixels can be one- or two-dimensional. For both of them, the group of accessed pixels is localized within a limited area. The processing presented here requires a customized memory subsystem that can deliver the requested pixels. Our proposition of such a memory subsystem is presented in the next chapter.

4

Memory Subsystem

THE digital revolution enabled efficient implementation of the storage elements that are essential ingredient of a memory subsystem. One of the first applications of storage elements in television was a delay-line in receivers compatible with the PAL television standard. The PAL standard achieves better picture quality than the NTSC (particularly color representation) at the additional cost of line averaging, which requires one line-delay. As a result of the digital revolution, today, a line-delay incurs a negligible cost increase, but this was different in the past. Looking back to the early days of television, television sets compliant to the PAL standard were implementing this delay in a cumbersome collection of coils and capacitors. This was soon replaced by an ultrasonic line-delay using a block of glass with two piezo-transducers glued to it. Finally, a digital line-delay enabled efficient implementation.

Line-delay from the previous example enables accesses to pixels from the previous pixel-line. This principle, with extensions to pixel- and picture-delays is widely used in the context of memory subsystems. It enables concurrent accesses to pixels from previous instances. An example that directly implements this principle we find in [125]. The cited work implements the motion estimator and uses pixel- and line-delays on-chip to enable accesses to a search area. Picture-delays are used to enable accesses to two successive pictures. Decades of research in this field, brought different memory architectures. We are going to list the different approaches in Subsection 4.2.

In the introduction of this thesis, we motivate the need for memory subsystems by referring to the well-known gap between the memory and processor speeds [12]. Let us roughly quantify the requested memory bandwidth. Here, we compute the first order approximation of the bandwidth needs of the three-frame block-based motion estimation used in high quality picture-rate up-conversion [22]. In our example, we start from the fol-

lowing assumptions. The luminance pixels are encoded with 8 bits. Assuming the HDTV standard (1920*1080 @ 60Hz), the raw input rate is equal to 120 MBytes/s per frame reference. The motion estimation uses three successive reference frames. Motion estimation repeatedly accesses pixels at two frame references while at the third one, it stays at the rate of 120 MBytes/s. In our setup, we analyze the extremely efficient motion estimation algorithm which evaluates 7 motion vector candidates. In the first order approximation, the read rate at two frame references is 7 times higher compared to the third one. In total, we arrive at the minimal read rate of (7+7+1)*120 MB/s $\approx$ 1.8 GB/s. However, motion estimation requires un-aligned pixel accesses within the off-chip memory. Assuming a 32 bit off-chip memory, the bandwidth is increased by 50% and we arrive to 2.7 GB/s as the requested bandwidth[1]. For clarity reasons, this calculation neglects some factors that influence the requested bandwidth, such as pixel interpolation. The requested bandwidth can be much larger in case of a chain of applications, or frame rate increase or resolution increase. In case of 120Hz processing, the bandwidth requirement doubles to 5.4 GB/s. In case of quad HDTV resolution (3840*2160), this bandwidth is four times larger, 10.8 GB/s. Such a large bandwidth requirement requires adequate memory subsystem.

Off-chip memory offers a limited bandwidth. To quantify the differences between the offered and requested memory bandwidth, we refer to the Digital TV (DTV) application chain example mentioned in Chapter 1. The application chain includes decoding the input video stream, de-interlacing, picture-rate conversion, spatial scaling and picture quality enhancement. These applications produce and consume the intermediate results (pictures), which are stored in the off-chip memory. Thereby, the off-chip memory bandwidth is shared by a number of producers/consumers and is estimated to 5.2 GB/s [25]. A 32 bit DDR2/DDR3 SDRAM with a bus clocked at 400 MHz offers a peak memory bandwidth of 3.2 GB/s. Assuming efficiency of 80% [25], we arrive to effective bandwidth of 2.56 GB/s. The effective bandwidth is doubled in case of 64-bit SDRAM, which is still not enough for the DTV application chain. A memory subsystem, capable of minimizing the off-chip memory bandwidth requirement is therefore essential ingredient of an SoC.

The memory subsystem proposed in this thesis bridges the gap between the requested and offered bandwidth services. It offers a solution for the two major problems To answer the high bandwidth requirement of the processing element, the proposed memory subsystem enables one- and two-dimensional un-aligned memory accesses. This omits the need for redundant fetches from the local memory and shuffling the data within the datapath. To answer the limited offered bandwidth at the off-chip memory side, the proposed memory subsystem minimizes the bandwidth towards the off-chip memory. Furthermore, it enables the tradeoff between the on-chip memory capacity and off-chip memory bandwidth, allowing a user to select the suitable tradeoff point.

This chapter is organized as follows. In Section 4.1, we analyze the application domain from a memory subsystem point of view and propose the six criteria for benchmarking a memory subsystem. In Section 4.2, we present a survey of earlier memory subsys-

[1] In a 32 bit memory, four 8-bit pixels are stored per location. To read an arbitrarily positioned 8 horizontally neighboring pixels, *three* reads from the off-chip memory are needed. The minimal read rate of 1.8 GB/s assumed two reads only.

tems. Sections 4.3, 4.4, 4.5 and 4.6 present our memory subsystem. The improvements that this work brings are quantified in Section 4.7. Section 4.8 concludes this chapter.

4.1 Application domain analysis and evaluation criteria

We start this section with a summary of the basic features of the application domain and requirements for the memory subsystem. We use the definition of a Block-Of-Interest (BOI) and aligned/un-aligned access types proposed in Subsection 3.2.1. Figure 4.1 provides a reminder.

Un-aligned read access

Chapter 3 has shown that the algorithms from the video post processing domain, require un-aligned read accesses at the side of the processing element (PE). At the off-chip memory side, the access pattern is aligned.

Number of pixels within a BOI accessed in parallel

The use-case we analyzed in the introduction of this chapter has shown that a high-quality application operating at high input resolution and frame rate imposes high requested services at the processing element (PE) side. The number of pixels fetched in parallel is dictated by the performance specifications; the typical value is 16.

The x and y dimensions of the rectangular BOI are parametric

The algorithms from the application domain require different access patterns, i.e. a BOI has different aspect ratios for different applications. For example, if a BOI contains 16 pixels, block-based algorithms (motion estimation/compensation) usually require access to a BOI of 8*2 pixels (2D BOI). The requirements of the pixel-based algorithms can be fulfilled with a BOI of 16*1 pixels (1D BOI).

Locality of reference is present in the application domain

Two major types of locality exist, the temporal and spatial [12]. The temporal locality means that it is likely that an application will reference the same block in the near future again. The spatial locality means that it is likely that the neighboring block of data will be referenced in the near future. Both of these localities are present in video post processing. For the block-based algorithms, e.g. motion estimation/compensation, the motion vectors may point to the same block multiple times (temporal locality). When processing the next block, it is also likely that some of the accessed blocks will be referenced again (spatial locality). Similar holds for the pixel-based algorithms, the same pixels are referenced multiple times while processing the current and next vector of data.

Fortunately, for all algorithms in the domain, the distance between a currently processed block and a fetched BOI is limited to an area containing few blocks of 8*8 pixels. We this area a *Window-Of-Interest*, WOI (indicated in Figure 4.1). The exact dimensions of a WOI depend on the application. For example, for motion estimation based applications, the WOI corresponds to the search area.

The BOI can be subsampled

Additionally, motion estimation in some cases requires access to a subsampled 2D BOI. Subsampling means skipping some pixels from a 2D BOI that are located at the same pixel-line or the same pixel-column. The most widely used subsampling factors are

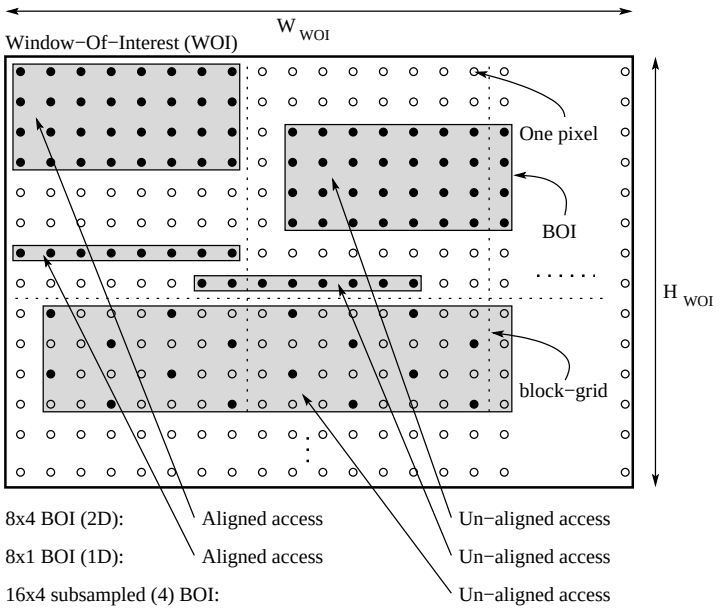


Figure 4.1: A Window-Of-Interest, split into blocks of 8*8 pixels. It shows the difference in 1D and 2D accesses with aligned and un-aligned types of accesses. The principle of subsampling is also illustrated emphasizing the term *useful pixel*. Useful pixels are denoted as black-filled circles in this figure.

two horizontally (every second pixel from the pixel-line is retrieved) and four horizontally (every fourth pixel from the pixel-line is retrieved). Often, the so-called quincunx pattern is preferred. Figure 4.1 illustrates the quincunx pattern for a 16*4 2D BOI with a horizontal subsampling factor of four. The same figure illustrates the term *useful pixel*. A pixel is useful if it can be directly processed by the PE.

Using the analysis of the application domain as the basis, we propose a set of six criteria for benchmarking a memory subsystem.

1. Minimal off-chip memory bandwidth: Our first criterion is the bandwidth towards the off-chip memory. A memory subsystem satisfies this criterion if each pixel from the off-chip memory is on the average accessed approximately once.

2. Predictability: If the performance of a memory subsystem is known at compile time, we call it a predictable memory subsystem.

3. High PE bandwidth: The processing element (PE) should not stall and the data access should not be a bottleneck. The memory subsystem should only provide the useful pixels to the PE.

4. Flexibility: The support for all the applications from the application domain requires a significant degree of flexibility from a memory subsystem. The flexibility reflects in the possibility for software changes. A list of the parameters that should be

programmable includes the following:

• The resolution of the input pictures,

• Different scanning orders, like left-to-right-top-to-bottom (LRTB), right-to-left-bottom-to-top (RLBT), meandering,

• The size of a WOI, aspect ratio of a BOI and the number of reads of a BOI.

5. Efficiency: Efficiency of a memory subsystem is judged using the two cost functions, area and power.

6. Scalability: A memory subsystem architecture is scalable with respect to cost (area, power) if it efficiently supports different problem sizes, for example WOI, BOI. Scalability is related to the hardware platform.

4.2 Prior work

The importance of memory organization in the context of embedded systems has been recognized by a large number of authors [12, 33, 121, 122, 144–148]. An exhaustive collection of different approaches available in the literature can be found in [149]. In general, we distinguish three different approaches, application-specific, general-purpose (cache based) and domain-specific memory subsystems. However, in case of memory subsystems, the border between the application-specific and domain-specific memory subsystems is not clearly defined. The concepts applied in a memory subsystem intended for one application under minor modifications can be applicable to other applications as well. For example, a memory subsystem, proposed in [125], intended for the 3DRS motion estimation algorithm as part of the picture-rate up-conversion is applicable to other motion-estimation based applications as well. Thereby, we focus on two groups, general-purpose (cache based) and domain-specific memory subsystems. Since the second group is more aligned with the work present in this thesis, we devote it more attention.

4.2.1 General-purpose memory subsystems

Over the past decades, architectures based on caches have been extensively studied and implemented [12, 121–123, 145, 146, 150–157]. The cache-based approach is the architecture of choice in the general-purpose processor (GPP) world, as it allows the coverage of a large application space. However, some quantitative characterizations have revealed that applications originating from different domains tend to utilize the memory quite differently [158]. The cache-based architectures typically try to solve the lack of performance across the entire application space by increasing the size of the cache and consecutively the area of the complete architecture.

One of the disadvantages of the cache-based approach is the lack of predictability. The performance is not known at compile time, since the processing element (PE) might stall due to cache misses. Typically, the penalty for a cache miss is an order of magnitude longer access latency than the previous (lower) memory hierarchy level [159].

Generally, three types of cache misses occur. A cold start or *compulsory miss* occurs

when the application tries to access a particular data block for the first time, because only previously accessed data is found in the cache. *Capacity miss* occurs when the cache evicts less recently used data to make room for the new data. If the same data is needed later, the PE has to wait until the data is fetched from the off-chip memory. Naturally, the capacity miss can be reduced if the cache size increases. From the cost point of view, data reorganization might be a better solution for cache miss reduction, especially in case of *conflict misses*. Conflict misses occur when the accessed data are mapped to the same cache line. Those can be reduced by introducing dummy memory words that change data layout [145]. Other data layout approaches have been proposed as well, for example, a generic method of splitting the original data arrays into sub-arrays (or tiles) of equal size in an interleaved fashion [146]. According to this method, the data should be stored in the off-chip memory in this fashion. This technique, in combination with dummy words insertion has been applied to a somewhat more real-world example, full search motion estimation at low resolution (196*256) [160]. The drawback of this method is that it requires data reorganization in the off-chip memory, which might not always be possible, especially in the context of an SoC with a lot of IPs of different vendors. The data layout can dynamically be reorganized in between computation stages [152]. Techniques that rely on data layout optimizations profit from the compile-time knowledge of the executed application. Lastly, many of the cache misses can be reduced (but not eliminated) through hardware or software prefetching [153, 154]. Especially in video processing, a number of papers reported that the usage of prefetching can significantly reduce the number of cache misses [155–157].

To alleviate the performance drop of cache-based systems for video applications, some authors go further with hardware modifications. For example, [151] proposes a reconfigurable cache design with minimal hardware changes compared to a classical cache. The mentioned design dynamically distributes the available memory into partitions that can be used for different processor activities. The processor can then concurrently access these partitions, which helps in exposing the parallelism embedded in video applications. Trimedia TM3270 [147] has a special instruction (LD_FRAC8) to perform a load combined with weighted average. This instruction is useful for bi-linear interpolation often used in sub-pel accurate motion estimation and compensation.

Intelligent prefetching, hardware changes (reconfigurable caches) and software changes (reorganization) are techniques proposed to reduce the miss rate without significantly increasing the capacity. These are the most commonly used methods to increase the efficiency of the cache-based systems. To address the predictability issue, mixed solutions, or combinations of caches and predictable memories such as scratchpads have also been proposed [161, 162]. The usage of customized memories increases the predictability. We address customized solutions with predictable performance in the next subsection.

4.2.2 Domain-specific memory subsystems

In the previous parts of the thesis, we have shown that there are two essential problems that have to be addressed by a memory subsystem, minimization of the off-chip memory bandwidth and supporting un-aligned pixel access. Most of the approaches we cite here,

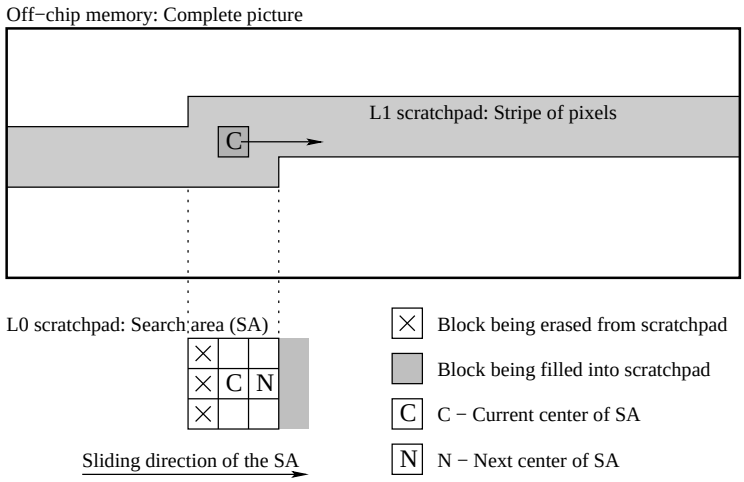


Figure 4.2: Two-level memory hierarchy used in [144]. The first level of hierarchy holds the stripe of the screen and the second one the search area of the estimator.

give more focus to only one of the two problems. Thereby, we divide them into two groups. The first group addresses the bandwidth reduction towards the off-chip memory.

4.2.2.1 Bandwidth reduction

Cache-based systems that we have addressed in the previous subsection, reduce the bandwidth towards the off-chip memory. The bandwidth can further be reduced by using customized memory subsystems. Few approaches have been proposed which achieve the theoretically minimal bandwidth towards the off-chip memory, where each pixel is fetched only once [144, 163, 164]. We illustrate one of them in the following paragraphs.

The memory subsystem used in [144] is based on a two-level on-chip memory hierarchy and the application is MPEG4 video encoding. Both levels of the memory hierarchy are organized in a block-based form. The chosen synchronization granularity is one block and the communication between the memory levels is block-based. The memory hierarchy is illustrated in Figure 4.2. The memory levels are denoted according to the notation used in this thesis which is different from the cited article.

The L0 scratchpad holds the search area of the motion estimator. As soon as the motion estimator has processed the current block (marked with *C* in Figure 4.2), the L0 scratchpad needs to be refilled and the blocks marked with *X* are overwritten by the gray-shaded blocks. The blocks used for refreshing come from the higher level of hierarchy, the L1 scratchpad. The L1 scratchpad holds just enough data to support the refreshing of the L0 scratchpad on a block-by-block basis.

The previously described approach does not offer any tradeoffs between the L1 scratchpad capacity and the required off-chip memory bandwidth. Tuan et al. [165] have proposed a one-level memory subsystem used for a full search block matcher that enables

that tradeoff. The authors offer four different levels of bandwidth savings, denoted as level-A to level-D (level-D has the minimal bandwidth). Level-D brings the bandwidth to the minimum and has the shape of the stripe which horizontally spans across the complete picture. Their level-D setup has a similar organization as the L1 scratchpad in [144, 163, 164]. The next level, level-C, offers a bandwidth overhead that is defined with the following equation:

$$BW_{\text{overhead}} \simeq 1 + \frac{SA_Y}{N}, \tag{4.1}$$

where SA_Y stands for the vertical dimension of the search area and N stands for the horizontal/vertical block dimension. In the use case addressed by the authors, $N = 16$ and $SA_Y = \{32, 64\}$. This leads to $BW_{\text{overhead}} = \{3, 5\}$. In other words, the bandwidth for level-C is three or five times higher than the minimal one (level-D). One-level memory subsystems have been used by other authors as well. For example Yang et al. use the one-level memory subsystem to store the search area of the motion estimator [166]. However, as we shall show in the next section, buffering just a search area causes bandwidth that is a few times higher than the minimal one. To conclude, the drawback of this and similar approaches is that the exposed off-chip memory bandwidth is still high.

4.2.2.2 Un-aligned data access

For the picture-rate conversion application at SD resolution (Standard Definition, 720*576 @ 50Hz), a hardwired solution to provide motion compensated 100Hz output has been proposed in [125]. The off-chip memory consists of two cascaded memories storing complete video fields. The order in which the pixels are stored in these two memories is the same as the order in which they are read. To double the input data rate, the data is read twice from the first memory. Thereby, the second memory operates completely at the output pixel rate and provides the references to the previous and current field necessary for motion estimation/compensation.

This IC is based on a one-level memory subsystem which consists of line- and pixel-delays. These delays enable accesses to individual pixels through a so-called switch matrix. The switch matrix has access to each pixel since it is connected to all pixel-delays. This memory subsystem operates at the output pixel-rate and has the capability to deliver one pixel at a time to the datapath. The architecture evaluates four motion vectors per block of pixels even though it can process only one pixel at a time. This is achieved by using subsampling by a factor of four. The pixel/line/switch matrix based memory subsystem is replicated twice to enable concurrent accesses to pixels originating from the previous and current fields.

In essence, the previous example is based on a pixel-based memory subsystem. The efficiency of such a memory subsystem can be increased if the pixel-delay elements are replaced by single-pixel wide memory banks. The usage of such memory banks enables accesses to individual pixels as well. The question is what is the minimal number of those banks.

More than three decades ago, the problem of determining a minimal number of single-pixel wide memory banks has been analyzed [167–170]. In [167], the authors support the

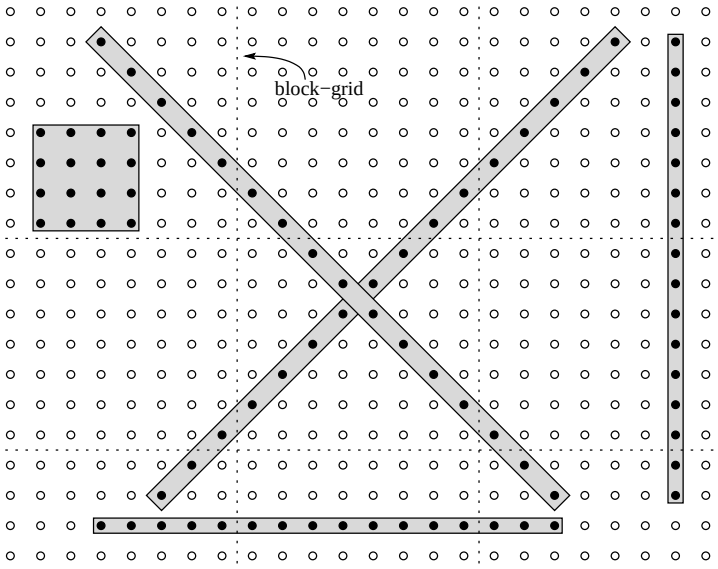


Figure 4.3: The basic access patterns consisting of 16 pixels that are enabled by the usage of 17 memory banks. The following patterns are supported: 2D block, 1D horizontal, 1D vertical, forward-diagonal and backward-diagonal.

following un-aligned 2D BOI access patterns: a 2D block $\sqrt{n}*\sqrt{n}$, column-like $1*n$, row-like $n*1$, forward-diagonal and backward-diagonal. These access patterns are illustrated in Figure 4.3. Let us denote with n the number of pixels within a 2D BOI and with m the number of used memory banks. The authors show that m and n have to be relatively prime numbers. Two integer numbers are relatively prime if their only common divisor is 1. If the access patterns are limited to the rectangular patterns, the number of needed memory banks is equal to n.

The authors in [168], [169] and [170] further improve the addressing logic. Their memory subsystems also use the mentioned prime number of single-pixel wide memory banks while the addressing efficiency is increased. The addressing in [170] is simplified since some of the computations are replaced by two look-up-tables located in two SRAMs. All the access patterns from Figure 4.3 are supported.

The idea of single-pixel wide memory banks has been exploited by more authors than mentioned here. A comprehensive review and analysis of various methods can be found in [170]. In the following few paragraphs, we address a few practical implementations of the single-pixel wide memory banks memory subsystems.

To benefit from two worlds, a combination of a cache-based approach and a customized memory approach has been proposed [162]. A twofold memory subsystem results, which is utilized by an array of four identical datapaths. The memory subsystem consists of a customized memory called the matrix memory and a small set-associative cache. The matrix memory, used for regular access patterns, provides access to an un-

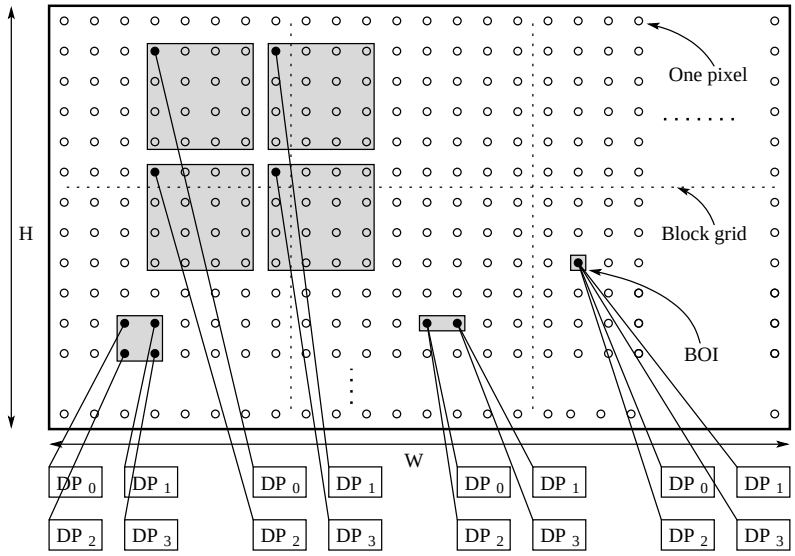


Figure 4.4: The matrix memory and possible access patterns proposed in [162]. From left to right: 2 x 2 adjacent pixels, 2 x 2 pixels with distances between them, broadcast of a 2 x 1 pixel matrix and broadcast of a single pixel to four datapaths.

aligned 2D BOI with the support of subsampling. For irregular access patterns, each of the four datapaths has a private access to a conventional set-associative cache. In the context of our work, we continue with analysis of the matrix memory organization.

The matrix memory is organized in 3*3 memory banks, each bank holding 256 16-bit pixels. It allows access to a 2D BOI of 2*2 pixels in one clock cycle. These four pixels are consumed or produced, in parallel, by the four datapaths. The matrix organization of memory banks coupled with the specific matrix dimension makes subsampling feasible. The addressing is realized through division and modulo operations, where the divisor is equivalent to the matrix dimension. The vertical and horizontal addressing of the matrix rows and columns are independent from each other, which enables independent control of the horizontal and vertical position of the top-left corner of the 2D BOI and the subsampling factor. We note that a subsampling by a factor of three is not possible in this architecture, since in this particular case a bank conflict occurs. This is not a drawback of the architecture as the most widely used subsampling factors are two and four. Lastly, since the access network between the datapaths and the matrix memory is fully connected, broadcast of the retrieved data to two or four datapaths is also possible. The possible access patterns are illustrated in Figure 4.4.

In order to increase the parallelism, i.e. the number of pixels within a 2D BOI, the matrix dimension must increase. The next configuration is a matrix of 5*5 memory banks offering a parallel processing of 16 pixels (2D BOI contains 16 pixels). The authors published this particular case in [122].

4.3 The memory hierarchy

The introduction of this chapter has shown that there is a gap between the offered and requested memory access services. A memory subsystem illustrated in Figure 4.5 is introduced in order to bridge that gap. In order to arrive to an efficient memory subsystem, the following questions arise.

- How many levels of memory hierarchy are needed?
- What is the capacity of each level of the memory hierarchy?
- How wide is a read/write access at each level of the memory hierarchy?
- Where to place a block within each level of the memory hierarchy?
- When to do an update of each level of the memory hierarchy?

We use two basic techniques within our memory subsystem. The first one enables unaligned read data access by using a customized organization of pixels to reduce the cost. The second technique reduces the off-chip memory bandwidth. These two techniques are presented in two sections, as separate hierarchy levels. Afterwards, a comparison of the 2-level and 1-level memory subsystems is provided. Since both are based on the same techniques, they require the same off-chip memory bandwidth and deliver the same bandwidth to the processing element. The comparison concludes that in most of cases, 1-level memory subsystems is superior with respect to area, power, performance, programming effort and algorithm requirements. We shall also list the exceptions.

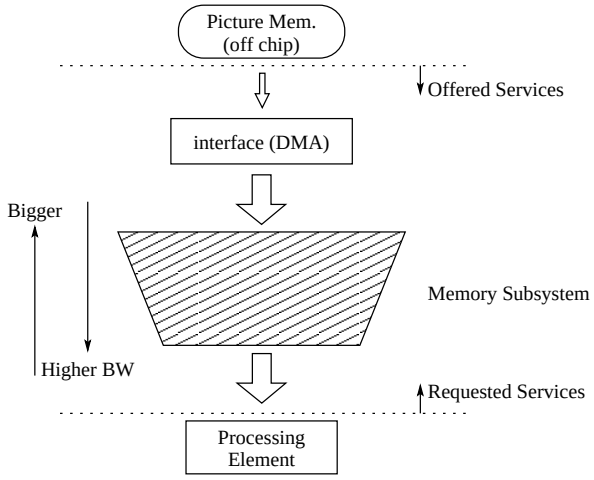


Figure 4.5: The memory subsystem bridges the gap between the offered and requested services. The width of the two major access points is defined according to system constraints (the offered services by the off-chip memory) and the application demands (the requested services by the Processing Element). The DMA provides an interface between the external world and the memory subsystem and adjusts the data access points widths.

4.3.1 The DMA interface

Picture 4.5 shows that an interface is present between the memory subsystem and the off-chip memory. The purpose of this interface is to enable direct memory accesses (DMA) to the off-chip memory. The DMA interface is capable of fetching and storing two-dimensional blocks of data from and to the off-chip memory. The idea is to offload the processing element from these tasks. Among other parameters, the processing element specifies the transferred block dimensions and the source and destination addresses.

For the purpose of this thesis, it is important to mention the other function of the DMA, namely, data width conversion. For performance reasons, the internal data width of the memory hierarchy is much larger than standard 32 bits. On the other hand, the data width of the surrounding system is typically 32 bits or occasionally 64 bits (e.g. the AMBA AHB bus connecting a number of components within an SoC [171]). The DMA has internal buffers which are used for data width conversion. The initial specifications of the DMA interface used in this thesis has been proposed by the author, which has been followed by two designs [172, 173].

4.3.2 Basic bandwidth calculus

According to a generic guideline, smaller memories are faster, and dissipate less power. Using this principle, we arrive to a hierarchy of memories with different speeds and capacities [12]. Let's first discuss the level 0 (L0) of the memory hierarchy, closest to the processing element (PE). For performance and predictability reasons, L0 misses are not acceptable during the computation of a block. Therefore, the capacity of the L0 scratchpad must at least be large enough to hold the complete WOI. Until Section 4.6, we set its capacity to exactly match the size of a WOI. This means that it contains all the data needed to compute a block. The size of a WOI depends on the length of motion vectors and the dimensions of the filter support. The question is whether to use a next level of the memory hierarchy, namely level 1 (L1) or directly fetch pixels from the off-chip memory.

The advantage of a L1 memory is a reduction of the number of accesses to the off-chip memory and the corresponding gain in dissipation. To quantify the need for the next level of the memory hierarchy, we compute the number of accesses to the off-chip memory without the usage of the L1 scratchpad.

The equations will be derived in terms of number of transfers (accesses) of blocks of pixels (usually, a block contains 8*8 pixels). To derive the equations, we use the following notation. $(L0_X, L0_Y)$ denotes the dimensions in terms of blocks of a 2D array representing a part of the picture (Window-Of-Interest) containing all the data needed for the computation of a complete block. For example, in case of motion estimation, it is the search area. $(L2_X, L2_Y)$ denotes the dimensions of a 2D array representing the picture stored in the off-chip memory. Note that the $(L1_X, L1_Y)$ is reserved for the L1 scratchpad. This notation reflects the algorithmic level only and does not relate to the physical dimensions of the used memory banks. The numbering ($L0$, $L1$, $L2$) clarifies the level of the memory hierarchy. The lower number means closer to the processing element.

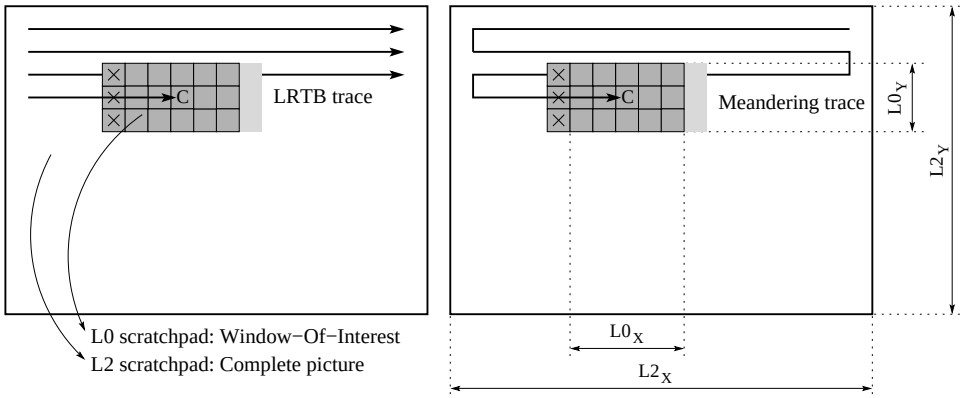


Figure 4.6: The processing (scanning) of the picture. Two common traces, left-to-right-top-to-bottom and meandering are shown. The common case of the L0 scratchpad update is also illustrated: block-column update. The notation used for the bandwidth calculation is also indicated.

Figure 4.6 illustrates two common scanning traces, left-to-right-top-to-bottom (LRTB) and meandering. Here, we derive the equations for the number of accesses within the off-chip memory for both. To simplify the derivation of these numbers, we neglect some boundary effects. We assume that each processed block is a center of a L0 scratchpad. This also holds for the blocks located at or close to the picture borders. This is equivalent to the case when the picture is extended to the left and right by $\lfloor L0_X/2 \rfloor$ blocks and to the top and bottom by $\lfloor L0_Y/2 \rfloor$ blocks. The number of accesses, NA, within the off-chip memory (denoted as L2) for the LRTB and meandering scanning trace of the L0 scratchpad are defined with the following two equations.

$$NA_{LRTB} = L2_Y(L2_X - 1)L0_Y + L2_Y L0_X L0_Y \tag{4.2}$$

$$NA_{MEAND} = L2_Y(L2_X - 1)L0_Y + (L2_Y - 1)L0_X + L0_X L0_Y \tag{4.3}$$

The first term in both equations stands for the block-column update, which occurs $L2_X - 1$ times per block-row. There are $L2_Y$ block-rows per picture. The last term in the first equation stands for a complete L0 scratchpad refill, which occurs $L2_Y$ times per picture. In case of meandering scan, this term is replaced by the block-row update ($L2_Y - 1$ times) and a complete L0 scratchpad refill (only once). In the first order approximation, the number of accesses for the LRTB and meandering scanning traces is equal to $L2_Y L2_X L0_Y$. The minimal number of the off-chip memory accesses, i.e. when each block is accessed only once, is equal to: $NA_{MIN} = L2_Y L2_X$. Thereby, the bandwidth between the off-chip memory and the L0 scratchpad is approximately $L0_Y$ times bigger than the minimal one.

We could also reach this conclusion by looking at Figure 4.6. While meandering horizontally, or performing the LRTB scan, the predominant type of L0 refill is a block-column refill. This means that for each processed block, for a majority of cases, $L0_Y$

blocks have to be read. That is why, in the first order approximation, each block within the off-chip memory is accessed $L0_Y$ times.

In Appendix A, the derivation of equations for LRTB and meandering traces is completed. It shows that the LRTB scanning trace requires more bandwidth, with the typical difference being in the range of 5-10%. The main reason for this is that in case of meandering, at the end of each block-line, a block-row refill occurs followed by the change of the scanning direction. The refill of the complete L0 scratchpad happens only once (the first refill). In case of the LRTB scan, the complete refill of the L0 scratchpad occurs at the beginning of each block-line.

4.3.3 Used memory models

To analyze the cost of a memory subsystem, we use the memory models from NXP [174]. The data is based on high speed, ultra high density, single port SRAM in 90nm CMOS memory technology. When calculating power, in the first order approximation, we assumed that writes and reads consume equal amount of power. The cost of the memory banks that are the most often used in this thesis is given in Figure 4.7. The picture shows that the power and area remain relatively constant for small capacities. The reason is that for those cases, the memory cells do not contribute the most to the cost. Apart from the memory cells, each memory bank consists of the input and output line buffers and the address decoder, which are dominant in case of small capacities.

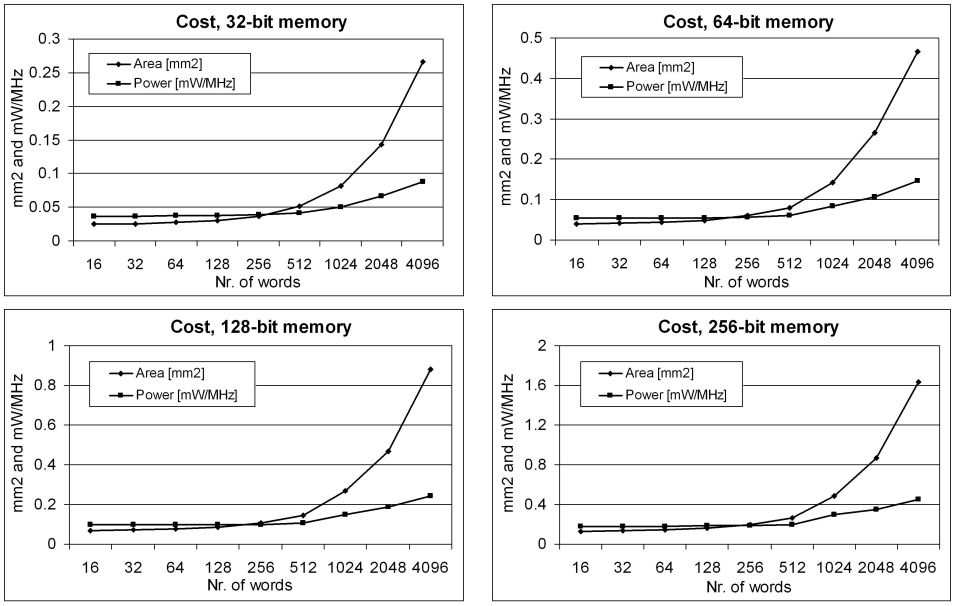


Figure 4.7: The cost of the most frequently used memory banks in this thesis.

4.4 The L0 memory

This section presents the architecture (organization of pixels) that enables un-aligned read accesses and aligned write accesses. As before, we denote it with the L0 scratchpad. With respect to cost, our basic assumption is that the memory banks are the dominant cost factors.

Recapitulating what has been explained before, the L0 scratchpad provides an un-aligned access to a 2D Block-Of-Interest (BOI) without misses for a single block calculation. As we have seen from the prior work, authors propose solutions with narrow memory banks, which are single-pixel wide [122, 162, 167–170]. In such a case, access to a multi-pixel BOI requires a large number of memory banks. As we shall show later in this section, having a large number of narrow banks is not efficient, neither from the area nor the power point of view. To cope with this, we approach the problem from a different angle. We group the neighboring pixels and store them in memory banks that hold the number of pixels per addressable location. To enable un-aligned access, the data is organized in a special way.

For better predictability and lower power dissipation we use scratchpad memories instead of caches. As a consequence, the application programmer is responsible to move data to the proper place at the correct point in time. Further, the location assignment (where to store each pixel) plays a central role. The location assignment is realized in hardware. Hardware realization maintains the sufficient flexibility to cover the applications from the domain. The advantage compared to software realization is less bookkeeping for the programmer. Software realization requires additional instructions. These instructions are only used for address manipulations and the occupied issue-slots cannot be used for some other processing. This has an impact to performance and the program code size (cost). In this and the following subsections, we analyze the location assignment in a greater detail.

We start with the classic scratchpad organization that uses single-pixel wide memory banks. According to this approach, each pixel from the reference picture is mapped to a unique memory location. This architecture has the constraint that only pixels within the same line can be accessed at the same clock cycle. The architecture will be modified to remove this constraint allowing access to more pixel-lines in parallel. To increase the efficiency, the number of banks will be reduced.

4.4.1 The classic scratchpad approach

Let us denote the width and the height of the reference picture in terms of pixels with W and H, respectively. The complete WOI, which is a part of the reference picture, holds W_{WOI} pixels horizontally and H_{WOI} vertically. The difference between the notation used in the previous section is that here we refer to pixels, instead of blocks. Similarly, the dimensions of the BOI are denoted with W_{BOI} and H_{BOI}. Based on this, we derive the physical descriptors of the L0 scratchpad such as the number of banks per scratchpad, number of pixels per bank, the depth of a bank, etc. The position of each pixel within the

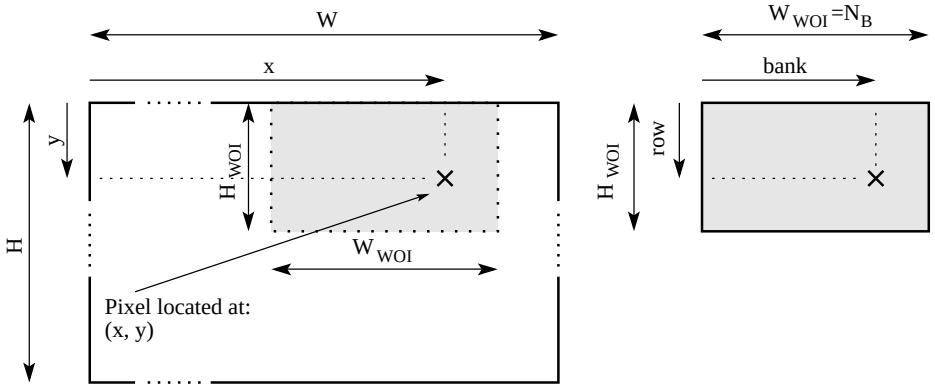


Figure 4.8: On the left-hand side, the position of a pixel within the picture is shown. It is denoted with (x, y). On the right-hand side, the location of the same pixel within the L0 scratchpad is defined with $(row, bank)$. The L0 scratchpad is realized using N_B single-pixel wide memory banks.

WOI will be described with row and $bank$ pointers.

This is a direct mapped approach where the pixel with coordinates (x, y) within the reference picture is mapped to a $(row, bank)$ address within the L0 scratchpad. We begin this analysis with a simpler case where the width of the L0 scratchpad is equal to the width of the stored WOI. Later, we shall generalize this to the case when the width of the WOI is greater than the width of the L0 scratchpad. Figure 4.8 clarifies the location of a pixel within the off-chip memory vs. its location within the L0 scratchpad.

The complete WOI is stored within the L0 scratchpad. We start with the simple L0 scratchpad architecture that uses single-pixel wide memory banks. We denote the number of used banks with N_B. The line size or the number of pixels per addressable location within a memory bank is denoted with N_P. In this case, $N_P = 1$. Accordingly, $W_{WOI} = N_B N_P = N_B$.

Here, we derive the equations defining the location of a pixel within the L0 scratchpad. Let us define the mapping. A pixel located at (x, y) position in the reference picture is stored at $mem[row][bank]$, where $row \in \{0, \ldots, H_{WOI} - 1\}$ and $bank \in \{0, \ldots, W_{WOI} - 1\} \equiv \{0, \ldots, N_B - 1\}$. The address components $(row, bank)$, defining the location of a pixel in the L0 scratchpad are given below.

$$\begin{aligned} row &= y \bmod H_{WOI} \\ bank &= x \bmod W_{WOI} \end{aligned} \tag{4.4}$$

The constraint applicable here is that only one access per bank per clock cycle is possible. However, different banks can be accessed in parallel, possibly with different addresses. Here we focus on read sequences, specifically to the 2D BOI reading illustrated

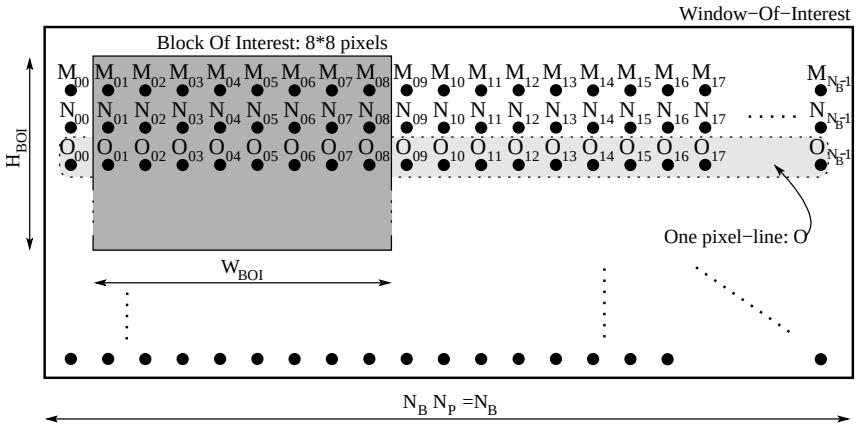


Figure 4.9: A snapshot of the contents of the Window-Of-Interest (WOI) with the 8*8 Block-Of-Interest (BOI) indicated as well. Since the used memory banks are single-pixel wide, $N_P = 1$.

in Figure 4.9. The first line of the 2D BOI ($M_{01}, \ldots, M_{08}$) can be read in a single clock cycle. However, other pixels from the 2D BOI cannot be accessed since they are located in already activated memory banks. In other words, due to the bank conflict, reading of the 2D BOI would take at least H_{BOI} clock cycles.

4.4.2 Skewing during write

The disadvantage of the previous approach was the fact that the reading the 2D BOI was sequential due to the bank conflict. In order to allow concurrent access to a 2D BOI, data can be reorganized during writing. Data can be written to the L0 scratchpad, such that it enables efficient read operations, as required by the application. Since the access pattern is rectangular, the performance of reading can be improved if the data is stored into the memory in skewed form, in anticipation of the rectangular read pattern. This concept is illustrated in Figure 4.10. The equation 4.4 is changed into:

$$\begin{aligned} row &= y \bmod H_{WOI} \\ bank &= ((y \bmod H_{BOI}) b_{\text{offset}} + x) \bmod W_{WOI} \end{aligned} \tag{4.5}$$

This equation introduces additional parameter, b_{offset}, standing for bank offset. It is defined in anticipation of the read request, and is equivalent to the number of activated banks during read operation. If W_{BOI} denotes the width of the 2D BOI, then $b_{\text{offset}} = W_{BOI}$ since banks are single-pixel wide.

The concept of skewing enables access to a number of pixel-lines concurrently while utilizing the same number of memory banks as in the classical approach. Since the data in different pixel-lines pertaining to the BOI are located in different memory banks, more than one pixel-line of the BOI can be accessed concurrently. The number of concurrently accessed pixel-lines depends on the number of available memory banks.

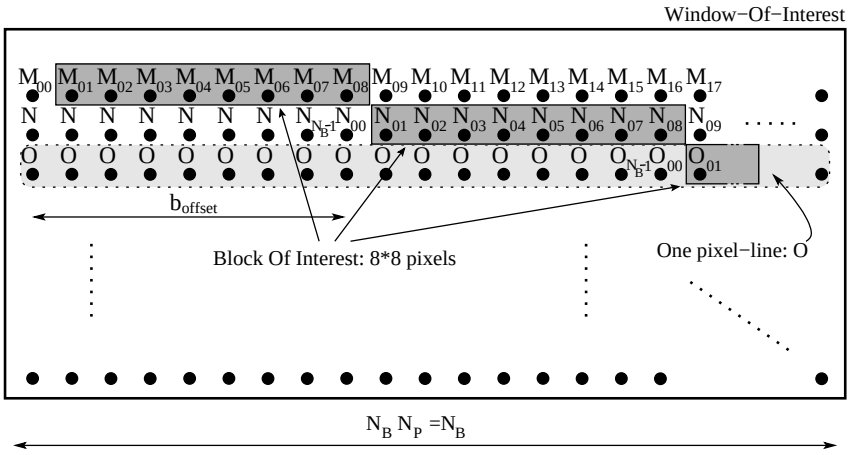


Figure 4.10: The effect of skewing applied during writing the pixels into the WOI. The consequence of skewing is increase of performance. More than one pixel-line within is accessible at the same clock cycle.

4.4.3 Reducing the number of banks

The previous solution enables high performance but its cost-effectiveness can be improved. In order to enable concurrent access to p pixel-lines within the BOI, the architecture must be based on at least pW_{BOI} memory banks. As we shall see in subsection 4.4.5, the architecture that uses narrow (single-pixel wide) memory banks is not cost-effective compared to the architecture that uses fewer multi-pixel wide memory banks. It is assumed that in both cases, the total capacity is kept the same.

Thereby, to increase the efficiency, we group neighboring pixels into a bank. In this case, $N_P > 1$, where N_P denotes the number of pixel per addressable location within a memory bank, as before. The extra level for identifying a column position of a within a bank is introduced in equations, and a pixel location is defined with $mem[row][bank][col]$, where $row \in \{0, \ldots, H_{WOI} - 1\}$, $bank \in \{0, \ldots, N_B - 1\}$ and $col \in \{0, \ldots, N_P - 1\}$. The address components $(row, bank, col)$, are defined with the following equations.

$$\begin{aligned} row &= y \bmod H_{WOI} \\ bank &= \lfloor x/N_P \rfloor \bmod N_B \\ col &= x \bmod N_P \end{aligned} \tag{4.6}$$

This location assignment is illustrated in Figure 4.11. This figure shows the L0 scratchpad holding a Window-Of-Interest (WOI) distributed along N_B memory banks of N_P pixels wide. An arbitrarily positioned BOI of 8*8 pixels is also indicated. In order to emphasize the wide banks used here, the notation of pixels is different compared to Figure 4.9.

Performance-wise, the result of this organization is equal to the case defined with Equation 4.4 and Figure 4.9 where only one pixel-line of the 2D BOI can be read per clock cycle. For instance, pixels M_{01}, M_{02}, M_{03}, M_{10}, M_{11}, M_{12}, M_{13}, M_{20} can be

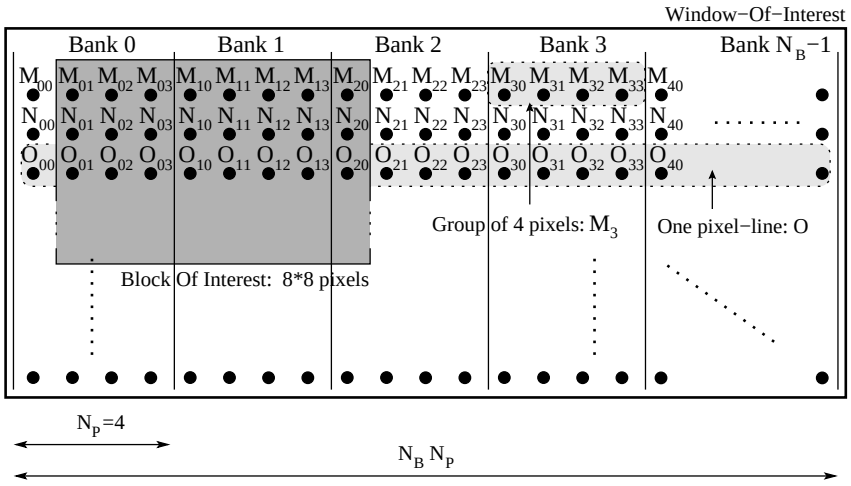


Figure 4.11: The usage of multi-pixels wide memory banks. Four pixels are grouped into one bank.

accessed in one read instruction. Note that, to read this first pixel-line from the 2D BOI, accesses to 3 banks are needed (banks marked with 0, 1 and 2 in Figure 4.11). Hence, each read action will deliver 12 pixels from which the required 8 pixels can be derived. In other words, from 8 successive read actions, 12*8 pixels are read from the scratchpad from which the required block of 8*8 pixels can be derived.

In order to improve the performance, we apply the proposed skewing concept. Equation 4.5 is appended by the *col* parameter:

$$\begin{aligned} row &= y \bmod H_{WOI} \\ bank &= ((y \bmod H_{BOI}) b_{\text{offset}} + \lfloor x/N_P \rfloor) \bmod N_B \\ col &= x \bmod N_P \end{aligned} \tag{4.7}$$

where b_{offset} has the same meaning as before but is defined differently, because of different bank organization.

$$b_{\text{offset}} = \lceil (W_{BOI} - 1)/N_P \rceil + 1 \tag{4.8}$$

Note that if $N_P = 1$, the value of b_{offset} is same as before, W_{BOI}. To be able to illustrate the concept of skewing applied to multi-pixel banks, Figure 4.11 is redrawn in a different shape. Four horizontally neighboring pixels pertaining to the same bank are grouped and represented with a single digit denoting the bank. Figure 4.12a depicts a classical scratchpad organization with 12 banks where each bank is four pixels wide ($N_B = 12, N_P = 4$). Figure 4.12b depicts a scratchpad organization with improved performance. An un-aligned block of 8*4 pixels can be accessed during one clock-cycle.

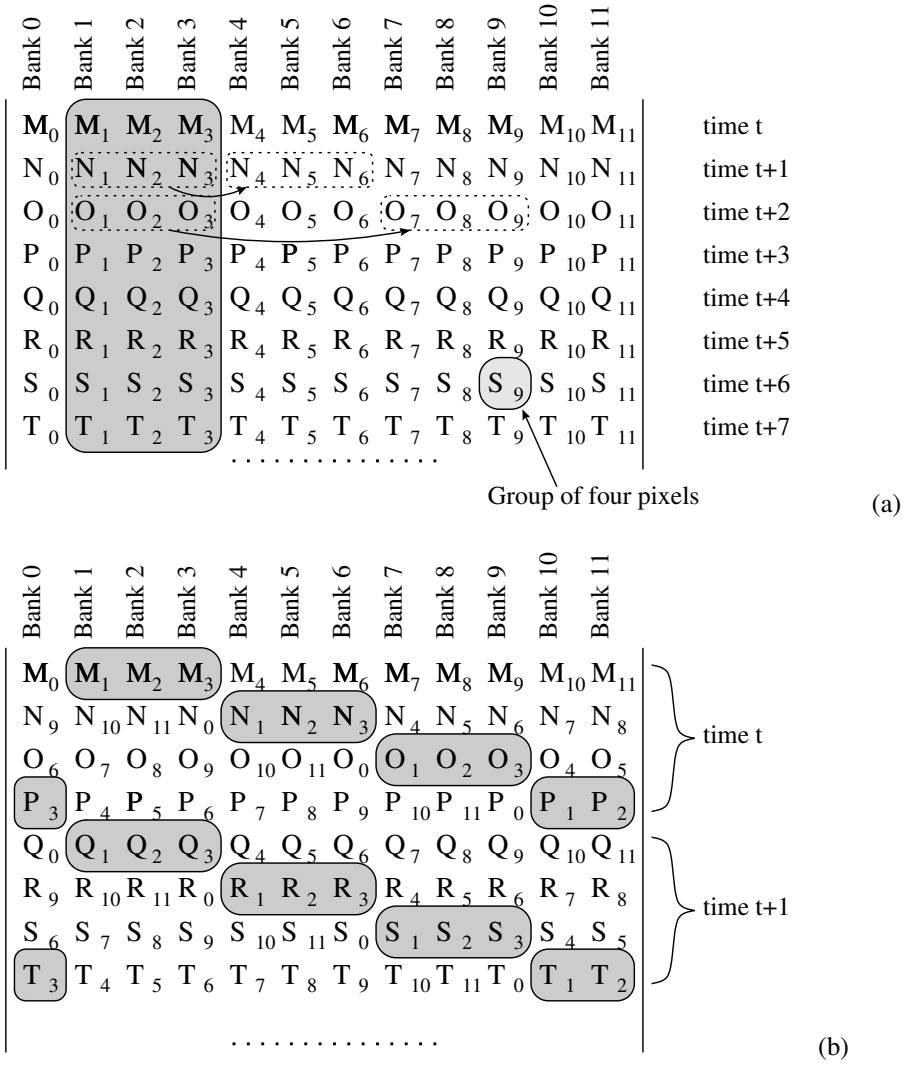


Figure 4.12: The skewing of pixels in the context of multi-pixel memory banks. Picture (a) shows the scratchpad architecture with classical data organization. Picture (b) is derived based on the picture shown under (a) by skewing the groups of four pixels during writing. Four lines can now be accessed in parallel.

Let us remove the limitation concerning the width of the L0 scratchpad. In general, the width of the L0 scratchpad is smaller than the width of the WOI, $N_B N_P < W_{WOI}$. In such a case, each pixel-line within the WOI is folded into multiple pixel-lines within

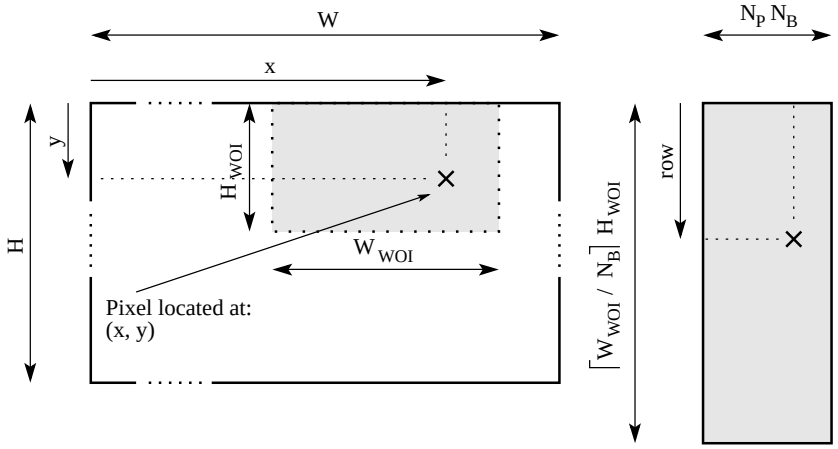


Figure 4.13: Generic case of the L0 scratchpad dimensions. The width of the WOI might not be a multiple of the L0 scratchpad's width. In case they are the same, the shaded surfaces are identical.

the L0 scratchpad. To cope with that, integer division and additional modulo operators are used. In the most general case, the width of the WOI is not a multiple of the L0 scratchpad's width. In such a case, to simplify the addressing logic, the first pixel of each pixel-line of the WOI will still be located at the first pixel location of the L0 scratchpad. Additionally, some locations of the L0 scratchpad are left empty which has as a consequence that the surface (capacity) of the L0 scratchpad exceeds the surface (capacity) of the WOI. Figure 4.13 illustrates this. Note that the surface of the WOI (indicated as the dotted shaded rectangle on the left-hand side) is smaller than or equal to the "surface" of the L0 scratchpad. Let us rewrite Equation 4.4 according to this general case.

$$\begin{aligned} row &= y' \lceil W_{WOI}/N_B \rceil + \lfloor x'/N_B \rfloor \\ bank &= x' \bmod N_B \end{aligned} \tag{4.9}$$

where $x' = x \bmod W_{WOI}$ and $y' = y \bmod H_{WOI}$. The ceiling function is necessary for cases when W_{WOI}/N_B is not integer. The capacity of the L0 scratchpad is defined with the following equation, $\lceil W_{WOI}/N_B \rceil H_{WOI}$. Thereby, $row \in \{0, \ldots, \lceil W_{WOI}/N_B \rceil * H_{WOI} - 1\}$. Similarly, Equation 4.5 becomes:

$$\begin{aligned} row &= y' \lceil W_{WOI}/N_B \rceil + \lfloor ((y' \bmod H_{BOI}) b_{\text{offset}} + x')/N_B \rfloor \\ bank &= ((y' \bmod H_{BOI}) b_{\text{offset}} + x') \bmod N_B \end{aligned} \tag{4.10}$$

The combination of using multi-pixel banks and folding the pixel-lines within the L0 scratchpad leads to a change of Equation 4.6 into:

$$\begin{aligned} row &= y' \lceil W_{WOI}/(N_P N_B) \rceil + \lfloor x'/(N_P N_B) \rfloor \\ bank &= \lfloor x'/N_P \rfloor \bmod N_B \\ col &= x' \bmod N_P \end{aligned} \tag{4.11}$$

Finally, let us rewrite Equation 4.7.

$$\begin{aligned} row &= y'\lceil W_{WOI}/(N_P N_B)\rceil + \lfloor((y' \bmod H_{BOI})b_{\text{offset}} + \lfloor x'/N_P\rfloor)/N_B\rfloor \\ bank &= ((y' \bmod H_{BOI})b_{\text{offset}} + \lfloor x'/N_P\rfloor) \bmod N_B \\ col &= x' \bmod N_P \end{aligned} \tag{4.12}$$

Where $b_{\text{offset}} = \lceil (W_{BOI} - 1)/N_P\rceil + 1$. The illustration of the scratchpad defined by the last equation is provided in Figure 4.14a. The mentioned figure illustrates the architecture of the scratchpad that can deliver two pixel-lines in parallel.

The derivation of the equations of the proposed architectures was such that only one dimension of a BOI is supported. However, the proposed architecture offers more flexibility. As long as the number of pixels within a BOI is constant, the proposed bank organization allows different BOI aspect ratios. For example, six four-pixel wide banks in Figure 4.14a support 8*2 BOI. However, 16*1 BOI is also supported. This BOI requires un-aligned access from only one pixel-line and thereby, no skewed storage is needed. However, access to five four-pixel wide banks is needed, and the architecture allows that. In order to support multiple aspect ratios of a BOI, the L0 scratchpad has to be configured according to the use case that requires the highest line-parallelism. Per each pixel-line that is accessed in parallel as part of the BOI, at least two banks are needed. Let's take as an example a 32-pixel BOI and a situation where we want to support two different aspect ratios, 32*1, 8*4. To support 32*1, we have three possibilities, 2 32-pixel wide, 3 16-pixel wide and 5 8-pixel wide memory banks. To support 8*4, only one possibility exists: 4*2 8-pixel wide memory banks. It is clear that to support both aspect ratios, architecture should be based on at least eight 8-pixel wide memory banks. More details about configuring the L0 scratchpad will be given in Subsection 4.4.5.

For convenient addressing, the width of the WOI should be a multiple of the L0 scratchpad's width, $N_B N_P$. This imposes additional constraint to the supported width of the WOI. The introduction of the set-based scratchpad relaxes this constraint.

4.4.4 The set-based architecture

To grasp the set-based concept and understand the differences compared to the previous proposals, we refer to Figure 4.14. Figure 4.14a illustrates the scratchpad defined by Equation 4.12 while Figure 4.14b illustrates the set-based architecture. Both illustrations address the case when a pixel-line is folded within the scratchpad.

In a set-based scratchpad, each pixel-line is stored within one set. Thereby, a pixel-line does not spread across all the available memory banks but only across those which pertain to the appropriate set. The consequence is that the number of concurrently accessed pixel-lines (i.e. the parallelism) is equal to the number of sets. The minimal number of banks per set is equivalent to b_{offset}. Thereby, b_{offset} constraint is embedded in the architecture and eliminated from equations. The set-based scratchpad is defined with the following

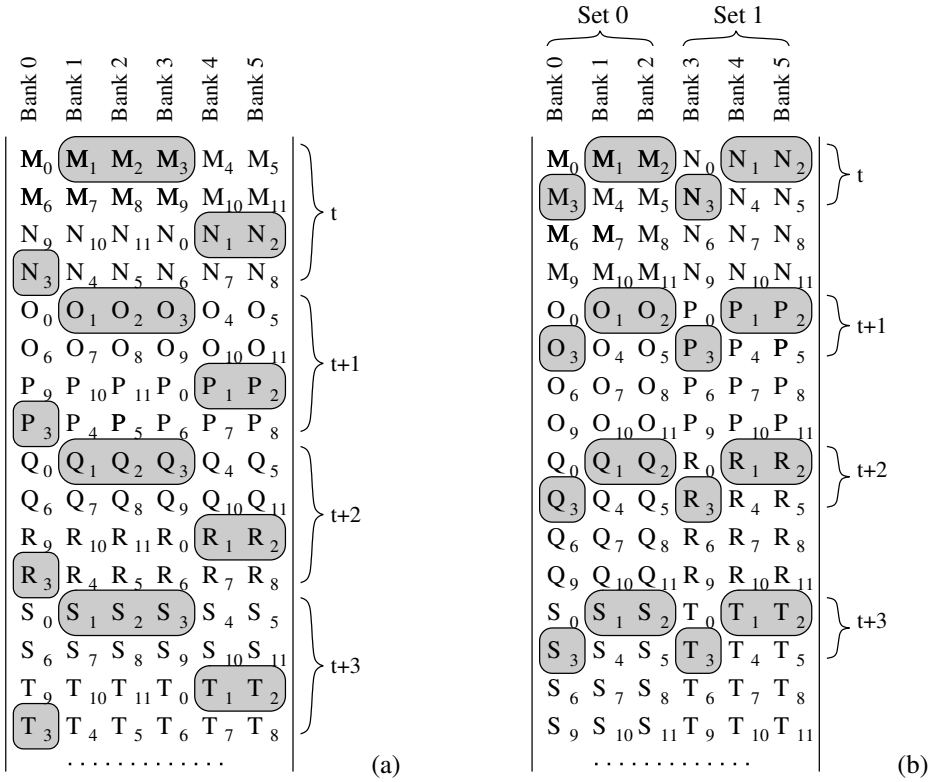


Figure 4.14: Both architectures offer the same line parallelism. Picture (b) introduces the concept of a set.

equations.

$$\begin{aligned} row &= \lfloor y'/N_S \rfloor \lceil W_{WOI}/(N_P N_B) \rceil + \lfloor x'/(N_P N_B) \rfloor \\ set &= y' \bmod N_S \\ bank &= \lfloor x'/N_P \rfloor \bmod N_B \\ col &= x' \bmod N_P \end{aligned} \tag{4.13}$$

The new parameter in equation 4.13 is the number of sets, N_S. The other parameters, N_B and N_P have the same meaning as before, with the remark that N_B stands for the number of banks per set. Each pixel-line within the scratchpad is folded and takes multiple lines in the scratchpad. Since a pixel-line spans only across the banks within a set and not across all the banks within the scratchpad, the additional benefit of the introduction of the set-based scratchpad is the increased freedom of the supported width of the WOI. The set-based scratchpad supports the width of the L0 scratchpad, W_{WOI} that is a multiple of $N_B N_P$. The previous solution supports only the widths equivalent to $N_S N_B N_P$. We can conclude that the set-based architecture offers an "N_S times" more flexible solution in terms of the supported widths of the L0 scratchpad, W_{WOI}. Additionally, the advantage

of the set-based architecture is that it can easily be configured to support different line-parallelisms. Namely, just by selecting the number of sets (which are all identical), a user can select the maximal number of pixel-lines accessed in parallel (maximal H_{BOI}).

Let us understand some drawbacks of the set-based architecture. Chapter 3 has shown that the applications from the domain require 1D and 2D un-aligned accesses. This implies that our L0 scratchpad has to support both of them. The architecture defined with Equation 4.12 offers the support of 1D and 2D access patterns, simultaneously. This means, that once the L0 scratchpad is filled-in, the user can read both, 1D and 2D BOIs. For example, the architecture illustrated in Figure 4.14a supports 16*1 and 8*2 BOI aspect ratios. The set-based architecture does support both aspect ratios, but not simultaneously. The scratchpad should be filled-in separately to support one of those two cases. The question is whether the simultaneous reading of both aspect ratios is needed by the application. Usually, the application is executed in sequence of steps and each step uses only one aspect ratio. In the remainder of this thesis, we decided in favor of flexibility of the supported widths of the WOI and in the following examples we shall use the set-based architecture. However, the choice is application driven, and the user may prefer one architecture over the other.

4.4.5 Efficiently configuring the L0 scratchpad

Here we show how the L0 scratchpad can efficiently be configured, i.e. how to define its parameters. The L0 scratchpad has three main parameters that should be configured at design time, the number of sets N_S, the number of banks per set N_B and the number of pixels per addressable location within a memory bank N_P. These three parameters should be defined for given dimensions of the specified WOI and performance (2D BOI dimensions). If we use the notation where W_{BOI} denotes the horizontal number of pixels in the 2D BOI and H_{BOI} denotes its vertical dimension, the above three architecture parameters can be defined taking into account the following constraints:

$$\begin{aligned} &N_S = H_{BOI} \\ &N_P \leq W_{BOI} \\ &N_B \geq b_{\text{offset}} = \lceil (W_{BOI} - 1)/N_P \rceil + 1 \text{ (see Equation 4.8)} \\ &N_P = 2^n; N_B = 2^m, n, m \in \mathbb{N} \\ &N_P N_W N_B N_S \geq W_{WOI} H_{WOI} \end{aligned} \tag{4.14}$$

where N_W stands for the number of words per memory bank.

The number of sets N_S is equal to the required line parallelism, or, the height of the 2D BOI. Thereby, common values for N_S are 1, 2, 4 or eventually 8, all powers of two, selected like that to fit better in the application domain requirements and simplify the address calculation. Value of N_P has an upper bound, it should always be less than or equal to W_{BOI}. Increasing it above that limit would only make the memory banks wider but will not reduce their number since the number of banks per set must always be greater than or equal to two. The number of banks, N_B, is defined based on Equation 4.8. Since

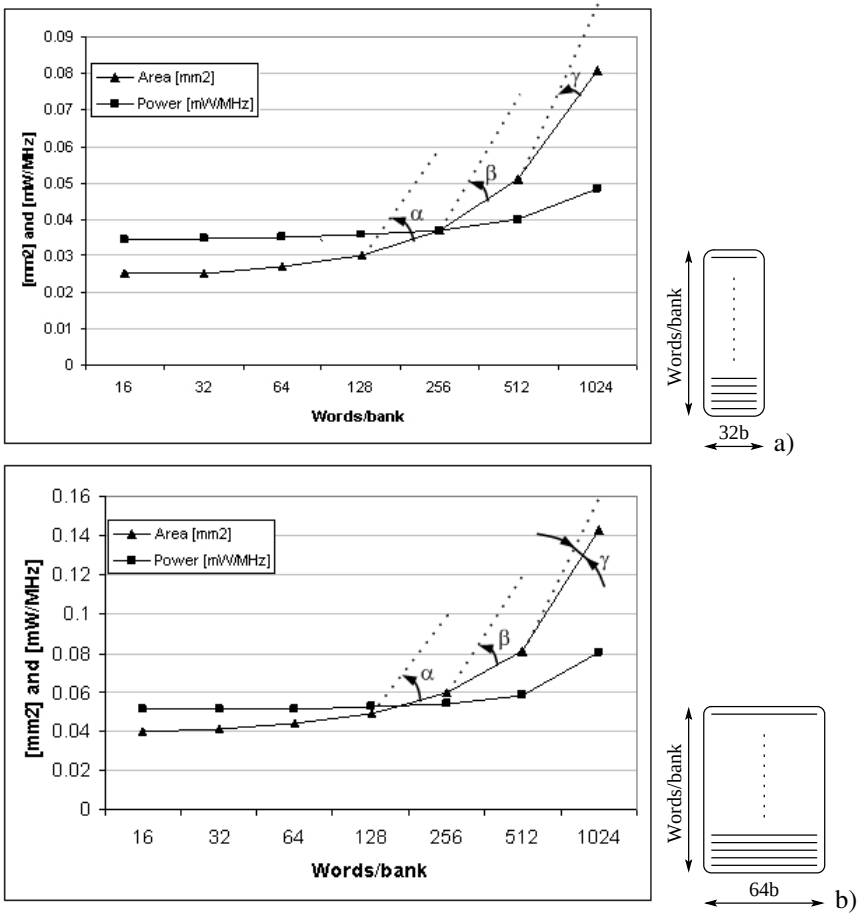


Figure 4.15: The effect the capacity of the memory bank has on the cost. Data given for the picture a) is valid for a four-pixel (32 bit) memory bank, and, picture b) for an eight-pixel (64 bit) bank. The area of memory banks with a small capacity shows almost negligible growth while the power dissipation is almost constant. The dotted line indicates the linear growth of area with capacity. An instance with 512 words per bank is more efficient in terms of area than an instance of 256 words per bank and an instance with 256 words per bank is more efficient than one with 128. This behavior is even more emphasized in case of power.

the addressing logic is based on modulo arithmetic, the implementation of the addressing logic is simplified if N_B and N_P are powers of two. Lastly, the total supported capacity should be greater than or equal to the requested capacity. These generic guidelines help in the selection of optimal N_B and N_P for a specific case.

Next we focus on the number of banks per set and pixels per bank. They should jointly be defined taking into account the cost efficiency and the number of concurrently accessed pixels within a single line. As a first guideline, the number of words per bank N_W should not be chosen too small. Indeed, since each bank is instantiated as a stand-alone mem-

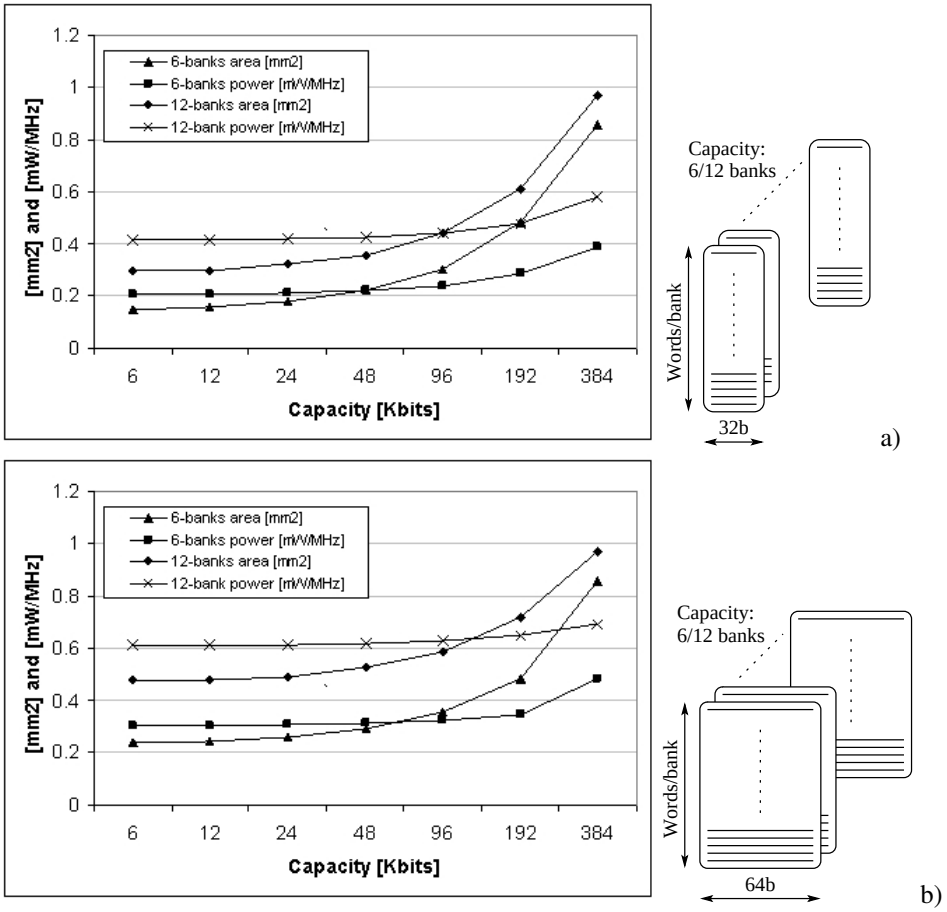


Figure 4.16: The effect the number of banks per L0 scratchpad has on the cost (area, power). The total capacity per L0 scratchpad is kept the same for all the use cases. Data given for the picture a) is valid for a four-pixel (32 bit) memory bank, and, picture b) for an eight-pixel (64 bit) bank.

ory, a small N_W leads to many small inefficient memories. The issue of inefficiency of small memory banks has been addressed in Subsection 4.3.3. The impact on cost (area, power) is also illustrated in Figure 4.15. This semi-logarithmic diagram reveals that until a certain threshold, the cost is almost constant while the capacity increases. To emphasize this effect, the graph is decorated with the dotted lines that represent the linear growth of area with respect to the capacity increase. The angle between the actual result (growth path) and the dotted line represents the measure of the efficiency of the particular point at the graph. From the graph it is clear that the capacity of the banks should be maximized. Therefore, the number of used memory banks should be minimized. Figure 4.16 illustrates this for the case of the complete scratchpad. The scratchpad that uses fewer

	$W_{BOI} = 8$					$W_{BOI} = 16$					
N_P	1	2	4	4	8	1	2	4	8	8	16
N_B	8	5	3	4	2	16	9	5	3	4	2

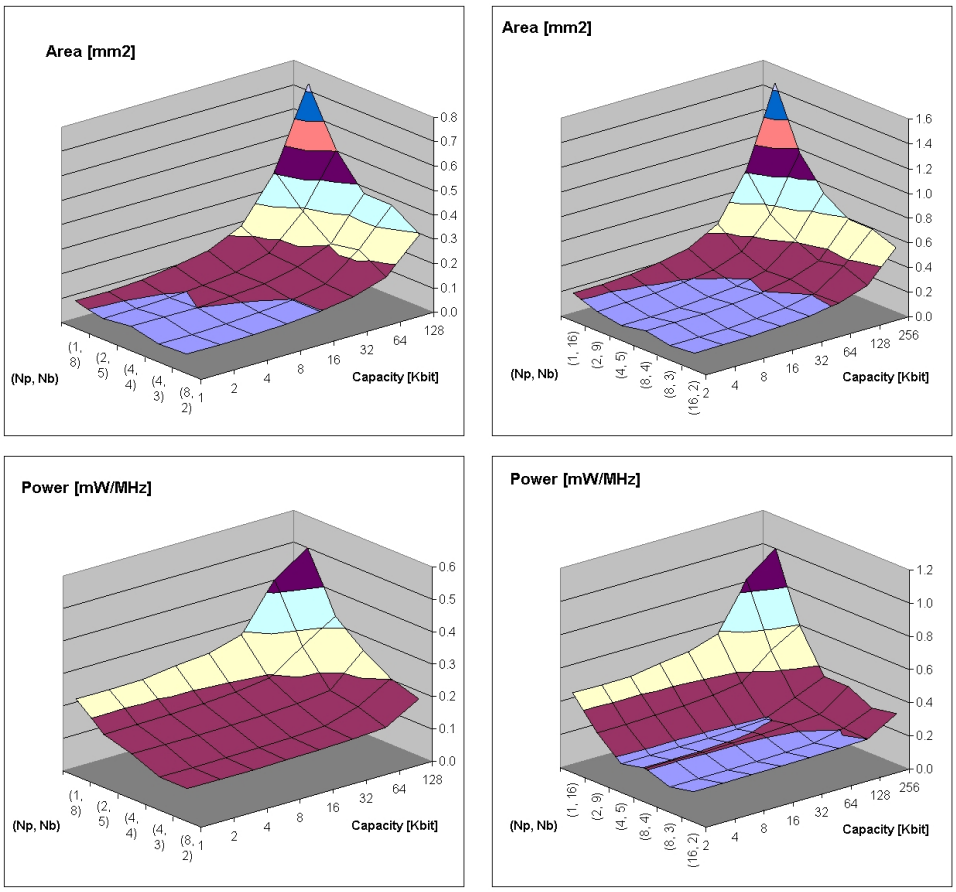


Figure 4.17: The cost of the L0 scratchpad as a function of the selected capacity and (N_P, N_B) pair. The analyzed values of the (N_P, N_B) pair are provided in the table above. The cost and capacity are given per one set of the L0 scratchpad. The left-hand side reflects the setup where $W_{BOI} = 8$. The right-hand side reflects the setup where $W_{BOI} = 16$.

banks (6 banks) is more efficient than the scratchpad that uses 12 banks. Similar results are obtained for the case of four and eight pixels per bank.

The analysis so far showed that N_P and N_B parameters should be jointly selected as a function of the specified L0 set capacity and the horizontal dimension of a BOI. The cost (area, power) is graphically presented for two cases of W_{BOI} in Figure 4.17,

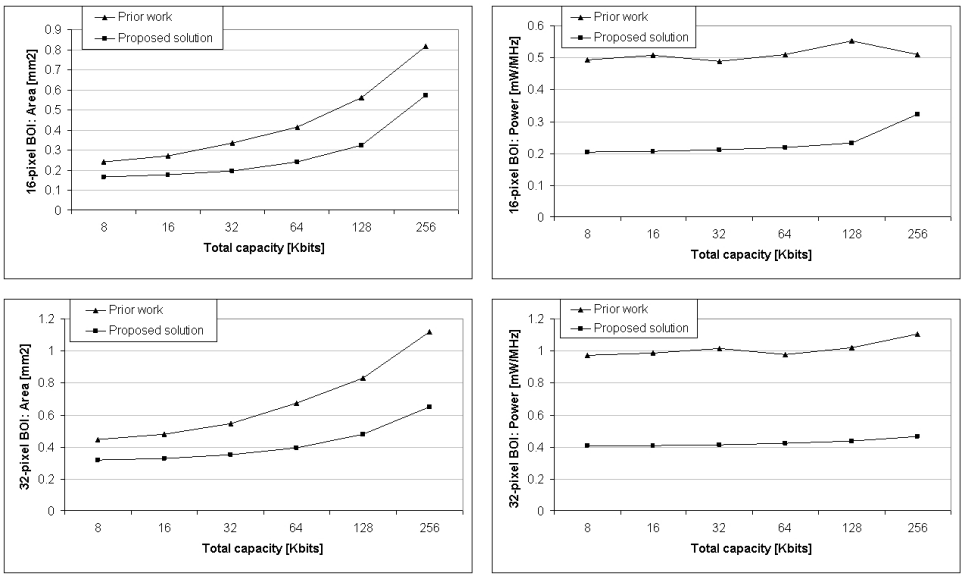


Figure 4.18: The difference in silicon efficiency between the proposed L0 scratchpad solution and the prior work [167–170] that is based on single-pixel wide memory banks. Top row presents the setup where the 2D BOI contains 16 pixels while the bottom row the setup where the 2D BOI contains 32 pixels.

$W_{BOI} \in \{8, 16\}$. The cost is plotted for a range of capacities of the L0 scratchpad. The selection of values for N_P and N_B is summarized in the table above based on constraints defined by Equation 4.14. If we keep the constraints that N_P and N_B are powers of two, the efficiency is highest when $N_P = W_{BOI}$. If we only keep the constraint for N_P, we get comparable results if $N_P = W_{BOI}/2$ and the number of banks N_B has the minimal value defined by Equation 4.14, $N_B = \lceil (W_{BOI} - 1)/N_P \rceil + 1$. Note that the resulting cost depends on the used memory technology.

In order to compare our solution for the L0 scratchpad with the prior work [167–170], we refer to Figure 4.17. This figure shows that the solutions from prior work that use single-pixel wide memory banks are not efficient. In order to quantify the numbers, we plot side-by-side the solution that uses multi-pixel memory banks and single-pixel memory banks.

Let's observe two cases, a 16-pixel BOI organized in two lines and 32-pixel BOI organized in four lines. According to the domain-specific memory subsystems, for the case of 16-pixel BOI, 17 banks are needed to enable un-aligned access to number of pixel patterns. However, since we are looking into only rectangular and row-like patterns, we assume that only 16 banks are needed. Similarly, for a 32-pixel BOI, 32 banks are needed.

According to our approach, to enable these access patterns, we use the setup where $N_P = W_{BOI}$ which results in two 8-pixel banks per set. For the case of 16-pixel BOI,

there are two sets for two pixel-lines in parallel (or four banks in total) and for the case of the 32-pixel BOI, there are four sets for four pixel-lines in parallel (or eight banks in total). Figure 4.18 illustrates the difference in cost (area and power) between our proposal and the domain-specific memory subsystems for various L0 scratchpad capacities. It shows that in both scenarios, our solution offers better area and power numbers. Section 4.7 addresses the efficiency and comparison against the prior work in more detail. A number of examples that cover small, medium and large capacity of the L0 scratchpad will be analyzed from cost point of view.

In conclusion, this architecture is modular and offers trade offs between cost and performance. Without modifying the dimensions of the Window-Of-Interest, it is possible to trade off the area for performance by adapting the number of used memory banks to the performance needs. The requested performance is specified by the number of concurrently accessed pixel-lines and the width of a pixel-line. By halving the number of concurrently accessed pixel-lines (and reducing the performance by a factor of two), we half the number of sets and consecutively the number of memory banks. We have also shown how the parameters of the L0 scratchpad can be determined such that the efficiency is maximized.

4.4.6 The addressing of the L0 scratchpad

To guarantee high performance, most of the internal addressing logic is realized in hardware leaving the user a set of API instructions. In this subsection, we provide the basic concept, while some of the additional implementation details can be found in the published implementation of the L0 scratchpad [148]. We distinguish two types of the API, the initialization and the read/write API.

Initialization API: Through the following initialization API, a sufficient number of parameters can be programmed:

1. *initWOI*: The x and y dimensions of the WOI.

2. *initBOI*: The x and y dimensions of the 2D BOI (and therefore the aspect ratio of the 2D BOI).

3. *initSUB*: The pattern applied in sub-sampling (in case sub-sampling is used)

initWOI API is used in the address computation during block-refill (write operation) and block-read (read operation). If smaller dimensions of the WOI (compared to physically available) are used, only the smaller WOI is refilled, which results in reduced bandwidth towards the next level of the memory hierarchy (the L1 scratchpad).

initBOI API sets up the parameters of the reading logic. This is used during reading of the 2D BOI. In the applications using motion estimation and compensation, the motion vector indicates the position of the requested 2D BOI. If sub-pixel accuracy of the motion vector is selected, the user provides only the integer part of the motion vector and the

necessary interpolations are performed in SW after fetching the 2D BOI.

initSUB API is used to setup the eventual use of sub-sampling. Denoting the sub-sampling factor with SSF, the following subsampling features are supported: $SSF \in \{1, 2, 4\}$. In case the subsampling is used, the popular quincunx patterns are also supported.

Read/Write API: The read/write (access) API is used to control the read and the refill of the L0 scratchpad. To simplify the addressing logic, only two possibilities for a refill exist: block-column refill and, block-row refill. The control unit (for example, the PE) supplies the refill command to the refill address logic of the L0 scratchpad. This command contains the information about the direction of sliding of the L0 scratchpad, top/bottom/left/right. Depending on this command, the refill logic determines the address of the block-column or the block-row that has to be refilled, and updates the information about the center of the L0 scratchpad. It is also possible to reset the L0 scratchpad. This command is used in case of a complete refill. Complete refill occurs at the beginning of each block-line in case of the LRTB scan and at the beginning of the first block-line in case of meandering. In this case, the reset command will be issued followed by the sequence of commands for a block-row refill or a block-column refill.

The read of the 2D BOI is performed after the read command has been issued by a user. The position of a 2D BOI is specified by a user as part of the read command. Based on this input, the addressing logic determines three parameters: The first memory bank in the set that has to be accessed, the address of each accessed bank and the offset with respect to the first pixel of the starting bank. Due to the set-based approach, these parameters are the same for all sets. This logic is realized in hardware. After the three parameters are identified, the appropriate memory banks are read and the block of $N_B N_P N_S$ pixels is available. In order to access a block of $(N_B - 1)N_P N_S$ pixels, only the appropriate pixels are kept. This is done through the limited rotation and keeping the appropriate $(N_B - 1)N_P$ pixels of each pixel-line. The same rotation is performed for each set (for each pixel-line). The number of rotation places is computed based on the position of the 2D BOI, provided as part of the read command.

The presented concepts are sufficient for un-aligned 2D access. As indicated before, motion estimation has two additional specific requests, sub-pel accuracy and usage of subsampling. Memory subsystem can be enhanced to support these two requests and reduce the amount of processing in the datapath. Sub-sampling will be addressed in greater detail in the following paragraphs. Similar concepts can be used for sub-pel accuracy.

There are two possibilities to implement the subsampling:

1. The memory subsystem returns the larger BOI to the datapath, which performs the pruning, and,

2. The memory subsystem does the pruning and returns the BOI.

We support the second possibility. In addition to the SSF parameter, subsampling is configured with one additional parameter, $START_{SET}$. This parameter specifies different starting pixels for each set. This feature also enables the quincunx pattern. The

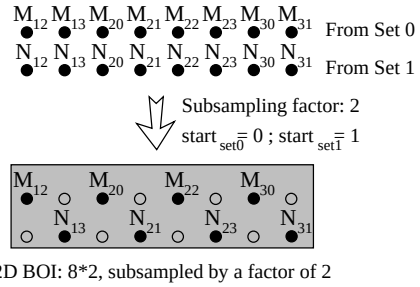


Figure 4.19: Producing the sub-sampled 2D BOI happens after reading un-aligned 2D BOI from the L0 scratchpad.

sub-sampling is applied horizontally, within each pixel-line. Vertical sub-sampling is not supported, partly because there is not enough application drive and partly to simplify the address computations and shuffling the data. Figure 4.19 illustrates the subsampling on the example of $SSF = 2$.

To clarify the reading, supported 2D BOIs and sub-sampling, we refer to a real-world example, applicable to the target application domain and set the number of delivered pixels to the PE to 16. In order to support all the 2D BOI dimensions, the number of sets is 4, number of banks per set to 2 and each bank stores 8 pixels. In such a case, the L0 scratchpad delivers the following 2D BOIs for different subsampling factors:

- SSF = 1 (no subsampling), 2D BOI is 16*1, 8*2, 4*4,
- SSF = 2 (subsampling by a factor of two), 2D BOI is 16*2, 8*4,
- SSF = 4 (subsampling by a factor of four), 2D BOI is 16*4.

It is important to note that power savings are possible in almost all the modes. As can be seen from previous expressions, in all the modes, some of the banks are not used. In such modes, the data from the inactive banks is not read.

In case of motion estimation/compensation, sub-pel accuracy is implemented through the bi-linear interpolation. To fully support the bi-linear interpolation in the memory subsystem, one additional pixel-line has to be fetched, which requires one additional set. This requires more banks, increases the complexity of the addressing logic and thereby the cost. Alternative is just to (horizontally) implement the linear interpolation as part of the memory subsystem and complete the interpolation in the vertical domain in the datapath. This tradeoff does not require any modification of the number of banks. The difference is that for each pixel-line (set) from the starting $N_B N_P$ pixels $(N_B - 1)N_P + 1$ pixels are selected. Then, two overlapping pixel groups of $(N_B - 1)N_P$ are selected and using the fractional part of the motion vector as the additional input, $(N_B - 1)N_P$ horizontally interpolated pixels are output.

4.5 The L1 memory

This section proposes the technique for bandwidth reduction towards the off-chip memory. It is implemented as a separate, higher hierarchy level. As before, we denote it with the L1 scratchpad.

A part of the picture is stored within the L1 scratchpad, which is scanned by the L0 scratchpad block-by-block. By doing so, the high memory bandwidth needed for a L0 refill (see Equations A.3 and A.4 from Appendix A) is not exposed to the off-chip memory but to the L1 scratchpad instead. The bandwidth needed to fill the L1 scratchpad is kept close to one access per pixel, which is the minimal one. In the text to follow, we analyze this bandwidth requirement in more details as well.

We use the same notation as in Subsection 4.3.2. The difference is the extra level of the memory hierarchy introduced in between the L0 scratchpad and the off-chip memory. Accordingly, the L0 scratchpad is as before the lowest level of the memory hierarchy, closest to the processing element. The next level is the L1 scratchpad, which holds part of the picture. Thereby, $(L1_X, L1_Y)$ denotes the dimensions of a 2D array representing the part of the picture stored within the L1 scratchpad. Finally, at the top of the memory hierarchy is the off-chip memory holding the complete picture. As before, it is denoted as L2. Similarly to Subsection, 4.3.2, all dimensions of the stored areas within the three levels of the memory hierarchy are given in terms of blocks. Again, they do not correspond to the physical dimensions of the scratchpads but only to the algorithmic width and height.

4.5.1 The stripe-based approach

As already discussed in Section 4.2, one of the basic concepts of utilizing the second level of the memory hierarchy is to store a number of complete block-lines, i.e. the stripe, within the L1 scratchpad. These block-lines spread horizontally across the complete picture width. The L0 scratchpad scans the picture portion stored within the L1 scratchpad block-by-block. After a block-line has been completely scanned by the L0 scratchpad, the L0 scratchpad moves one block-line down and starts processing the new block-line. Thereby, one block-line (the top-most one) within the L1 scratchpad is not used anymore by the L0 scratchpad and can be overwritten by the new data. In other words, the read and write address pointers need to be maintained in a circular fashion for the L1 scratchpad.

One of the important aspects that influences the performance is prefetching the data within the L1 scratchpad. Writing the new block-line must be done in parallel to the processing. If the vertical dimension of the L1 scratchpad is greater than the vertical dimension of the L0 scratchpad by one block, $L1_Y = L0_Y + 1$, prefetching is possible[2]. In such a case, while the data located within the top-most $L0_Y$ block-lines are being processed, the data coming from the off-chip memory can be stored in this additional block-line. The DMA can be programmed to prefetch the data. This data is then used in the next processing iteration. This strategy is illustrated in Figure 4.20.

The consequence of storing complete block-lines is that the bandwidth requirement

[2]Prefetching is possible if there are one or more blocks available. Our choice of having a complete block-line relaxes the schedule.

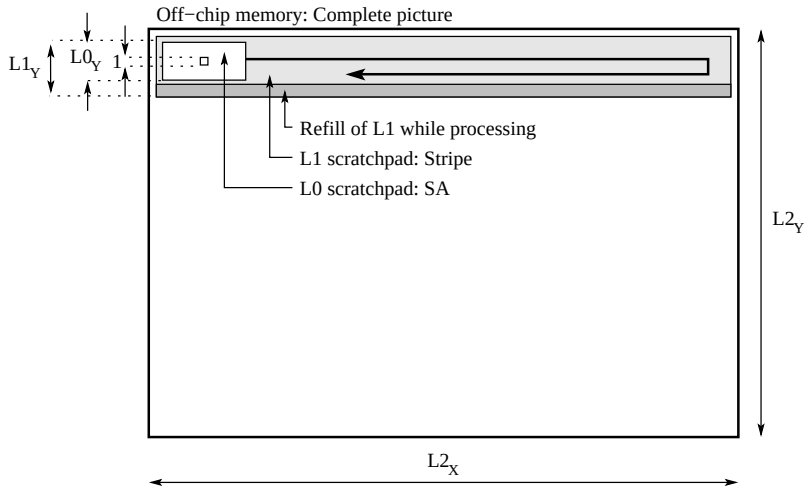


Figure 4.20: The 2-level memory subsystem using the stripe-based concept. The L1 scratchpad holds a stripe of the picture consisting of $L0_Y + 1$ of complete block-lines. The L0 scratchpad performs the meandering (or LRTB) scan within that stripe.

towards the off-chip memory is minimal (one access per pixel). In addition to the high cost (caused by the high capacity of the L1 scratchpad), the other major drawback of this approach is the lack of flexibility concerning the size of the L1 scratchpad. The vertical dimension of the L1 scratchpad is driven by the vertical dimension of the L0 scratchpad, and its horizontal dimension by the horizontal dimension of the picture. Thereby, any change in the algorithm, which increases the vertical dimension of the L0 scratchpad cannot be accommodated by this approach. For example, change of the aspect ratio of the L0 scratchpad (which might not increase the capacity of the L0 scratchpad) might cause the vertical dimension of the L1 scratchpad to become smaller than the L0 scratchpad's one. Similar situation occurs when the horizontal dimension of the picture increases.

An additional drawback is that this approach does not directly support multiple scans of the data within the L1 scratchpad. In some applications (for example motion estimation), multiple scans are required [23, 175]. Since the vertical dimension of the L1 scratchpad is kept to the minimum, i.e. $L1_Y = L0_Y + 1$, multiple scans are not possible because the L0 scratchpad has only room to scan one block-line. To enable the multiple scans, the height of the L1 scratchpad should increase, which increases the cost. To enable multiple scans within L1 scratchpad (which has the limited height, smaller than the picture), the vertical dimension of the L1 scratchpad has to increase, which has an additional impact on the cost. Alternatively, the data should be fetched multiple times to the L1 scratchpad, which increases the off-chip memory bandwidth. For example, to enable the top-down scan, followed by the down-top, performed at the level of the complete picture, the bandwidth increases by a factor of two.

4.5.2 The region-based approach

The region-based approach addresses the limitations of the stripe-based approach related to flexibility. According to the region-based approach, the complete picture is partitioned into regions (sometimes, they are also called tiles) that are fetched from the off-chip memory and stored within the L1 scratchpad. This way, the dimensions of the L1 scratchpad do not depend on the picture dimensions. The algorithm is written as follows:

```
⋮
for (Every Region) {
   ProcessRegion () ;
}
⋮
```

So, the region is defined as a portion or partition of the picture wherein the processing takes place. The algorithm executes within the limits of a particular region according to a well-defined scanning pattern. So the hierarchy of the algorithm can also be interpreted as a hierarchy of the scanning pattern. The additional advantage compared to the stripe-based approach is the fact that multiple scans within a region are possible. This concept is illustrated in Figure 4.21.

The region-based approach is quite generic. Other approaches such as the previously

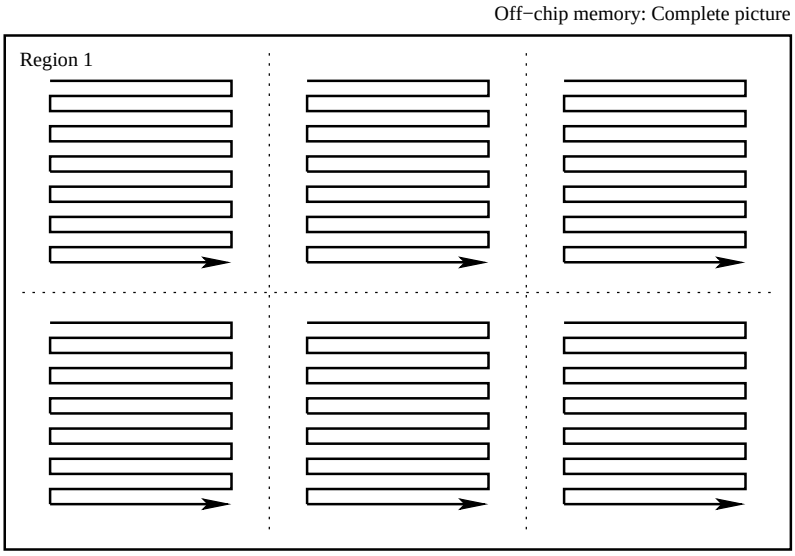


Figure 4.21: The concept of a region-based scan. The picture is divided into independent regions. The algorithm is executed within each region.

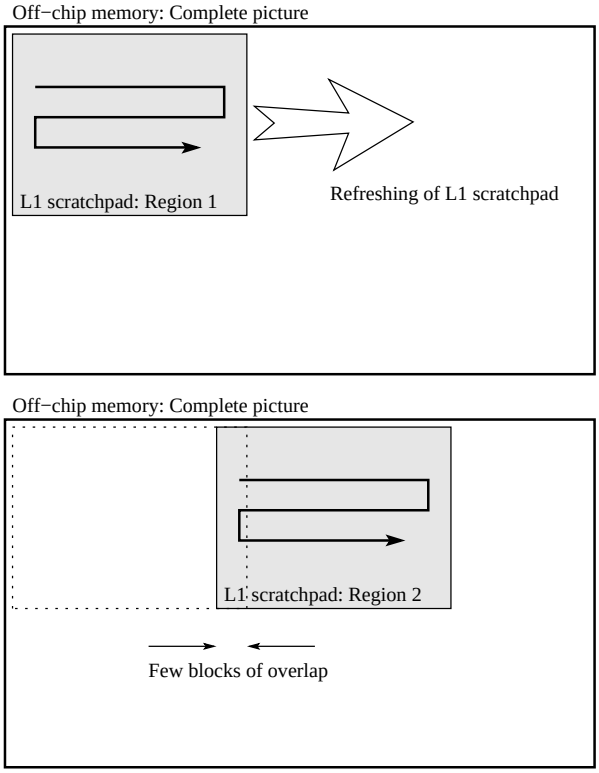


Figure 4.22: The region-based approach. The L1 scratchpad holds a region of the picture and the L0 scratchpad performs the meandering scan within the region. When region 1 is processed, the content of the L1 scratchpad is updated by the new region.

described stripe-based approach can be seen as special cases. In the stripe-based approach, there is only one scanning trace defined for the entire picture. Therefore, the stripe-based approach is algorithmically equal to the region-based approach where the dimensions of the region are equal to the dimensions of the picture.

Since there is neither direct coupling with the dimensions of the picture nor the L0 scratchpad, the solution is more flexible and open to algorithmic modifications (the L0 scratchpad size) or changes to the picture size. If the algorithm specifications or the picture dimension change, the system could still function. Furthermore, the aspect ratio of the region can be adjusted to best match the new situation keeping its capacity intact. This makes the region-based approach advantageous compared to the stripe-based approach.

After the data within the L1 scratchpad have been processed, the L1 scratchpad has to be refilled (see Figure 4.22). The problem is that the L1 scratchpad contains an order of magnitude more blocks compared to the L0 scratchpad, and refilling of the complete content requires a lot more processing cycles. Even worse, the data is usually transmitted

from the off-chip memory to the L1 scratchpad over the standard 32 bit link and the arrival of data depends on the availability of the off-chip memory and other resources. This might cause stalling of the computation and reduction of the performance of the complete video subsystem. This problem can be solved by using prefetching mechanism similar to the one described for the stripe-based approach. During the last scan of the current region, the additional block-line is used to prefetch the data from the next region.

In order to process every block within the picture and accommodate a WOI around it, regions are overlapping as illustrated in Figure 4.22. The dimensions of the region must be defined. From the cost point of view, it is advantageous to choose smaller regions but this has two negative consequences. The first consequence is that the bandwidth towards the off-chip memory increases because of the overlap between the neighboring regions. This will be detailed in the bandwidth analysis provided in Subsection 4.5.7. The second consequence is that the L0 scratchpad's scanning trace is interrupted (broken) at the region boundaries. This could degrade the performance for some algorithms, particularly motion estimation. The following paragraph tackles this degradation for the case of a recursive motion estimation algorithm, such as the 3DRS.

From the motion estimation algorithm point of view, the L0 scanning trace is broken when switching from region 1 to region 2, as illustrated in Figures 4.21 and 4.22. Let us analyze the situation of a moving background. A moving background appears as a result of panning or traveling of the camera or of the camera following the foreground object. Let's assume the case where the correct velocity of the moving object is captured in region 2 and not in region 1. The captured velocity cannot propagate to region 1 instantaneously (since region 1 has already been processed), but only in one of the following scanning iterations. In general, this is not a major problem if the number of regions is moderate, since the convergence process can be delayed by only few frames. The author proposed a technique called the dynamic aspect ratio of regions that reduces this problem [176].

4.5.3 The sliding-L1 approach

To reduce the needed L1 capacity, we propose to decouple the vertical dimension of the L1 memory from the size of the region. This is called the *sliding-L1* approach. According to this method, the width of the L1 remains the same as in the classical region-based approach. However, the height of the supported region increases and is equivalent to the height of the picture while the L1 scratchpad significantly reduces its capacity compared to the case of the classical region-based approach. This is achieved if the L1 scratchpad has the minimal height and *slides down the region*. The method is illustrated in Figure 4.23.

The sliding-L1 offers a few benefits compared to classical region-based and stripe-based approaches. One of the most important benefits of the sliding-L1 approach is a significant reduction in the capacity of the L1 scratchpad, compared to both the stripe-based and the region-based approaches. Namely, the height of the L1 scratchpad is equal to $L0_Y + 1$ blocks while its width is typically a few times smaller than the width of the picture. In addition, the number of regions in the vertical domain is just one, which

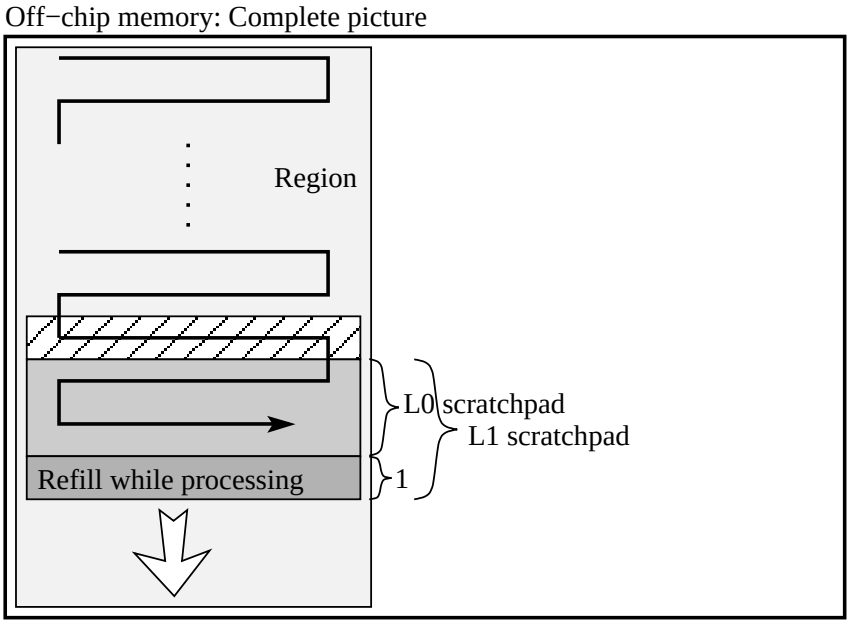


Figure 4.23: The concept of a sliding-L1 scratchpad. The region is much bigger compared to the region-based approach while the capacity of the L1 scratchpad is significantly reduced.

reduces the overall number of regions per picture. Since the number of regions is reduced, there is less interruption in the scanning order, which could potentially increase the quality and also reduce the software complexity.

Multiple scans within the sliding-L1 are possible under the price of area increment. Since the width of the sliding-L1 scratchpad is smaller than the stripe-based L1, its increase in height would result in a smaller cost increase compared to the stripe-based L1 scratchpad. So, the sliding-L1 approach offers a tradeoff between the cost of the L1 scratchpad and quality improvements enabled by the multiple scans.

4.5.4 Bandwidth analysis: Region-based approach

As we have mentioned in Subsection 4.5.2, the region-based approach is the most generic one. In this subsection, we analyze the consequences for the bandwidth requirements. Based on the numbers that come out of this analysis, the numbers for the stripe-based L1 and the sliding-L1 will be derived as special cases.

One of the characteristics of the ideal L1 scratchpad is minimal bandwidth requirement towards the off-chip memory. In the target application domain, each (block of) pixels has to be processed[3] and, thereby, the minimal bandwidth towards the off-chip

[3]This is not true for all video applications. For example, in video decoding, some of the blocks from the

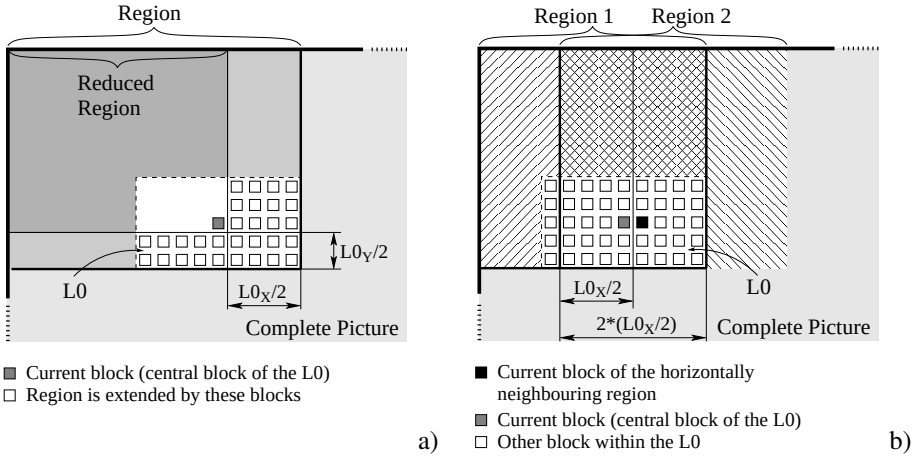


Figure 4.24: The L0 scratchpad (size 9*5 blocks) centered around the currently processed block (gray) and located at the corner of a region. It is not drawn to scale. Picture a) illustrates that the distance between a current block and the edge of the region is $\lfloor L0_X/2 \rfloor$ blocks horizontally and $\lfloor L0_Y/2 \rfloor$ blocks vertically. Picture b) illustrates that the width of the vertical overlapping stripe between two regions is equal to $2\lfloor L0_X/2 rl floor$. Blocks located within this stripe are fetched twice from the off-chip memory.

memory means that each pixel within the picture is accessed exactly once. The region-based approach is characterized by slightly higher bandwidth requirements compared to this minimal one. The difference is due to boundary effects.

4.5.4.1 Traffic between the L2 and L1 scratchpad

When the L0 scratchpad is located at the border of a region, some of its blocks of pixels lie outside the region. Figure 4.24a illustrates that scenario. If the L1 scratchpad holds the area denoted as the reduced region, the blocks of pixels illustrated as white squares are not accessible and the current block cannot be processed. If the L1 scratchpad holds the area denoted as the region, these blocks are accessible. Thereby, the horizontal distance between the lastly processed block of the region and the edge block within the region has to be $\lfloor L0_X/2 \rfloor$ blocks. Similarly, the vertical distance is $\lfloor L0_Y/2 \rfloor$ blocks.

In order to determine the number of pixels accessed twice, we focus on the horizontal overlap between regions. Figure 4.24b provides the illustration. In order to process the black pixel pertaining to Region 2, some blocks from the previously processed region (Region 1) need to be accessed as well. These blocks are located in vertical stripes. The width of a vertical stripe is $2\lfloor L0_X/2 \rfloor$ blocks, where integer division is used. Similarly, we can conclude that the height of a horizontal stripe is $2\lfloor L0_Y/2 \rfloor$ pixels. Pixels within

reference picture might not be accessed at all, while some can be accessed multiple times. In spite of that, the proposed memory hierarchy can be beneficial for such applications as well, since it offers access to un-aligned 2D BOI, predictability, etc. However, this exceeds the scope of this thesis and will not be further analyzed.

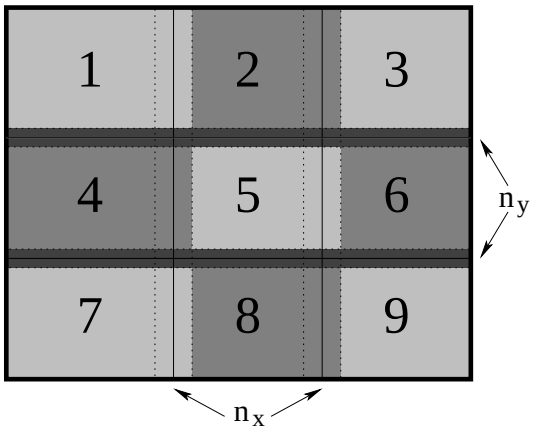


Figure 4.25: The blocks located in the vertically overlapping area (dark gray). These blocks have to be filled to the L1 scratchpad twice. The numbers indicate the processing order.

such stripes are accessed twice within the off-chip memory.

Figure 4.25 illustrates the locations of the pixels that have to be fetched multiple times. With n_x we denote *the number of vertically overlapping stripes* (in x-domain) and with n_y the number of horizontally overlapping ones. Later, we formally derive n_x and n_y. The following equations define the total bandwidth overhead (compared to the minimal bandwidth of 1 access per pixel) towards the off-chip memory. As before, to derive the bandwidth overhead, we compare the number of accesses.

$$\begin{aligned}
&\frac{NA_{REG-L2-L1}}{NA_{MIN-L2-L1}} = \\
&\qquad \text{Vertical stripes in Fig. 4.25} \quad \text{Horizontal stripes in Fig. 4.25} \\
&= \frac{L2_X L2_Y + \overbrace{L2_Y n_x 2\lfloor L0_X/2 \rfloor} + \overbrace{L2_X n_y 2\lfloor L0_Y/2 \rfloor}}{L2_X L2_Y} \\
&\approx \frac{L2_X L2_Y + L2_Y n_x L0_X + L2_X n_y L0_Y}{L2_X L2_Y} \\
&= 1 + n_x \frac{L0_X}{L2_X} + n_y \frac{L0_Y}{L2_Y}
\end{aligned} \tag{4.15}$$

where $L2_X$ and $L2_Y$ denote the width and the height of the picture located in the off-chip memory. This is a generic equation that will be used as the starting point in the

computation of the bandwidth overhead for all three L1 scratchpad approaches. With the help of Figure 4.26, n_y can be derived. When computing the number of regions in the vertical domain, we need to take into account the overlaps between the vertically neighboring regions and the fact that the bottom-most (or top-most, depending on the reference point) region has no overlap. In a similar fashion, we can also derive n_x. Following two equations define n_x and n_y.

$$\begin{aligned} n_x &= \lceil \frac{(L2_X - L1_X)}{(L1_X - 2\lfloor L0_X/2 \rfloor)} \rceil \approx \lceil \frac{(L2_X - L1_X)}{(L1_X - L0_X)} \rceil \\ n_y &= \lceil \frac{(L2_Y - L1_Y)}{(L1_Y - 2\lfloor L0_Y/2 \rfloor)} \rceil \approx \lceil \frac{(L2_Y - L1_Y)}{(L1_Y - L0_Y)} \rceil \end{aligned} \tag{4.16}$$

The approximation in the above two equations can lead to an error when the L0 scratchpad consists of an odd number of blocks[4]. In those cases, an error is equal to one and it is introduced due to approximating the second term of the denominator. Typically, the dimensions of the L1 scratchpad are much larger than one, e.g. 50-100 blocks. So, the error we might introduce is small, e.g. 2-1 %. When the L0 scratchpad consists of an even number of blocks, no error is introduced. We perform this approximation in order to simplify the equations.

The overhead from Equation 4.15 can be reduced if the regions are scanned in a certain order. For example, if all the regions in the picture are processed from left-to-right-top-to-bottom, LRTB (as illustrated in Figure 4.25), the blocks located in the vertical stripes can be reused when switching from two horizontally neighboring regions (e.g. when going from region 2 to region 3 in Figure 4.25). The result is that the second term in Equation 4.15 disappears.

However, when all the regions located at the same horizontal stripe have been processed (regions 1, 2 and 3 in Figure 4.25), the next region to be processed is located in a different stripe (region 4 in Figure 4.25). To process the first few top block-lines of region 4, due to the fact that the top part of the L0 scratchpad partly lies within region 1, a few block-lines from region 1 would be needed. Hence, apart from being present within region 1, these blocks need to be part of region 4. This analysis could be extended for regions 5 and 6 and leads to the conclusion that for the LRTB region scanning strategy, the blocks located in the stripe marked with darkest gray in Figure 4.25 have to be fetched from the off-chip memory twice. Formally, for the case of the LRTB region scanning, the bandwidth overhead is defined with the following equation.

$$\begin{aligned} \frac{NA_{LRTB-REG-L2-L1}}{NA_{MIN-L2-L1}} &= \\ &= 1 + n_y \frac{L0_Y}{L2_Y} \end{aligned} \tag{4.17}$$

In order to estimate the range of this overhead, we start from the assumption that the dimensions of the L0 scratchpad are roughly an order of magnitude lower than those of

[4]The condition that the L0 scratchpad consists of an odd number of blocks is a necessary but not sufficient condition that leads to an error. The eventual wrong value for n_x or n_y is also influenced by the combination of the dimensions of all three factors that contribute in the equation.

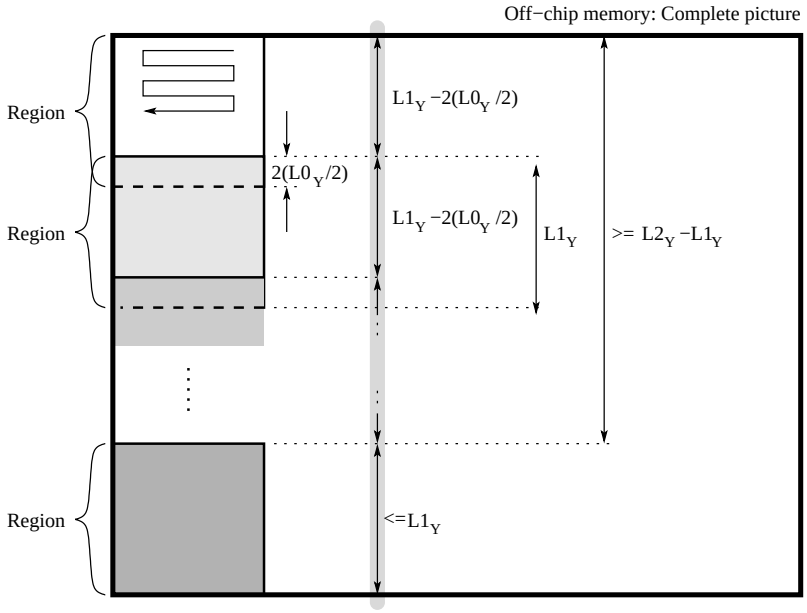


Figure 4.26: Picture clarifies the number of regions used in the vertical domain. The thin vertically oriented shaded area illustrates how is computed the number of regions used in the vertical domain.

the picture stored in the off-chip memory. This brings us to the rough estimate of a few tens of percents.

Note that different region scanning strategies are possible such as meandering or a spiral path leading to lower bandwidth overhead and more complex scanning algorithms. We shall not explore them and in the remaining parts of this thesis, we shall always assume the LRTB region scanning order.

4.5.4.2 Traffic between the L1 and L0 scratchpad

In the region-based approach, the picture is split into $(n_x + 1)(n_y + 1)$ regions (see also Figure 4.25). We assume that those regions are identical. Combining the number of regions with Equation A.1 from Appendix A, we compute the number of accesses for the case of the region-based L1 scratchpad and LRTB scanning order of the L0 scratchpad.

$$\begin{aligned} & NA_{REGION-L1-L0} = \\ & = \quad (n_x + 1)(n_y + 1)L1_Y(L1_X - 1 + L0_X)L0_Y \end{aligned} \tag{4.18}$$

Note that we cannot assume $L1_X \gg L0_X$. By making this assumption, the number of accesses would resume to the typical case of a block-column refill. Doing so, we would remove any influence of the complete refill of the L0 scratchpad. This influence is very important in case of horizontally small regions.

4.5.5 Bandwidth analysis: Sliding-L1 approach

In the sliding-L1 approach, the data reuse from the neighboring regions in Figure 4.25 is already embedded in the approach; regions 1, 4 and 7 are merged into one region and the L1 scratchpad slides downwards in that region. Thereby, the blocks located at the borders of horizontally neighboring regions (indicated by the vertical dotted lines in Figure 4.25) need to be fetched twice. We arrive at a bandwidth overhead number if we omit the last term from Equation 4.15.

$$\begin{aligned} &\frac{NA_{SLIDING-L2-L1}}{NA_{MIN-L2-L1}} = \\ &= \quad 1 + n_x \frac{L0_X}{L2_X} \end{aligned} \tag{4.19}$$

This bandwidth overhead is comparable with the region-based approach. To quantify the traffic between the L1 and L0 scratchpads, we note that the picture is split into $(n_x + 1)$ regions. Thereby, the number of accesses within the L1 scratchpad caused by the L0 scratchpad refill is defined with the following equations.

$$\begin{aligned} &NA_{SLIDING-L1-L0} = \\ &= \quad (n_x + 1)L1_Y(L1_X - 1 + L0_X)L0_Y \\ &= \quad (n_x + 1)L2_Y(L1_X - 1 + L0_X)L0_Y \end{aligned} \tag{4.20}$$

In this case, $L1_Y = L2_Y$, since we consider the region dimensions and not the scratchpad dimensions.

4.5.6 Bandwidth analysis: Stripe-based approach

If the width of the L1 scratchpad is extended to the limits of the picture, the sliding-L1 approach becomes identical to the stripe-based approach. Formally, if we use the property $L1_X = L2_X$ in Equation 4.16, n_x becomes zero. Replacing n_x with zero in Equation 4.19, we arrive at a ratio of one, meaning that there is no bandwidth overhead. Thereby, the bandwidth requirement of the stripe-based approach is theoretically minimal.

$$\frac{NA_{STRIPE-L2-L1}}{NA_{MIN-L2-L1}} = 1 \tag{4.21}$$

In order to compute the number of accesses of the L0 scratchpad to the L1 scratchpad, we note that the stripe-based approach actually utilizes only one region. The dimensions of that region are equal to the dimensions of the picture. This approach results in a minimal number of accesses since it uses one region only. Replacing $L1_X$ with $L2_X$ and $L1_Y$ with $L2_Y$ ($L2_Y$ is the vertical dimension of the region) in Equation 4.18, we compute the number of accesses for the case of the stripe-based L1 scratchpad.

$$\begin{aligned} &NA_{STRIPE-L1-L0} = \\ &= \quad L2_Y(L2_X - 1 + L0_X)L0_Y \end{aligned} \tag{4.22}$$

To compute the ratio between the bandwidth requirements of the region-based and the stripe-based approach, we use Equations 4.18 and 4.22.

$$\frac{NA_{REGION-L1-L0}}{NA_{STRIPE-L1-L0}} = \\ = (n_x+1)(n_y+1)\frac{L1_Y(L1_X-1+L0_X)}{L2_Y(L2_X-1+L0_X)} \tag{4.23}$$

In a similar way, the ratio between the bandwidth of the sliding-L1 and the stripe-based L1 is computed. Equations 4.20 and 4.22 are used.

$$\frac{NA_{SLIDING-L1-L0}}{NA_{STRIPE-L1-L0}} = \\ = (n_x+1)\frac{L1_X-1+L0_X}{L2_X-1+L0_X} \tag{4.24}$$

4.5.7 L1 bandwidth and capacity comparison

This subsection compares the three approaches in terms of bandwidth and capacity of the L1 scratchpad. We compare the region-based, the sliding-L1 and the stripe-based L1 scratchpads. The cost (in terms of number of stored pixel-blocks), the bandwidth overhead towards the off-chip memory and the bandwidth from the L1 scratchpad towards the L0 scratchpad are summarized in Table 4.1 as a function of the parameters L2, L1 and L0. For the comparison, the expressions in Table 4.1 are rewritten as a function of the parameters n_x and n_y because the conclusions depend on the ratios L1/L2 and L0/L2 and not on the absolute values of L2, L1 and L0. We assume that all the regions have the same dimensions or equivalently, that n_x and n_y are integers, without the usage of the ceiling function in Equation 4.16. For the L0 scratchpad, we use two scenarios. According to the first scenario, the L0 scratchpad is relatively large where $L0_X = L2_X/10; L0_Y = L2_Y/10$, and according to the second one, the L0 scratchpad is four times smaller, $L0_X = L2_X/20; L0_Y = L2_Y/20$. Let us start with the most general case, i.e. the region-based L1 approach. Rewriting Equation 4.16 leads to:

$$L1_X = \frac{10+n_x}{10(n_x+1)}L2_X \qquad L1_Y = \frac{10+n_y}{10(n_y+1)}L2_Y \tag{4.25}$$

for the first scenario, and,

$$L1_X = \frac{20+n_x}{20(n_x+1)}L2_X \qquad L1_Y = \frac{20+n_y}{20(n_y+1)}L2_Y \tag{4.26}$$

for the second scenario. The derivation of these equations is available in Appendix B. In case of the sliding-L1 and the stripe-based L1 scratchpads, we have to rewrite $L1_Y = L0_Y + 1$. If we assume that one block-line is typically 1 % of $L2_Y$[5], we arrive at $L1_Y =$

[5]Let's assume that one block-line consists of 8 pixel-lines. For SD resolution (720*576), there are 576/8=72 block-lines and thereby, one block-line is 1/72*100=1.38% of $L2_Y$. For HD resolution (1920*1080), there are 1080/8=135 block-lines and thereby, one block-line is 1/135*100=0.74% of $L2_Y$. On the average, it is a bit more than 1% of $L2_Y$.

Table 4.1 Summary of cost and bandwidth requirements for the three approaches (stripe-based, region-based and sliding-L1 scratchpad). The bandwidth is given relative to the minimal bandwidth (caused by the stripe-based L1 scratchpad).

	L1 Mem. Capacity	BW L2 to L1	BW L1 to L0
	[blocks]	[relative to stripe-based L1]	
Stripe	$L2_X(L0_Y+1)$	1	1
Region	$L1_X L1_Y$	$1+n_y\frac{L0_Y}{L2_Y}$	$\frac{(n_x+1)(n_y+1)}{\frac{L1_Y(L1_X-1+L0_X)}{L2_Y(L2_X-1+L0_X)}}$
Sliding-L1	$L1_X(L0_Y+1)$	$1+n_x\frac{L0_X}{L2_X}$	$\frac{(n_x+1)}{\frac{L1_X-1+L0_X}{L2_X-1+L0_X}}$

$L0_Y+1 = 1/10 * L2_Y + 1/100 * L2_Y = 11/100 * L2_Y$ for the first scenario and $L1_Y = L0_Y+1 = 1/20 * L2_Y + 1/100 * L2_Y = 6/100 * L2_Y$ for the second scenario. Based on the above assumptions, we have summarized the capacity and the bandwidth of the three approaches in Table 4.2. Figure 4.27 illustrates the capacity and the bandwidth requirements of the region-based approach as functions of its two degrees of freedom, n_x and n_y. The graph is plotted according to equations in Table 4.2, relative to the stripe-based approach. The left-hand side of the figure shows the case of the larger L0 scratchpad and the right-hand side of the smaller one.

For a better comparison, the results are plotted in a 2D diagram. The region-based approach has one degree of freedom more compared to the other two approaches. To enable graphical comparison between the three approaches in 2D space, one of its degrees of freedom needs to be frozen. We keep the width of the region, and thereby n_y, constant. The height of the region, and thereby n_x, is parametric. This enables to monitor the tradeoff between the region capacity and the bandwidth towards the off-chip memory. In addition, as we shall see in the further text, this will enable the comparison of the region-based and the sliding-L1 approaches on capacity and the bandwidth from the L1 to the L0 scratchpads. As we can see in Figure 4.27, the selected n_x affects the bandwidth between the L1 and the L0 scratchpad. Thereby, we select it as a compromise between the region capacity and the bandwidth between the L1 and the L0 scratchpad. For example, selecting $n_x = 4$ for the case of the larger L0 scratchpad and $n_x = 8$ for the case of the smaller L0 scratchpad might lead to a suitable compromise. This choice is arbitrarily made. In reality, the selection of these parameters is the result of architectural constraints.

For the case of the sliding-L1 approach, its width and thereby n_y is a parameter allowing to monitor the tradeoff between its capacity and the bandwidths towards the off-chip

Table 4.2 Two instances based on Table 4.1. Table a) is given for the case of $L0_X = L2_X/10$ and $L0_Y = L2_Y/10$. Table b) is given for the case of $L0_X = L2_X/20$ and $L0_Y = L2_Y/20$.

	L1 Mem. Capacity	BW L2 to L1	BW L1 to L0
	[relative to stripe-based L1]		
Stripe-L1	1	1	1
Region-L1	$\frac{1}{11}\frac{10+n_x}{n_x+1}\frac{10+n_y}{n_y+1}$	$1+\frac{1}{10}n_y$	$\frac{1}{1090}(19n_x+109)(10+n_y)$
Sliding-L1	$\frac{10+n_x}{10(n_x+1)}$	$1+\frac{1}{10}n_x$	$\frac{1}{109}(19n_x+109)$

a)

	L1 Mem. Capacity	BW L2 to L1	BW L1 to L0
	[relative to stripe-based L1]		
Stripe-L1	1	1	1
Region-L1	$\frac{5}{120}\frac{20+n_x}{n_x+1}\frac{20+n_y}{n_y+1}$	$1+\frac{1}{20}n_y$	$\frac{1}{2080}(9n_x+104)(20+n_y)$
Sliding-L1	$\frac{20+n_x}{20(n_x+1)}$	$1+\frac{1}{20}n_x$	$\frac{1}{104}(9n_x+104)$

b)

memory and the L0 scratchpad. Note that its height is precisely defined as in case of the stripe-based approach. Figure 4.28 graphically presents the comparison between the three approaches. The stripe-based approach is selected as the reference point.

To conclude, the result of the comparison between the three approaches is that the sliding-L1 approach offers the best compromise between the capacity, the bandwidth towards the off-chip memory and the bandwidth towards the L0 scratchpad. Comparing it with the stripe-based approach, for the case of the larger L0 scratchpad, the ideal point might be $n_x = 2$ where the capacity of the L1 scratchpad is 60% lower, bandwidth towards the off-chip memory is 20% higher and bandwidth towards the L0 scratchpad is 35% higher. For the case of the smaller L0 scratchpad, the ideal point might be $n_x = 3$ where the capacity of the L1 scratchpad is 71% lower, bandwidth towards the off-chip memory is 15% higher and bandwidth towards the L0 scratchpad is 26% higher. Let us compare the sliding-L1 with the region-based L1 approach. Making n_y constant in case of the region-based L1, enabled that both sliding-L1 and region-based L1 have the same

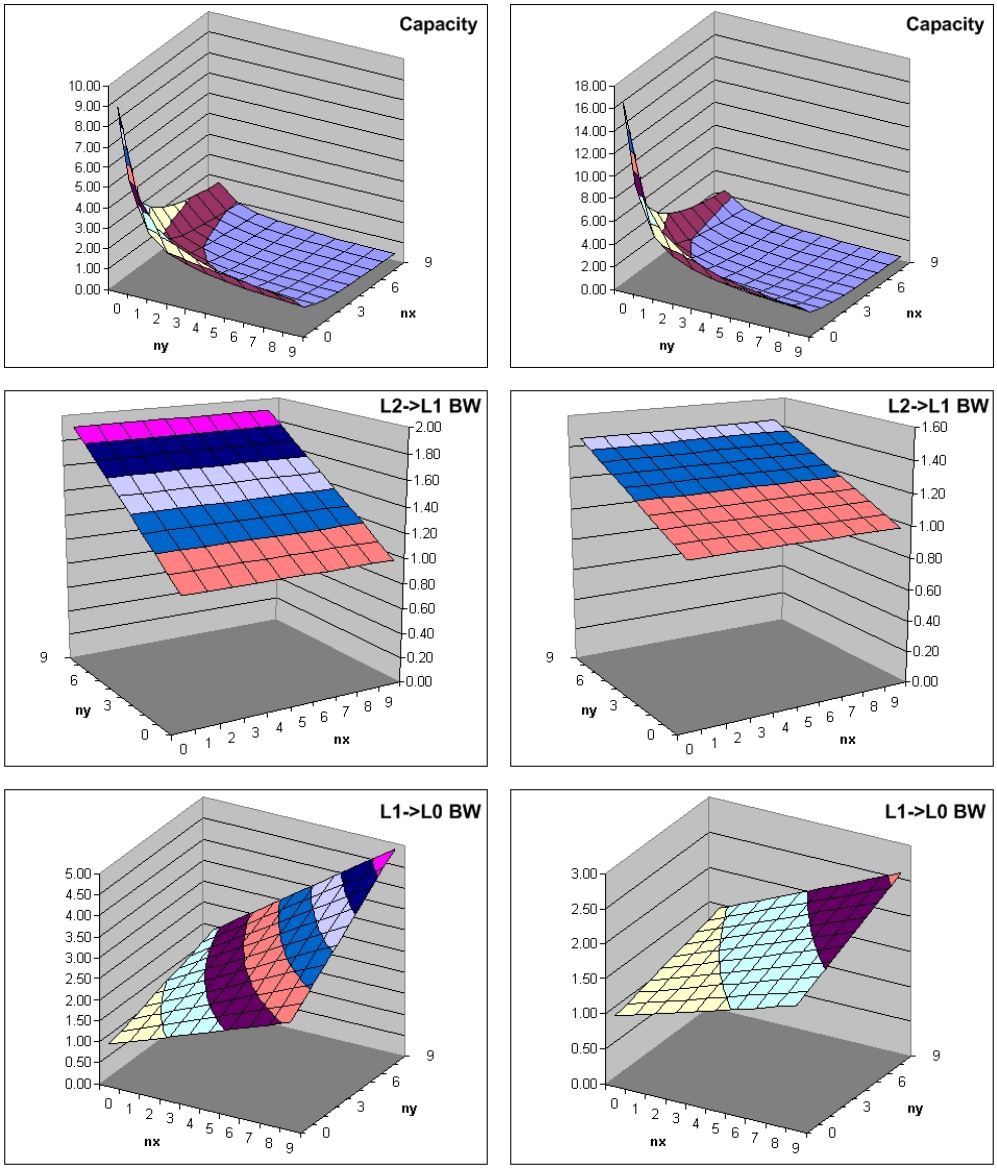


Figure 4.27: Plots of the capacity and the bandwidth requirements of the region-based approach as a function of its two degrees of freedom, n_x and n_y. The left-hand side reflects the setup where $L0_X = L2_X/10$; $L0_Y = L2_Y/10$. The right-hand side reflects the setup where $L0_X = L2_X/20$; $L0_Y = L2_Y/20$. All the graphs are plotted relative to the stripe-based approach.

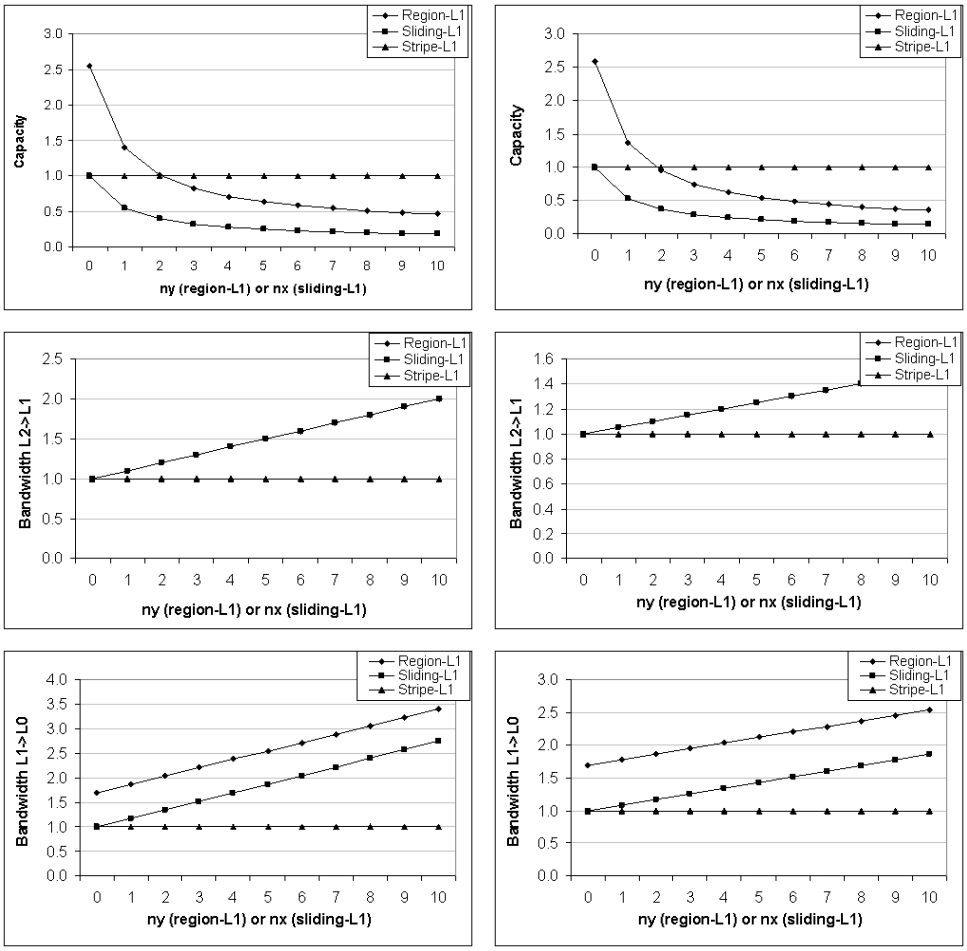


Figure 4.28: The difference in the L1 scratchpad capacity requirements and bandwidth overhead towards the off-chip memory and the L0 scratchpad. For the region-based approach, n_y is the degree of freedom while for the sliding-L1 approach, it is n_x. The capacity and the bandwidth are expressed relative to the stripe-based approach. The left column is given for the setup where $L0_X = L2_X/10; L0_Y = L2_Y/10$ and $n_x = 4$ in case of the region-based approach. The right column is given for the setup where $L0_X = L2_X/20; L0_Y = L2_Y/20$ and $n_x = 8$ in case of the region-based approach.

bandwidth overhead. This enabled comparison of other parameters, the capacity and the bandwidth from the L1 to the L0 scratchpad. For all the cases, the sliding-L1 offers lower capacity, typically two times. Bandwidth towards the L0 scratchpad is lower as well.

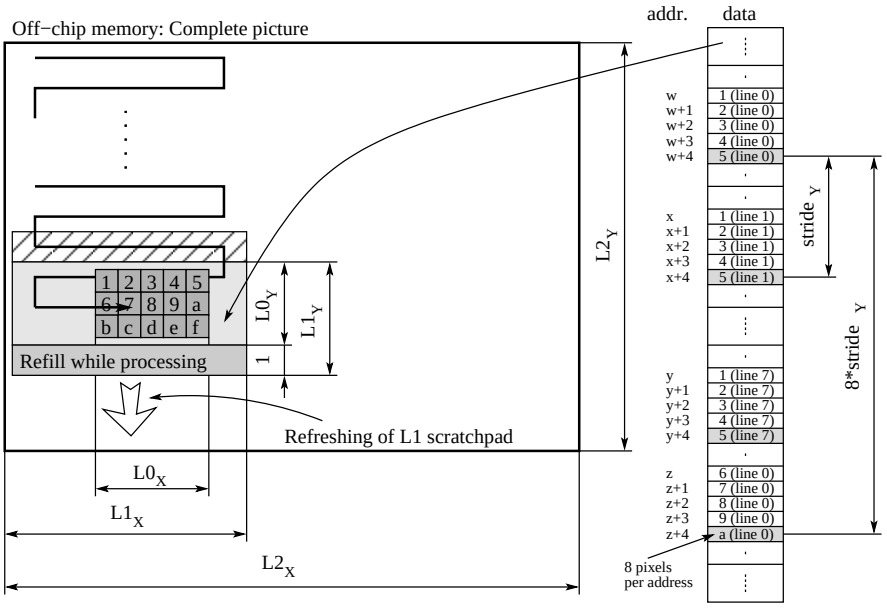


Figure 4.29: The internal organization of (blocks of) pixels within the scratchpad utilizing sliding region. It assumes 8*8 pixels per block and a width of the L1 scratchpad's memory bank of 8 pixels. The picture illustrates the common case when the data is written to the L1 scratchpad in a line-by-line manner. The picture also illustrates which addresses need to be accessed in order to load a block-column from the L1 scratchpad.

4.5.8 Data organization within the L1 scratchpad

From the L1 scratchpad point of view, the data flow from the off-chip memory as well as the data flow to the L0 scratchpad is block-aligned. Thereby, there is no need for an unaligned read or write data access. This enables straightforward addressing, which can be realized in software, without the need for support of any specific hardware. Hence, for the L1 scratchpad, the use of standard memory banks and standard addressing mechanisms (base address plus the offset) is sufficient. The width of the access points of the L1 scratchpad is dictated by the required performance. In a VLIW processor, load/store units are attached to the L0 and L1 scratchpads. The data that is communicated to or from these scratchpads is produced or consumed in the datapath of the processor. The data width conversion can be avoided if the widths of the access points of the L0 and L1 scratchpads are equal. This simplifies the architecture and here we make the choice to make these access points equal.

SoCs are usually based on a system-bus architecture where all the components in the SoC communicate over one or more data buses such as the AMBA AHB multilayer bus [171]. Usually, the width of these buses is just 32 bits. To improve the performance, the preferred way of communication are burst data transfers. As we have mentioned in Sub-

section 4.3.1, the task of performing the data transfers to and from the off-chip memory can be realized through a DMA controller [172, 173]. Apart from communicating with the off-chip memory using burst transfers, the DMA controller also does packing and unpacking data and thereby functions as an interface towards the ”external world”.

The data usually arrives in a line-by-line fashion to the off-chip memory. In the same fashion it is stored. Pixels are grouped horizontally and stored in the off-chip memory, which is most often 32-bit or 64-bit wide. The data is also stored in a line-by-line fashion in the L1 scratchpad. Figure 4.29 illustrates that. The only difference is that the pixels are regrouped (by the DMA) and stored in wider words, to fit the internal architecture of the L1 scratchpad. Loading a block-column, which is a typical case for the L0 scratchpad refill, is straightforward. The typical base and offset addressing mechanism is sufficient to efficiently load one block column. As an example, pseudo code for a block-column load from the L1 scratchpad and store to the L0 scratchpad is provided below. This pseudo code is realized in software.

```
// Reading one L0 block-column
// Block-column is centered around (bx, by)
raddress = (by-L0_Y/2) mod L1_Y ;
raddress = raddress * (8 * strideY) ;
raddress = raddress + bx*8/8 ;
// *8 pixels per block /8 pixels per mem.  loc.  in L1
for (offset = 0 ; offset < 8*strideY ; offset += strideY) {
   load (L1scratchpad, raddress, offset, subblock) ;
   pixel_pad (subblock, padded_subblock) ;
   store (L0scratchpad, padded_subblock) ;
}
```

The software can also be written for different types of accesses, for example a block-row update. Since the sliding-L1 approach only assumes aligned read and write accesses, software-based addressing is sufficient. Different padding schemes can also be programmed. This software realization supports flexibility within the target domain.

4.6 Number of hierarchy levels

The previous sections motivate the need for each level of the memory hierarchy. The L1 scratchpad is needed to reduce the bandwidth requirements towards the off-chip memory to a minimum. Accesses within the L1 scratchpad are aligned, which allows the usage of a generic memory block for L1. Addressing is realized in software. The L0 scratchpad enables accesses to an arbitrarily positioned 2D BOI. Read accesses are un-aligned. To enable this, more than one memory bank is used and additional hardware support is needed.

The question is how many levels of memory hierarchy are needed. Specifically, we compare two memory architectures. One is the memory subsystem that consists of two levels of memory hierarchy (2-level memory subsystem). The second one uses only one level of memory hierarchy (1-level memory subsystem). The 1-level memory subsystem is a result of a merge between the two concepts, the L0 memory architecture as presented in Section 4.4 and the sliding-L1 data organization as presented in Subsection 4.5.3. The 1-level memory subsystem must behave in the same way from the processing element as well as from the off-chip memory point of view. In other words, it must enable un-aligned access to a 2D BOI and maintain the minimal off-chip memory bandwidth.

The 2-level memory subsystem has the disadvantage that the same pixels are present in multiple locations, in different memory locations. Pixels are read from the L1 scratchpad and copied to the L0 scratchpad. Having just one level omits the need for copying pixels from the L1 to L0 scratchpad. We denote it with $L01_M$ (L0 and L1 Merged) scratchpad. The $L01_M$ scratchpad has an internal organization of the L0 scratchpad and the capacity of the L1 scratchpad as defined according to the sliding-L1 approach. The differences between the two architectures are illustrated in Figure 4.30.

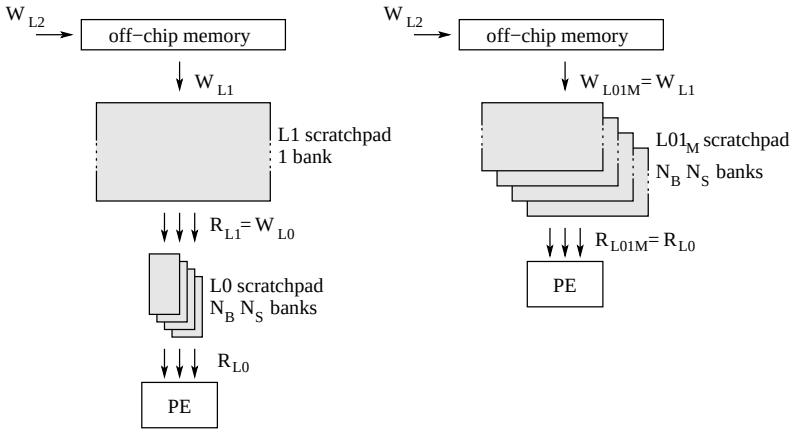


Figure 4.30: On the left-hand side, the 2-level memory subsystem. On the right-hand side, the 1-level memory subsystem that is based on the merged, $L01_M$ scratchpad. The access points are appended with read and write access rates.

Table 4.3 Bandwidth [MByte/s] and memory capacity requirements [Kbytes] for five analyzed approaches. The data are given assuming HDTV interlaced material (1920*1080i @60Hz) and one referenced video field.

	Proposed memory subsystem	
	2-level	1-level
L1 dim. [blocks]	88*8	/
L1 cap. [blocks]	704	/
L1 cap. [KB]	22	/
L0/L01$_M$ dim. [blocks]	13*7	88*8
L0/L01$_M$ cap.[blocks]	91	704
L0/L01$_M$ cap. [KB]	2.8	22
Off-chip mem. BW [MB/s]	65.3	65.3
Off-chip mem. BW overhead [%]	10	10
L1 $\rightarrow$ L0/L01$_M$ BW [MB/s]	461.7	/
L1 $\rightarrow$ L0/L01$_M$ BW overhead [%]	10.7	/
L0/L01$_M$ $\rightarrow$ PE BW [MB/s]	415.3	415.3

Let us illustrate the differences in capacity and similarities in bandwidth requirements between the 2-level and 1-level memory subsystems using the case study of the high-quality motion-compensated de-interlacing application applied for input interlaced HDTV material (1920*1080i @60Hz) [20]. This application references multiple frames. We analyze the access patterns related to only one reference picture because other references are similar.

The size of the L0 scratchpad is 13*7 blocks of 8*4 pixels. The vertical dimension of a block is 4 pixels and not 8 pixels since the input material is interlaced. Consecutively, the size of the frame is 240*135 blocks. It is assumed that the scanning trace of the L0 scratchpad is horizontal meandering and for each processed block, seven motion vector candidates are evaluated. A pixel is encoded with 8 bits. Table 4.3 illustrates the bandwidth and area requirements of the two approaches. As expected, the 1-level memory subsystem requires less capacity and both memory subsystems require the same bandwidth from the off-chip memory and the processing element.

The following four subsections compare the 1-level and 2-level memory subsystems on the following important aspects, cost (area), cost (power), impact on the software, impact on the algorithm (WOI size).

Table 4.4 Differences in area between the 1-level and 2-level memory subsystems for six use-cases. Organization of the L0 and $L01_M$ scratchpads uses the following setup: $N_B = 2, N_P = 8$.

$N_B = 2, N_P = 8$	16-pixel BOI, $N_S = 2$			32-pixel BOI, $N_S = 4$		
WOI [blocks]	3*3	13*7	21*15	3*3	13*7	21*15
L0 scratchpad [blocks]	9	91	315	9	91	315
L1 scratchpad [blocks]	76	552	1504	76	552	1504
L0 scratchpad [mm^2]	0.160	0.216	0.464	0.320	0.368	0.520
L1 scratchpad [mm^2]	0.112	0.524	1.235	0.169	0.567	1.235
Total 2-level [mm^2]	0.272	0.74	1.699	0.489	0.935	1.755
$L01_M$ scratchpad [blocks]	76	552	1504	76	552	1504
$L01_M$ scratchpad [mm^2]	0.208	0.696	1.516	0.360	0.872	1.664
Total 1-level [mm^2]	0.208	0.696	1.516	0.360	0.872	1.664
1-level is smaller by [%]	31	6	12	36	7	5

The influence on area and power will be analyzed through two groups of use cases that are chosen in accordance with our application domain. The first group is based on a 16-pixel BOI (8*2) and the second one on a 32-pixel BOI (8*4). For each group, we analyze three dimensions of the WOI: small (3*3 blocks), medium (13*7 blocks) and large (21*15 blocks). A block contains 8*8 pixels. Small WOI is used by pixel-filtering algorithms while large and medium WOIs by motion estimation/compensation. The internal organization in terms of number of banks and sets is identical for the L0 and the $L01_M$ scratchpad: $N_P = 8$, $N_B = 2$ and $N_S = 2$ (8*2 pixel BOI), and $N_P = 8$, $N_B = 2$ and $N_S = 4$ (8*4 pixel BOI). In all cases, the dimensions and capacity of the L1 and the $L01_M$ scratchpad are defined according to the sliding-L1 approach.

4.6.1 Comparison: Cost (Area)

From the previous discussion, it is clear that the 1-level memory subsystem requires less memory capacity than the 2-level. They differ exactly by the capacity of the L0 scratchpad. However, the difference in area is influenced by the internal architecture. To explore those differences we analyze the mentioned use cases. The results are shown in Table 4.4.

The results show that in all cases, a 1-level memory subsystem offers lower area. The gains are the biggest in cases of small WOI. The reason is that in case of the 2-level memory subsystem, even small capacity of the L0 scratchpad is distributed in number of banks, 4 in case of the 16-pixel BOI and 8 in case of the 32-pixel BOI. As we have seen in Subsection 4.3.3, banks with small capacities are inefficient. The $L01_M$ scratchpad has the same number of banks with larger capacities, which leads to higher efficiency. As the size of the WOI grows, the efficiency of the L0 scratchpad increases and the difference in area between the two memory subsystems becomes smaller.

4.6.2 Comparison: Cost (Power)

Power dissipation consists of two components, active power (AP) and standby leakage power (LP). For the memory technology used here, leakage power is significantly lower than active power. Note that this is not the case for lower technology nodes, such as 45nm. In our case, assuming that memory is clocked at 200 MHz, leakage power is lower than 0.3 % of the active power. Therefore, we focus on active power only.

Let us analyze similarities and differences between the two approaches. Looking at the PE side, the read rates of the L0 and the $\mathrm{L01_M}$ scratchpads are the same. The situation is different at the off-chip memory side. The frequencies with which pixels are written into the 2-level and 1-level memory subsystem are the same. However, the $\mathrm{L01_M}$ scratchpad is based on a number of banks, while the L1 scratchpad is typically based on just one bank. All of the $\mathrm{L01_M}$ scratchpad's banks are active during read (to enable un-aligned access), but not all are active during write. For example, in case of the 2D 8*2 pixel BOI where $N_B = 2, N_P = 8$, only 2 out of 4 banks are active during write. This means that 50% of the banks are active and thereby halves the effective frequency of the write accesses. Finally, in case of the 2-level memory subsystem, additional power is caused by copying the pixels from the L1 to the L0 scratchpad.

The power dissipation will be analyzed using the equations derived in this chapter, in particular, Equations 4.2, 4.3, 4.14, 4.16, 4.19, 4.20. In addition, a simplified power model, illustrated in Figure 4.31, will be used. This model is piece-wise linear, consisting of only two segments. The left segment we call the flat and the right one the inclined segment. We use three widths of scratchpads here and, as before, assume that pixels are encoded with 8 bits. Thereby, the banks of the L0 and $\mathrm{L01_M}$ scratchpads are 64 bits wide. In case of 16-pixel BOI, the L1 scratchpad is 128 bits wide, and in case of 32-pixel BOI, it is 256 bits wide. We analyze two use cases, the small and the medium/large WOI.

Small WOI: The number of words for all the scratchpads (L0, L1 and $\mathrm{L01_M}$) is relatively small. The L1 scratchpad requires the largest number of words, which in case of the 16-pixel BOI is 304. Thereby, all of them are located in the flat segment of our power model. Because of this, the power dissipations caused by the reads from the L0 and $\mathrm{L01_M}$ scratchpads are almost the same. In the following paragraph, we compare the

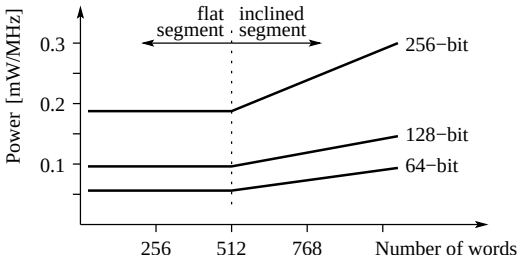


Figure 4.31: The piece-wise linear power model that is used for power dissipation analysis.

power caused by the writes to the same scratchpads.

Since the 1-level memory subsystem keeps the off-chip memory bandwidth requirement minimal, the assumption is that each pixel is approximately written once to the $\mathrm{L01_M}$ scratchpad. However, in case of the 2-level memory subsystem, each pixel is written to the L0 scratchpad approximately $L0_Y$ times. Even in case of a small WOI, $L0_Y$ is greater than one[6]. On top of that, the power dissipation of the L1 scratchpad has to be included. Our expectation is that for the low number of accesses (e.g. low number of motion vectors or low number of pixels in the filter support), the 1-level memory subsystem will consume significantly less power than the 2-level. As this number increases, the difference will be smaller and smaller. At some point, the power caused by the PE reads will prevail and the fact that the $\mathrm{L01_M}$ scratchpad has higher power dissipation than the L0 scratchpad (because of higher capacity) will play a dominant role. Only then, the 2-level memory subsystem will have lower overall power dissipation.

Medium/Large WOI: The number of words for the $\mathrm{L01_M}$ and L1 scratchpads is large. The smallest is in case of the medium WOI and 32-pixel BOI ($\mathrm{L01_M}$ scratchpad requires 552 words) and the largest is in case of the large WOI and 16-pixel BOI (L1 scratchpad requires more than 6000 words). Thereby, both are in the inclined segment of our power model. On the other hand, the number of words for the L0 scratchpad is located in the vicinity of the knee of the curve. It ranges from 91 words in case of the medium WOI and 32-pixel BOI to 630 words in case of the large WOI and 16-pixel BOI.

The power dissipation caused by the reads from the lowest level of the memory hierarchy is significantly larger in case of the $\mathrm{L01_M}$ scratchpad (especially when the PE issues a large number of reads). On the other hand, the power caused by the writes into the L0 scratchpad is significant, since the WOI is medium to large implying larger values of $L0_Y$. In addition, a large L1 scratchpad also causes significant power dissipation. Based on the analysis so far, we conclude the following: If PE issues a low number of reads, the most dominant component is the L1 scratchpad and power caused by the writes to the L0 scratchpad. Note that, this component of the power dissipation remains constant for any number of the PE accesses. If the number of accesses is low, the dominant power component is caused by the reads from the lowest level of the hierarchy. In such cases, the 1-level memory subsystem is expected to have lower power dissipation. As the number of accesses increases, the power caused by the reads increases as well. At some point, this power becomes dominant and the 2-level memory subsystem is better in terms of power.

Use cases: Here we refer to two groups of use cases defined before. The results for HDTV standard (1920*1080p @ 60 fps) are plotted in Figure 4.32. The number of block read accesses from the lowest level of the memory hierarchy ranges from 1 to 64 per block of 8*8 pixels. In case of pixel filtering, the number of accesses is equal to the number of pixels in the pixel support. In most cases, it is lower than 16. In case of motion estimation, if full-pel accurate vectors are used, the number of accesses is equal to number of motion vectors, which is typically 6-8. If the vectors are sub-pel accurate

[6]Theoretically, $L0_Y$ can be equal to one, e.g. in cases when pixel interpolation does not require accesses to any vertically neighboring pixel-lines.

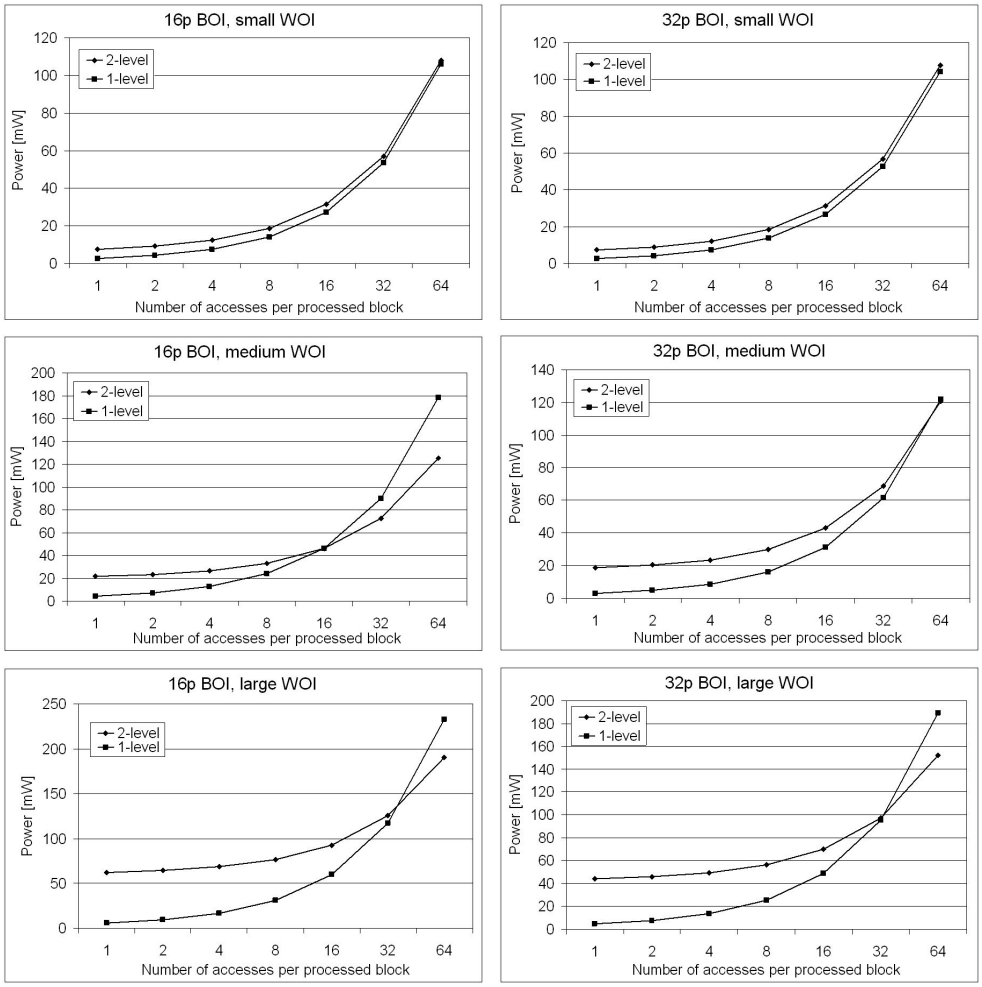


Figure 4.32: The difference in power dissipation between the 2-level and 1-level memory subsystem. The selected configuration of the lowest level of the memory hierarchy is $N_B = 2, N_P = 8$. When the number of accesses is moderate, the 1-level memory subsystem offers far lower power dissipation. In some cases, the difference is more than 10 times.

and the bi-linear interpolation has to be used, the number of accesses is four times larger, 24-32. The number of accesses can significantly be reduced if the linear interpolation is embedded in the memory subsystem (see Subsection 4.4.6). In such a case, one additional pixel-line has to be fetched per block of 8*8 pixels. This means that from the original 6-8 accesses, the number of accesses becomes 9/8 times higher, or approximately 7-9. In case of motion compensation, only one access is needed. Thereby, for most of cases, the 1-level memory subsystem offers lower power dissipation.

4.6.3 Comparison: Impact on the software

In a scratchpad-based approach, a programmer is responsible for placing the memory operations within the memory hierarchy. This task begins with address calculation. This address is needed to read the pixels from the L1 scratchpad. These pixels are conditionally padded (needed in the vicinity of picture borders) and written to the L0 scratchpad. Many of the mentioned actions are not required for the case of the 1-level memory subsystem. The differences are illustrated in Figure 4.33.

The differences between the two approaches are reflected in the increased performance, the reduced size of the program memory and additional gains in power. Let us briefly explain those gains from the perspective of the 1-level memory subsystem. Since a number of instructions from Figure 4.33 are not executed, clock cycles are spared, which causes a performance increase. Without those instructions, the code size is reduced, which reflects in area and power. The overall power dissipation is further reduced since the datapath does not execute those instructions. Lastly, different scanning patterns are implemented in an easier way. At the left hand side of Figure 4.33, the LRTB scanning order is implemented. In order to implement the meandering, one of the changes is that the filling of the complete L0 scratchpad has to be replaced by the block-row filling. In case of the 1-level memory subsystem, the changes are much smaller. They only require a change of the read coordinates provided to the $L01_M$ scratchpad.

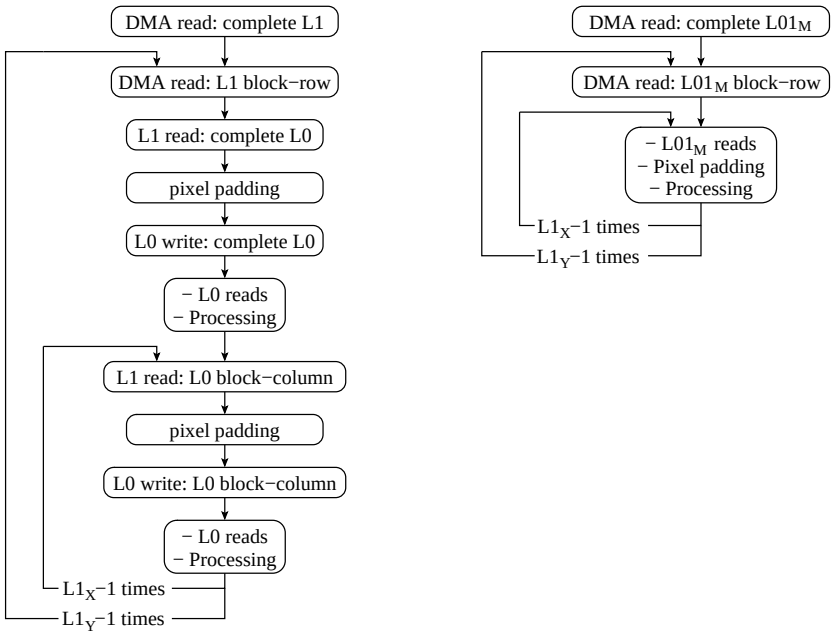


Figure 4.33: On the left-hand side, the picture illustrates the simplified snapshot of the software in case of the 2-level memory subsystem. On the right-hand side, in case of the 1-level.

4.6.4 Comparison: Impact on the algorithm

A final advantage of the 1-level memory subsystem that we mention here is the increased size of the supported WOI. This is especially important for motion-compensated applications. The dimensions of the $L01_M$ scratchpad are equal to the dimensions of the L1 scratchpad and thereby the covered WOI is bigger than in case of the L0 scratchpad. Compared to the 2-level memory subsystem, there are no differences in the WOI height. Let us compare the differences in the WOI width, W_{WOI}, between the two approaches. We analyze the situation where the overlap between the sliding regions is the same for both approaches (see equations 4.15, 4.16 and Figure 4.24). In case of the 2-level memory subsystem, W_{WOI} corresponds to the L0 scratchpad. In case of the 1-level memory subsystem, W_{WOI} corresponds to the $L01_M$ scratchpad and is a few times wider than in case of the 2-level memory subsystem. However, motion vectors are asymmetrical as illustrated in Figure 4.34. The maximal horizontal vector length is influenced by the position of the currently processed block. In all cases, the minimal horizontal vector length is greater than or equal to the one supported by the 2-level memory subsystem.

The 1-level memory subsystem offers a WOI that is horizontally a few times bigger. This is important for applications that use motion estimation and compensation as it increases the range of the motion vectors by a number of times. In post processing applications, this is of limited importance since the WOI is asymmetrical and not all velocities (directions) can be tracked. In case of the video compression and decompression this maybe more important. These applications typically do not aim to true motion and thereby do not suffer from the asymmetrical WOI. They only look at the minimal block difference and enlarged search space improves that process.

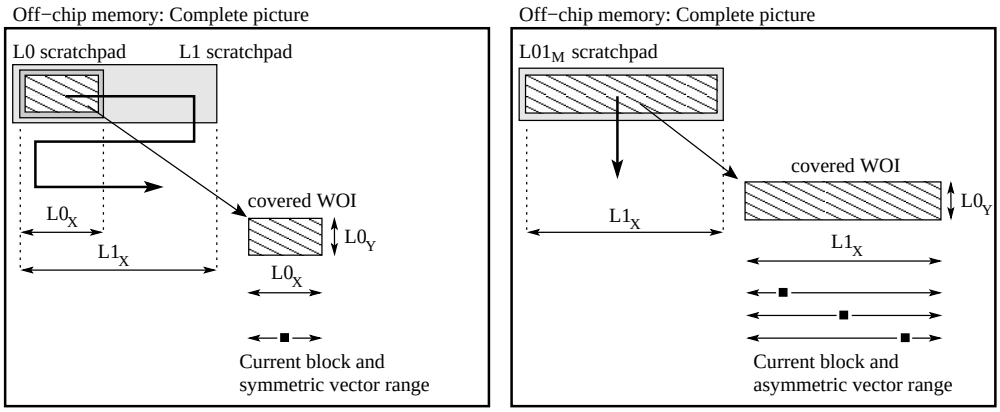


Figure 4.34: The difference in the supported WOI size. On the left-hand side, the 2-level memory subsystem is illustrated and on the right-hand side, the 1-level. The 1-level memory subsystem supports a much larger WOI.

4.7 Benchmark of the memory subsystems

This section benchmarks our memory subsystem against prior work. The benchmark is performed using the proposed six criteria. The support of each criterion is marked with one of the following possible answers, *yes*, *partly* or *no*. Table 4.6 given at the end of this Subsection summarizes the benchmark results.

4.7.1 Cache-based memory subsystems

Two techniques are typically used to optimize the cache performance for streaming applications, data partitioning and prefetching. The data replacing strategies such as the "least recently used blocks" do not guarantee that all the data needed for computation of the current block are available in the cache. The application programmer has typically no control what is overwritten in the cache. This is one of the causes for the cache misses that increase the off-chip memory bandwidth and stall the processing element.

Here we assume that the cache contains sufficient memory capacity to keep the off-chip memory bandwidth low, in spite of the cache misses. The first criterion is thus supported. The cache-based approach is not predictable although it is typically possible to define the lower bound of the estimated performance. Thereby, we rank the predictability with partly. In general, caches are not capable to deliver un-aligned blocks of data. In some cases, like in the Trimedia 3270 [147], un-aligned access to a group of four pixels is supported. However, the cache misses cause stalls and thereby, the support to the High PE Bandwidth criterion is marked as partly. Cache architectures are scalable and flexible. The need to support all applications results in a cache size increase. This increase leads to off-chip memory bandwidth reduction but also to inefficient implementation. Here, we assume that some of the techniques for reducing the miss-rate mentioned in Subsection 4.2.1 are applied such that the cache size is not dramatically increased. Taking into account the usage of these techniques, the efficiency of pixel storage and pixel access is improved but not to a level of the sliding-L1 approach, which guarantees the off-chip memory bandwidth for selected capacity. Another drawback is the area and power overhead as a consequence of cache tags. We conclude that the cache-based architectures partly satisfy the efficiency criterion.

4.7.2 Domain-specific memory subsystems

The publications that target domain-specific memory subsystems can be split into two major groups. The first group minimizes the off-chip memory bandwidth and the second one focuses on memory architectures that enable read accesses to un-aligned BOIs. To the best of our knowledge, there were no attempts to jointly address this problem in the context of custom-based memory subsystems for video processing. Firstly, we briefly summarize the methods for off-chip memory bandwidth reduction and then, in greater detail, the architectures that enable un-aligned pixel accesses.

Most of the authors from prior work solve the problem of high off-chip memory band-

Table 4.5 Differences in area between the prior work [167–170] and our proposal. Both are based on a 1-level memory subsystem and have identical capacity. Organization of the $L01_M$ scratchpads uses the following setup: $N_B = 2, N_P = 8$.

$N_B = 2, N_P = 8$	16-pixel BOI			32-pixel BOI		
Total capacity [blocks]	19*4	69*8	94*16	19*4	69*8	94*16
1-pixel bank [mm^2]	0.022	0.068	0.13	0.019	0.078	0.078
Prior work [mm^2]	0.352	1.088	2.080	0.608	1.248	2.496
8-pixel bank [mm^2]	0.052	0.174	0.379	0.045	0.109	0.208
Our proposal [mm^2]	0.208	0.696	1.516	0.360	0.872	1.664
Prior work is larger by [%]	69	56	37	68	43	50

width by using a two-level memory hierarchy [125, 144, 164, 165]. In a typical case, the higher level of the memory hierarchy contains the complete stripe that extends across the complete picture. Thereby, each pixel is accessed only once, which results in the minimal bandwidth. The cost is high since the required memory capacity is large, especially for HDTV picture resolution. The work of Tuan et al. [165] offers four points for tradeoff between the bandwidth increase and capacity reduction. However, the bandwidth increases sharply with area reduction. The cited publications do not discuss the memory organizations that enable un-aligned pixel access. The exception is [125], which is based on inefficient pixel- and line-delays. We shall not benchmark the mentioned proposals as they do not propose memory organizations. Instead, we focus to memory subsystems that are based on the single-pixel wide memory banks, which enable un-aligned accesses.

A substantial effort has been made to propose an architecture of a memory subsystem which enables un-aligned pixel access [122, 162, 167–170]. All the cited approaches are based on single-pixel wide memory banks. The advantages are that many dimensions and shapes of a BOI can be addressed, including the diagonal patterns. Thereby, more attention will be devoted to these approaches. Most emphasis will be on efficiency and cost (power, area). We focus on the following architectures, [167–170] since they require fewer banks than the scratchpads reported in [122, 162].

These architectures are proposed to enable un-aligned access pattern. The authors made no effort to minimize the off-chip memory bandwidth. Thereby, we do not rank the support to the first criterion. These architectures are predictable; criterion two is therefore supported. The architectures offer high PE bandwidth allowing accesses to an un-aligned (subsampled) 2D BOI. The PEs are constantly fed with useful pixel data. Thereby, the third criterion is supported. To score these architectures on the flexibility criterion we note the following. There are no limitations in terms of the scanning order as the (groups of) pixels can be written in any order. The utilized size of the WOI can be different than

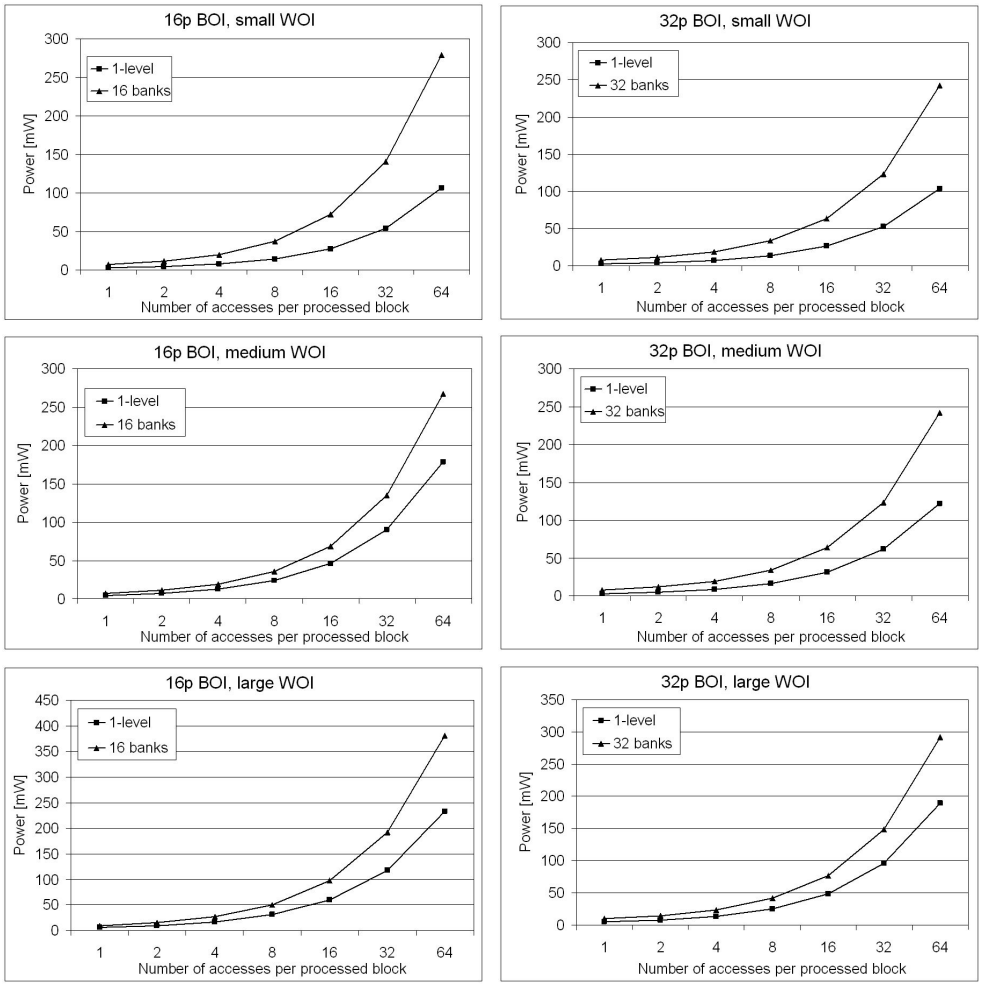


Figure 4.35: The difference in power dissipation between our proposal (the 1-level memory subsystem) and the memory subsystem based on single-pixel wide memory banks. In all cases, the power dissipation of our 1-level memory subsystems is lower.

the physically available, which offers an opportunity for bandwidth reduction. Different aspect ratios of the 2D BOI are also supported. There are no limits in terms of number of reads of a BOI. In summary, we conclude that the flexibility criterion is satisfied.

To evaluate the efficiency criterion, we note that due to the usage of many narrow banks, this architecture does not offer efficient (area and power) pixel storage. The area comparison is provided in Table 4.5 and the differences in power dissipation between this work and our proposal are illustrated in Figure 4.35. Pixels are written individually to the memory, which is not cost-effective. Namely, to write a block of n pixels to a memory

Table 4.6 Benchmarking of the proposed memory subsystem with existing solutions. Six proposed criteria are used.

	General-purpose	Application-domain specific	
	Generic Cache	[122, 162, 167, 168, 170]	Our work
Minimal off-chip mem. bandwidth	yes	n.a.	yes (typ. 1.2 acc/pix)
Predictability	partly	yes	yes
High PE bandwidth	partly	yes	yes
Flexibility	yes	yes	yes
Efficiency	partly	no	yes
Scalability	yes	yes	yes

that consists of n banks, all n banks are activated. In this case, the memory architecture does not profit from the fact that the write access is aligned and that the written pixels are spatial neighbors. Situation during reading is similar but not the same since the accesses are un-aligned. All the banks are again activated. Thereby, we conclude that the efficiency criterion is not supported.

These approaches are scalable since increase of the bank capacity causes reasonable increase of area and power. Namely, as we have seen in Subsection 4.3.3, the slope of the increase curve is always less than or equal to 45 degrees (doubling the capacity in worst case causes doubling the area). Power increases much slower than area. If the 2D BOI doubles its size (from 16 to 32 to 64 pixels), the number of banks increases linearly (from 16 to 32 to 64 banks). Thereby, the scalability criterion is supported.

4.7.3 Our work

1. Minimal off-chip memory bandwidth: The bandwidth to the off-chip memory is reduced close to one access per pixel. The overhead is in the order of ten percents and it is user-controlled.

2. Predictability: Performance of the proposed memory subsystem is known at compile time, i.e. the memory subsystem is predictable.

3. High PE bandwidth: Memory subsystem is designed such that it delivers un-aligned 2D BOI per access, i.e. useful pixels. The PE is not stalled. We conclude that our memory subsystem satisfies the third criterion.

4. Flexibility: The Sliding-L1 approach tolerates the change of the picture width and height. The architecture does not impose any restriction about the scanning order,

number of reads of a BOI or the size of the WOI. Aspect ratio of the un-aligned 2D BOI is programmable. In summary, the algorithmic parameters are controlled in software and thereby our architecture satisfies this criterion.

5. Efficiency: The pixels are stored into the memory subsystem using aligned access triggering significantly less address locations compared to the number of stored pixels. This is power efficient. The system is area effective since it utilizes wide banks, each bank holding multiple pixels. The proposed memory subsystem offers a number of novel methods to increase the efficiency. It is superior in area and power compared to prior work based on the single-pixel wide memory banks.

6. Scalability: In order to support larger WOI, the capacity of the memory subsystem has to be moderately increased. The number of used memory banks remains the same. If the 2D BOI increases in the pixel count, the number of sets (and consecutively banks) increases. For example, in order to double the number of pixels within a 2D BOI to 32 pixels distributed in four lines, the number of sets (and banks) is doubled. This proves that the architecture is fully scalable.

Finally, our memory subsystem is configurable at design time offering a number of tradeoffs. It is possible to tradeoff the performance (number of pixels delivered in parallel) for area by selecting the appropriate number of memory banks. In case of the set-based approach, the number of sets is equal to the number of pixel-lines delivered in parallel. This tradeoff does not affect the supported size of the WOI. In addition, the sliding-L1 approach enables the possibility to tradeoff bandwidth towards the off-chip memory for area. The total memory capacity is selected according to the off-chip memory bandwidth and area constraints.

4.8 Conclusions

Unlike the compute performance, the offered off-chip memory bandwidth failed to follow Moore's law [12]. Meanwhile, the screen resolutions and picture rates have been steadily increasing to a level where the bandwidth became the bottleneck. This resulted in a gap between the available compute resources and memory bandwidth, which directly affects the application performance. Memory subsystems bridge this gap and reduce the off-chip memory bandwidth by moving the repetitive accesses within the memory subsystem. This results in lower power dissipation.

The importance of on-chip memory subsystems in the context of embedded systems has been recognized in the literature [12, 33, 121, 122, 144–148]. Many of the proposed and implemented memory subsystems is based on caches, especially in the general-purpose processor (GPP) world [12, 123, 145, 146, 150–157]. To improve the performance, cache based systems use intelligent prefetching and hardware and software changes. However, the usage of these techniques does not remove all capacity and conflict cache misses and thereby solves the problem only partially. In previous chapters, we have shown that the applications from the target domain access data within a well defined

area. Even though for some algorithms, such as motion estimation and compensation, the locations of accessed pixels are not known at compile time, accesses do not exceed the limits of the search area. Thereby, caches do not seem to be suitable solutions for such applications. Customized memories, such as scratchpads, which enable predictability, seem to be a more adequate choice for our application domain [33, 122, 144, 148, 149, 161–164, 167–170]. Yet, such solutions are less addressed in the literature compared to cache-based solutions. This chapter has proposed one such customized solution. The summary of our proposal and findings is provided below.

In addition to predictability, our goal was to enable un-aligned access with maximal bandwidth towards the Processing Element (PE) and minimal bandwidth towards the off-chip memory. These two goals have been addressed in two Sections, 4.4 and 4.5, respectively. The high bandwidth towards the PE has been achieved through a customized memory architecture that enables un-aligned accesses to a 2D BOI. Pixels are skewed during writing, which makes parallel access to multiple pixel-lines possible. Our solution is based on multi-pixel wide memory banks, which results in an efficient implementation. The proposed solution is parametric, based on three main parameters, the number of sets N_S, the number of banks per set N_B and the number of pixels per addressable location within a memory bank N_P. We have defined a guideline for selection of these parameters according to defined specifications. The specifications include the number of pixels in a BOI, aspect ratio of a BOI, dimensions of the stored WOI, etc. Finally, it is also possible to instantiate a number of scratchpads within a VLIW processor. Instantiating a few scratchpads enables concurrent access to a few BOIs and increases the total offered bandwidth to the Processing Element. This adds the third dimension to our access patterns.

The second important goal of a memory subsystem is keeping the off-chip memory bandwidth minimal. Our scratchpad-based memory subsystem brings down the number of accesses to approximately one access per pixel. The bandwidth requirement is known at compile time. The cost-effective realization has been enabled by using the proposed sliding-L1 approach of refreshing the scratchpad content. The sliding-L1 approach enables to tradeoff scratchpad capacity and off-chip memory bandwidth. This makes our proposal applicable to various use cases.

We have addressed the important question of how many levels a memory subsystem should contain. The drawbacks of the 2-level memory subsystem have been highlighted and a 1-level memory subsystem has been presented as a merge of the two concepts, the L0 memory architecture as presented in Section 4.4 and the sliding-L1 approach as presented in Subsection 4.5.3. The 1-level memory subsystem behaves in the same way from the processing element and the off-chip memory point of view. The 1-level memory subsystem in most cases offers lower area and power compared to the 2-level memory subsystem. In some cases where the processing element issues a large number of accesses per processed block (larger than 16), the 2-level memory subsystem offers lower power. With respect to software, the code becomes simpler in case of the 1-level memory subsystem. A number of load, store and pixel padding instructions disappear. With respect to the algorithm, the 1-level memory subsystem covers a wider, although asymmetrical

WOI, which may be beneficial for some motion estimation based applications.

Finally, the question is how good is our proposed memory subsystem. To answer that question, we have proposed a generic set of six criteria to benchmark a memory subsystem. The criteria are defined as follows: Minimal off-chip memory bandwidth, predictability, high processing element bandwidth (high performance), flexibility, efficiency, and, scalability. Our memory subsystem is benchmarked against the cache-based solutions and customized solutions that use single-pixel wide memory banks over the proposed six criteria and proves to be better on the average than the other solutions. Per individual criterion, it is better than or equal to other evaluated solutions.

5

Conclusions and Future Work

DECADES of research within the field of digital video post processing resulted in an impressive stack of different algorithms. The diversity of these algorithms is visible in many aspects. Algorithms are based on spatial, temporal or spatio-temporal processing, are sometimes recursive, content-adaptive, linear or non-linear. Such diversity causes implementation challenges for a single flexible though application specific processing architecture.

The goal of this thesis is an architecture that enables efficient mapping of high quality algorithms from the video post processing domain. We identify the on-chip memory subsystem as the key component of the proposed architecture to which we devote most of our attention. The importance of the memory subsystem in the context of embedded systems has been recognized in the literature as well [12, 33, 121, 122, 144–148]. Many of the proposed and implemented memory subsystems are based on caches, especially in the general-purpose processor (GPP) world [12, 123, 145, 146, 150–157]. To minimize the datapath stalls that occur due to cache misses, cache-based systems use intelligent prefetching and hardware or software changes. However, the usage of these techniques does not remove all capacity and conflict cache misses and thereby solves the problem only partially.

One of the peculiarities of the video post processing domain is locality of reference, i.e. the processing element always requests pixels located in a predefined vicinity of the currently processed pixel and the same pixel might be requested multiple times. Customized memories such as scratchpads, which enable predictability, may profit from the compile-time knowledge of the executed application (especially the access patterns) and seem to be a more adequate choice for our domain [33, 122, 144, 148, 149, 161–164, 167–170]. Yet, such solutions receive little attention in the literature compared to the cache-

based solutions. In this thesis, we have proposed one such customized solution. The major contributions of this thesis are the *concepts* for data organization that enable un-aligned 2D accesses and minimize the off-chip memory bandwidth. Although the discussion and analysis of these concepts are considered the main contributions and their proof requires a considerable investment of time, we shall not leave the reader without any proof of concept. Therefore, in Appendix C we discuss such a proof.

5.1 Conclusions

To conclude this thesis, we summarize our contributions below.

• We have identified two generic classes of algorithms that together cover a wide range of applications. The first class contains block-based motion estimation/compensation and the second one, pixel-based content-adaptive filtering. Both of them enable high quality video post processing functions to be implemented at a favorable cost level.

1. Block-based motion estimation, particularly the 3DRS [23], enables a substantially better price/performance than other approaches throughout the application domain [21, 22, 35–43, 87, 89]. For the reader's convenience, we have included a summary of 3DRS in Section 2.4.

2. Content-adaptive filtering has been selected as the second distinct approach for two reasons. The first reason is that a number of video post processing functions can be implemented as a content-adaptive filtering providing a high picture quality [44, 45, 82, 99]. The biggest drawback of content-adaptive filtering, the large Look-Up-Table (LUT), has been addressed in recent publications [77, 78]. The second reason is that content-adaptive filtering is generic and also covers algorithms that obtain a good quality with fixed coefficients [100, 101, 113, 115, 116, 118, 119] [Chapter 2].

• We have analyzed the representative algorithms from both algorithmic classes to determine their requirements for a common architecture. This analysis has shown that both classes require un-aligned accesses to a one- or two-dimensional groups of pixels. The group of accessed pixels spreads over multiple time references, adding another dimension, time. Within a particular time instance, this group has a rectangular shape and we call it a Block-Of-Interest (BOI). Using representative algorithms, we demonstrate how both algorithmic classes can be vectorized. The approach we adopt here is to fetch from a local memory subsystem a relatively large BOI, for example containing 16 or 32 pixels. The number of pixels within a BOI equals the width of the vector datapath. After being fetched from the memory subsystem, the BOI is directly processed by the datapath using the vector instruction set. For the sake of demonstration, the used instruction set is generic. We show how the software can be written regardless of any algorithmic peculiarity, such as the number of pixels in the support, the shape of the support, etc. Customized instructions can be added to further improve the performance [Chapter 3].

• We have proposed a scratchpad architecture that enables one-dimensional and two-dimensional accesses to arbitrarily positioned blocks of data. This has been achieved through a customized memory architecture that is based on a specific pixel organization. Pixels are skewed during writing in order to enable parallel access to multiple pixel-lines. Our solution is based on multi-pixel wide memory banks, which results in an efficient implementation. The proposed solution is parametric, based on three main parameters, the number of sets N_S, the number of banks per set N_B and the number of pixels per addressable location within a memory bank N_P. We have defined a guideline for selection of these parameters according to different specifications. The specifications include the number of pixels in a BOI, the aspect ratio of a BOI, the dimensions of the stored area or Window-Of-Interest (WOI), etc. Finally, instantiating a number of scratchpads enables concurrent access to a three-dimensional group of pixels [Section 4.4].

• We have proposed the sliding-L1 concept, a scratchpad organization and addressing technique to minimize the off-chip memory bandwidth. The sliding-L1 concept enables the tradeoff between the scratchpad capacity and off-chip memory bandwidth. The minimal bandwidth where each pixel in the off-chip memory is accessed only once is achieved in case the width of the scratchpad is equal to the width of the picture. According to the proposed concept, the scratchpad width, and thereby the capacity, can almost be halved if the bandwidth requirements increase by typically 10% or less. If the bandwidth requirements further increase by the same percentage, the scratchpad capacity further reduces. The tradeoff curve is exponential and saturates for high bandwidth, the scratchpad capacity remains the same regardless of bandwidth increase. This makes our proposal applicable to various use cases and the system architect can select the optimal point on the curve. In all cases, the bandwidth requirement is known at compile time since the system is predictable. Lastly, the flexibility is enabled through a software-based addressing which only uses aligned accesses [Section 4.5].

• We have discussed the number of memory hierarchy levels. The proposed architecture, which enables un-aligned 1D and 2D accesses and the technique for bandwidth reduction, naturally map onto two levels of memory hierarchy. We show, however, that it is possible to design a memory subsystem based on just one level of memory hierarchy that uses both proposed techniques. The advantages include a reduced software complexity, which has an impact on the overall performance and cost, an improved algorithmic performance and, in most cases, a reduced power dissipation and area. The power dissipation is lower for the 2-level memory subsystem when the processing element requires a large number of accesses per processed block. The number of accesses, which determines the crossover point, depends on the capacity. Our experiments for a given technology show that in the case of a small WOI, it is typically larger than 64. In cases of medium to large WOI, the crossover point is located between 16 and 64 accesses [Section 4.6].

• We have included a benchmark of our memory subsystem. In order to rank a memory subsystem, we have proposed a set of six criteria: Minimal off-chip memory bandwidth, Predictability, High processing element bandwidth (high performance), Flexibility, Efficiency and, Scalability. Our memory subsystem is benchmarked against the cache-based solutions and against the customized solutions that use single-pixel wide memory

banks using the proposed six criteria. Our proposal proves to be better on the average than the other solutions. Per individual criterion, our subsystem is better or equal than other solutions as well [Subsection 4.7].

• We have included a proof of concept. A processor that implements the three-dimensional recursive search motion estimation algorithm [23] and processes HDTV material (1920*1080) in real-time, has been developed and synthesized. Comparison of the performance per square millimeter with the Trimedia TM3270 processor shows that our implementation is eight times more efficient [Appendix C].

5.2 Future work

A number of possibilities exist for further extension of this work. A few that we identified as the most promising, are outlined as Subsections in the text to follow.

5.2.1 Improved user interaction

In this thesis, we have proposed a method for off-chip memory bandwidth minimization and a data organization that enables efficient un-aligned accesses. However, the user interaction with the proposed memory subsystem, or the Application Programming Interface (API), can further improve. Namely, the application programmer is responsible for placing the scratchpad read and write operations. Here we list the problems and propose a research direction to address them.

1. During the read operations, the eventual pixel padding has to be taken care of explicitly in software. Pixel padding is typically needed at picture borders and its implementation in software causes several negative effects: Conditional code execution, increased complexity of software, increased program memory capacity, performance penalties, longer software design and verification time, etc. One of the possibilities is to store the padded pictures in the off-chip memory. This solution increases the off-chip memory bandwidth and is therefore particularly inadequate in case of larger WOI dimensions. Somewhat better solution would be that the DMA interface performs the padding. This does not increase the off-chip memory bandwidth, but still requires larger on-chip memory bandwidth and capacity. An alternative solution is to implement the pixel padding during reading from the lowest level of the memory subsystem ($L0/L01_M$). This solution leaves the off-chip memory bandwidth, the on-chip memory bandwidth and capacity intact.

2. The application programmer places the store commands at proper locations in software. The 1-level memory subsystem greatly reduces this task by omitting the unnecessary store commands from higher to lower level of memory hierarchy. The DMA helps further from a performance point of view. However, the application programmer must have in-depth knowledge of the memory subsystem and the DMA. This problem can be addressed by a software template approach. The application programmer would start from a generic software template that already

has loop structure and positions of the DMA requests set in accordance with the sliding-L1 method. We could go one step further and envisage a complete Graphical User Interface (GUI) that will enable the application programmer to graphically enter the picture scanning structure with dimensions of the sliding-L1. Based on the graphically entered input, a software template can be generated with loops structure and dimensions, as well as the code fragments related to memory subsystem loads. The additional output of the GUI may be the cost information such as the bandwidth towards the off-chip memory, the required capacity and the estimated area and power of the memory subsystem, etc.

5.2.2 On-chip improvements

On-chip bandwidth reduction:

Chapter 4 shows that due to four major reasons, the 1-level memory subsystem outperforms the 2-level memory subsystem. Probably, the only exception is power dissipation, which is lower for the 2-level memory subsystem in cases where the processing element issues a large number of block accesses. This happens only beyond a certain number of block accesses because the data traffic from the L1 to the L0 scratchpad causes significant power dissipation. We did not investigate techniques for reducing this bandwidth. Further research might go into the direction of on-chip memory bandwidth reduction and thereby reduce power dissipation of the 2-level memory subsystem. Additional argument that goes in favor of the 2-level memory subsystem is that a small L0 scratchpad could be faster, since it has significantly smaller capacity than the L1 scratchpad.

On-chip memories:

The selected technology influences the on-chip memory subsystem. For example, in smaller technology nodes, such as 65 nm, the leakage power is larger. Significant differences exist even for the same technology node, between different vendors. Additionally, memory banks can be optimized for speed or power dissipation and there are also different memory technologies, such as SRAM and DRAM. Design parameters, such as the pixel encoding precision, the supported BOIs and the instantiated on-chip memory capacity coupled with specific memory technology can lead to different applications of the proposed pixel skewing during writes. This thesis analyzed only a subset of the mentioned technologies. Further research could focus to effects of various technologies to efficient configuring the parameters of the L0/L01$_{\mathrm{M}}$ scratchpads.

Technology also influences the decision considering on-chip integration of reference pictures. The miniaturization has led to the integration of tens and hundreds of millions of transistors on a single chip, implementing multiple processors. This trend may further continue and include the integration of the off-chip memory, which contains the picture references. The dominant factor that prevents this from massively happening is cost. The technology has been advancing to smaller and smaller nodes (e.g. 65 nm or 45 nm CMOS process) resulting in increased density of components. However, the additional consequence of that miniaturization is increased cost. In the move from 0.5 μm to 90 nm technology, the mask costs have increased by a factor of 45 [177]. In addition, the cur-

rent technology has a miniaturization limit. New technologies, like hafnium, are pushing the limits to 45 nm and beyond, and enable both faster and more energy efficient chips. However, even such advanced technologies eventually will stop since the thickness of the transistor structure is already measured in number of molecules [178]. On the other side, the picture resolutions have already increased from SD to HD Ready and full HD. There are indications that this trend may continue beyond full HD to resolutions such as quad HDTV (3840*2160). In addition to higher resolution, the drive for higher picture quality may cause further increase of the number of used picture references.

On the other side, the memory technology is also improving. For example, some studies from Intel discuss the 3D stacking, i.e. mounting the CPU on a RAM, which is expected to provide an order of magnitude more bandwidth and lower power dissipation. However, in spite of trends to improve the current memory technology, it is not likely that in the near future the picture references will be stored on-chip in case of large picture resolutions (HD-like). The probability that this integration massively happens for smaller picture sizes and refresh rates is higher. This will enable faster accesses and also wider access points, which alleviates the bandwidth problem. Although for power reasons, the memory hierarchies will remain interesting, this integration will bring a new light onto the problems analyzed in this thesis.

5.2.3 Video coding

Even though not directly targeted by this work, video coding is an important video application of today's world. Many, international, national and proprietary video coding standards exist. At product level, whether it is a video recorder or player, portable or not, digital camcorder or still image camera, it almost always contains a video coder, or a decoder. There are two major common ingredients of a majority of coding standards:

- They are based on motion estimation/compensation, and
- They use some sort of pixel interpolation.

Since the proposed memory organization strongly supports (block-based) motion compensated processing as well as pixel interpolation and filtering with a flexible shape of the filter footprint, our architecture is a good candidate for video coding. Future work could investigate how well the proposed concepts fit in the video coding application domain. One of the questions is how efficient are the proposed ideas in comparison to hardwired solutions. The proposed memory subsystem in combination with the programmable architecture offers flexibility, but the specifications of a video standard are rather fixed. Furthermore, there are only a limited number of international standards that are currently in use or emerging. The additional price of programmability compared to hardwired solutions includes the cost of the program memory, additional register files of uniform width, connectivity. etc. This cost can be reduced (loosing flexibility) if some parts of the algorithm are realized in hardwired logic. An example may be that the output of our memory subsystem goes directly to the hardwired interpolation unit and the hardwired unit that computes the sum-of-absolute-differences value (or some other error criterion).

A

The second level of the memory hierarchy

THIS appendix quantifies the need for the second level of memory hierarchy. In the text below, we derive the equation for number of accesses, NA, within the off-chip memory (denoted as L2). We analyze two common scanning traces of the L0 scratch-pad, left-to-right-top-to-bottom (LRTB) and meandering. The equations show that the number of accesses in both cases is few times bigger than the minimal one. Finally, we quantify the difference between the meandering and LRTB scanning traces in terms of bandwidth requirement. The results show that typically, the LRTB scanning trace results in 5-10% more bandwidth towards the upper level of the hierarchy. The equations are derived assuming that the upper level of the memory hierarchy is the off-chip memory. If the L1 scratchpad is present in the system, the computed bandwidth would be exposed to it instead.

$$\begin{aligned} NA_{LRTB} = & \\ &= L2_Y(L2_X - 1)L0_Y \\ &+ L2_Y L0_X L0_Y \\ &= L2_Y(L2_X - 1 + L0_X)L0_Y \end{aligned} \tag{A.1}$$

$$\begin{aligned} NA_{MEAND} = & \\ &= L2_Y(L2_X - 1)L0_Y \\ &+ (L2_Y - 1)L0_X \\ &+ L0_X L0_Y \end{aligned}$$

$$
\begin{aligned}
&= L2_Y(L2_X-1)L0_Y + L2_Y L0_X + L0_X L0_Y - L0_X \\
&= L2_Y((L2_X-1)L0_Y + L0_X) + L0_X(L0_Y-1) \\
&\approx L2_Y((L2_X-1)L0_Y + L0_X)(1 + \overbrace{\frac{L0_Y-1}{L2_Y}}^{<1}\overbrace{\frac{L0_X}{(L2_X-1)L0_Y + L0_X}}^{\approx 0}) \approx \\
&\approx L2_Y((L2_X-1)L0_Y + L0_X)
\end{aligned}
\tag{A.2}
$$

In the last approximation we made, the following two properties have been used: $L2_Y > L0_Y - 1$ and $L2_X \gg 1$; $L0_X \approx L0_Y$. The consequence of the last two properties is that $L2_X L0_Y \gg L0_X$. Finally, $(L2_X - 1)L0_Y + L0_X \gg L0_X$.

Note that the above two equations are derived according to the L0 scratchpad's input bandwidth. Namely, they assume transfers of even those blocks that are located "outside" the picture. The pixels close to picture border should be read from the L1 scratchpad, conditionally padded outside the picture, and stored in the L0 scratchpad. This border effect is present regardless of the scanning order and increases the bandwidth towards the L0 scratchpad. When computing the ratio between two approaches, for example the ratio between the bandwidth caused by the LRTB and meandering scan, these border effects will partly cancel each other.

Alternative solution would be to clip the read coordinates provided to the L0 scratchpad. To compute the bandwidth and bandwidth differences according to this solution, the equations would stay similar, only the picture dimensions should be reduced to the left and right by $L0_X/2$ blocks and to the top and bottom by $L0_Y/2$ blocks. However, clipping the read coordinates near the picture borders might not be allowed in some cases because it modifies the algorithm behavior. Example is motion estimation where conditional clipping of motion vectors might not be allowed.

The ratio between the computed number of accesses for the LRTB scanning style and the minimal number of accesses is defined with the following equation.

$$
\begin{aligned}
&\frac{NA_{LRTB}}{NA_{MIN}} = \\
&= \frac{L2_Y(L2_X - 1 + L0_X)L0_Y}{L2_Y L2_X} \\
&= \frac{(L2_X - 1 + L0_X)L0_Y}{L2_X} \\
&= L0_Y(1 + \frac{L0_X - 1}{L2_X}) \approx L0_Y
\end{aligned}
\tag{A.3}
$$

The equation $L2_X > L0_X$ is always valid and in the first order approximation, $L2_X \gg L0_X$. Note that this is not always the case. Should that not be the case, the above ratio is even bigger than $L0_Y$ which even more emphasizes the need for the second

level of memory hierarchy. Similarly is defined the ratio between the computed number of accesses for the meandering scanning style and the minimal number of accesses:

$$\begin{aligned}
&\frac{NA_{MEAND}}{NA_{MIN}} = \\
&= \frac{L2_Y((L2_X-1)L0_Y + L0_X)}{L2_Y L2_X} \\
&= \frac{L2_X L0_Y + L0_X - L0_Y}{L2_X} \approx L0_Y + \frac{L0_X}{L2_X} - \frac{L0_Y}{L2_X} \\
&= L0_Y(1 + \overbrace{\frac{1}{L2_X L0_Y / L0_X}}^{\approx 0} - \overbrace{\frac{1}{L2_X}}^{\approx 0}) \approx L0_Y
\end{aligned} \tag{A.4}$$

In the last approximation we made, we used the following property: $L2_X \gg 1$; $L0_X \approx L0_Y$. Both equations result with the same ratio, being equal to the vertical dimension of the L0 scratchpad, $L0_Y$.

In the following text, we investigate the bandwidth differences between the LRTB and meandering scanning orders. In the case of the LRTB scanning order, the full refill occurs at the beginning of each block line, while in the case of meandering, only once, at the beginning of the scanning trace. In the following equations, we quantify that difference. We use Equations A.1 and A.2

$$\begin{aligned}
&\frac{NA_{LRTB}}{NA_{MEAND}} = \\
&= \frac{L2_Y(L2_X - 1 + L0_X)L0_Y}{L2_Y((L2_X-1)L0_Y + L0_X)} \\
&= \frac{(L2_X-1)L0_Y + L0_X L0_Y}{(L2_X-1)L0_Y + L0_X} \\
&= \frac{(L2_X-1)L0_Y + L0_X + L0_X(L0_Y-1)}{(L2_X-1)L0_Y + L0_X} \\
&= 1 + \frac{L0_X(L0_Y-1)}{(L2_X-1)L0_Y + L0_X} \\
&= 1 + \frac{L0_Y - 1}{\frac{L2_X - 1}{L0_X} L0_Y + 1} \\
&\approx 1 + \frac{L0_Y - 1}{\frac{L2_X}{L0_X} L0_Y + 1}
\end{aligned} \tag{A.5}$$

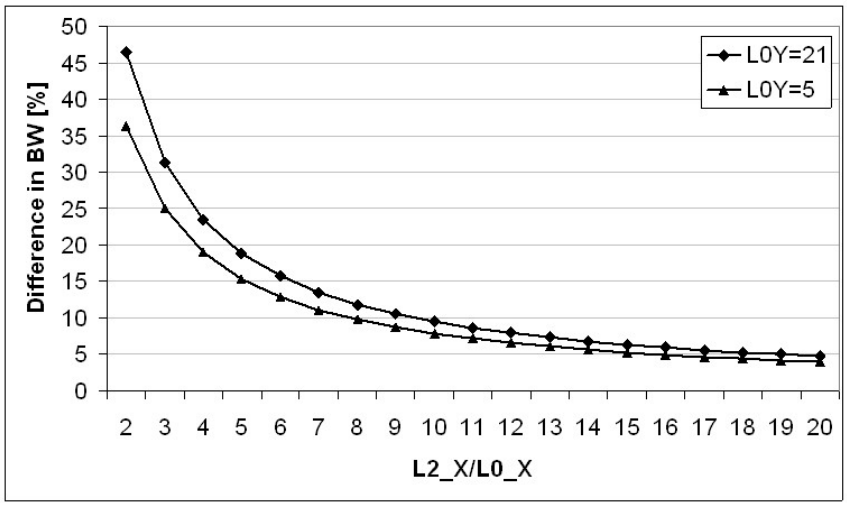


Figure A.1: The difference in bandwidth requirements between the LRTB and meandering scanning order as a function of the ratio between the picture width and the width of the L0 scratchpad. The results for two heights of the L0 scratchpad are provided.

Based on Equation A.5, it is possible to compute the difference between the number of accesses between the LRTB and meandering scanning traces. The difference, showing for how many percents is the bandwidth of the LRTB scanning order larger than the meandering one is computed as follows.

$$\begin{aligned}\frac{NA_{LRTB} - NA_{MEAND}}{NA_{MEAND}} &= \\ &= \frac{NA_{LRTB}}{NA_{MEAND}} - 1 \\ &= 1 + \frac{L0_Y - 1}{\frac{L2_X}{L0_X} L0_Y + 1} - 1 \\ &= \frac{L0_Y - 1}{\frac{L2_X}{L0_X} L0_Y + 1}\end{aligned} \tag{A.6}$$

This difference is computed as a function of $\frac{L2_X}{L0_X}$ and $L0_Y$. Figure A.1 illustrates this difference for two values of $L0_Y$. The LRTB scanning trace typically requires 5-10% more bandwidth than the meandering.

B

Capacity and bandwidth of the L1 scratchpad

THIS appendix provides the derivation of equations related to the capacity and bandwidth requirements of the stripe-based, region-based and sliding-L1 scratchpads. As stated in Subsection 4.5.7, we analyze two cases, $L0_X = L2_X/10$; $L0_Y = L2_Y/10$ and $L0_X = L2_X/20$; $L0_Y = L2_Y/20$. In this appendix, we also use the approximation that one block-line is equal to 1% $L2_Y$. Example of usage of this approximation is given here for the case of the vertical dimension of the sliding-L1 scratchpad where $L0_Y = 1/10 * L2_Y : L1_Y = L0_Y + 1 = 1/10 * L2_Y + 1/100 * L2_Y = 11/100 * L2_Y$.

B.1 Case 1: $L0_X = L2_X/10$; $L0_Y = L2_Y/10$

For the case of the region-based and sliding-L1 approaches, we assume that all regions have the same dimensions. This should be the goal of the architect as the number of regions defines the number of region overlaps and thereby the bandwidth overhead. For the case of $L0_X = L2_X/10$; $L0_Y = L2_Y/10$, the number of region overlaps is defined with the following equation. This derivation is performed replacing the selected L0 scratchpad's dimensions in Equation 4.16:

$$n_x = \lceil \frac{L2_X - L1_X}{L1_X - 1/10 * L2_X} \rceil = 10 \lceil \frac{L2_X - L1_X}{10L1_X - L2_X} \rceil \tag{B.1}$$

In case all regions are identical, the same value for n_x and n_y is computed with and without the usage of the ceiling function. Thereby, from the above equation, the ceiling function is omitted and $L1_X$ can be derived:

$$n_x(10L1_X - L2_X) = 10(L2_X - L1_X) \Longrightarrow L1_X = \frac{10 + n_x}{10(n_x + 1)} L2_X \tag{B.2}$$

Similar equations can be derived for the case of $L1_Y$ (every occurrence of n_x has to be replaced by n_y).

$$\begin{aligned} CAP_{STRIPE-L1} &= L2_X(L0_Y+1) = \frac{11}{100}L2_X L2_Y \\ CAP_{REGION-L1} &= L1_X L1_Y = \frac{10+n_x}{10(n_x+1)}\frac{10+n_y}{10(n_y+1)}L2_X L2_Y \\ CAP_{SLIDING-L1} &= L1_X(L0_Y+1) = \frac{11}{100}\frac{10+n_x}{10(n_x+1)}L2_X L2_Y \end{aligned} \tag{B.3}$$

Subsection 4.5.7 compares the three approaches using the generic numbers. The comparison is performed with respect to the reference point, which is the stripe-based approach. We continue with that here. Thereby, relative capacities of the region-based and the sliding-L1 scratchpad are given with the following equations.

$$\begin{aligned} \frac{CAP_{REGION-L1}}{CAP_{STRIPE-L1}} &= \frac{100}{11}\frac{10+n_x}{10(n_x+1)}\frac{10+n_y}{10(n_y+1)} = \frac{1}{11}\frac{10+n_x}{n_x+1}\frac{10+n_y}{n_y+1} \\ \frac{CAP_{SLIDING-L1}}{CAP_{STRIPE-L1}} &= \frac{10+n_x}{10(n_x+1)} \end{aligned} \tag{B.4}$$

Bandwidth overhead towards the off-chip memory of the region-related approaches is already defined with Equations 4.17, 4.19 and 4.21. Here we just replace the dimensions of the L0 scratchpad with the chosen values.

$$\begin{aligned} \frac{NA_{STRIPE-L2-L1}}{NA_{MIN-L2-L1}} &= 1 \\ \frac{NA_{REGION-L2-L1}}{NA_{MIN-L2-L1}} &= 1 + n_y\frac{L0_Y}{L2_Y} = 1 + \frac{1}{10}n_y \\ \frac{NA_{SLIDING-L2-L1}}{NA_{MIN-L2-L1}} &= 1 + n_x\frac{L0_X}{L2_X} = 1 + \frac{1}{10}n_x \end{aligned} \tag{B.5}$$

Here we analyze the data traffic between the L1 and L0 scratchpads. We start from the generic equation of the region-based approach, Equation 4.18. This traffic is minimal in case of the stripe-based approach. The reason is that this approach utilizes only one region. In such a case, $n_x = n_y = 0$. Based on this and Equation A.1, the number of accesses (transfers) between the stripe-based L1 scratchpad and the L0 scratchpad for the LRTB scanning order is equal to $(n_x+1)(n_y+1)L2_Y(L2_X - 1 + L0_X)L0_Y = L2_Y(L2_X - 1 + L0_X)L0_Y$. The following equation computes how many times is the number of accesses between the region-based L1 and the L0 scratchpad bigger then the number of accesses between the stripe-based L1 and the L0 scratchpad.

$$\begin{aligned}
&\frac{NA_{REGION-L1-L0}}{NA_{STRIPE-L1-L0}} = \\
&= (n_x+1)(n_y+1)\frac{L1_Y(L1_X-1+L0_X)}{L2_Y(L2_X-1+L0_X)} \\
&= (n_x+1)(n_y+1)\frac{10+n_y}{10(n_y+1)}\frac{(\frac{10+n_x}{10(n_x+1)}-1/100+1/10)L2_X}{L2_X-1/100L2_X+1/10L2_X} \\
&= (n_x+1)(n_y+1)\frac{19n_x+109}{109(n_x+1)}\frac{10+n_y}{10(n_y+1)} \\
&= \frac{1}{1090}(19n_x+109)(10+n_y)
\end{aligned} \tag{B.6}$$

The matching equation for the sliding-L1 scratchpad is obtained based on the previous one for the case of $n_y = 0$.

$$\frac{NA_{SLIDING-L1-L0}}{NA_{STRIPE-L1-L0}} = (n_x+1)\frac{19n_x+109}{109(n_x+1)} = \frac{1}{109}(19n_x+109) \tag{B.7}$$

B.2 Case 2: $L0_X = L2_X/20$; $L0_Y = L2_Y/20$

We repeat the same process for the smaller L0 scratchpad. Similar to the first case, we use the property that one block-line is equal to 1% $L2_Y$ on few occasions, for example, $L0_Y + 1 = 1/20 * L2_Y + 1/100 * L2_Y = 6/100 * L2_Y$. Let us first derive the set of dimensions of the L1 scratchpad. We start again from Equation 4.16 and replace $L0_X = 1/20 * L2_X$:

$$n_x = \lceil\frac{L2_X - L1_X}{L1_X - 1/20 * L2_X}\rceil = 20\lceil\frac{L2_X - L1_X}{20L1_X - L2_X}\rceil \tag{B.8}$$

Assuming the same region dimensions, from the above equation the ceiling function is omitted and the width of the L1 scratchpad $L1_X$ can be derived.

$$n_x(20L1_X - L2_X) = 20(L2_X - L1_X) \Longrightarrow L1_X = \frac{20+n_x}{20(n_x+1)}L2_X \tag{B.9}$$

Similar equations can be derived for the case of $L1_Y$ (every occurrence of n_x has to be replaced by n_y). Based on these equations, we derive capacities of the three approaches.

$$\begin{aligned}
CAP_{STRIPE-L1} &= L2_X(L0_Y+1) = \frac{6}{100}L2_XL2_Y \\
CAP_{REGION-L1} &= L1_XL1_Y = \frac{20+n_x}{20(n_x+1)}\frac{20+n_y}{20(n_y+1)}L2_XL2_Y \\
CAP_{SLIDING-L1} &= L1_X(L0_Y+1) = \frac{6}{100}\frac{20+n_x}{20(n_x+1)}L2_XL2_Y
\end{aligned} \tag{B.10}$$

Thereby, relative capacities of the region-based and the sliding-L1 scratchpad are given with the following equations.

$$\begin{aligned}\frac{CAP_{REGION-L1}}{CAP_{STRIPE-L1}} &= \frac{100}{6}\frac{20+n_x}{20(n_x+1)}\frac{20+n_y}{20(n_y+1)} = \frac{5}{120}\frac{20+n_x}{n_x+1}\frac{20+n_y}{n_y+1} \\ \frac{CAP_{SLIDING-L1}}{CAP_{STRIPE-L1}} &= \frac{20+n_x}{20(n_x+1)}\end{aligned} \tag{B.11}$$

Similar as in the first case, we compute the bandwidth overheads towards the off-chip memory.

$$\begin{aligned}\frac{NA_{STRIPE-L2-L1}}{NA_{MIN-L2-L1}} &= 1 \\ \frac{NA_{REGION-L2-L1}}{NA_{MIN-L2-L1}} &= 1 + n_y\frac{L0_Y}{L2_Y} = 1 + \frac{1}{20}n_y \\ \frac{NA_{SLIDING-L2-L1}}{NA_{MIN-L2-L1}} &= 1 + n_x\frac{L0_X}{L2_X} = 1 + \frac{1}{20}n_x\end{aligned} \tag{B.12}$$

Finally, we analyze the data traffic from L1 to L0 scratchpad.

$$\begin{aligned}&\frac{NA_{REGION-L1-L0}}{NA_{STRIPE-L1-L0}} = \\ &= (n_x+1)(n_y+1)\frac{L1_Y(L1_X-1+L0_X)}{L2_Y(L2_X-1+L0_X)} \\ &= (n_x+1)(n_y+1)\frac{20+n_y}{20(n_y+1)}\frac{(\frac{20+n_x}{20(n_x+1)}-1/100+1/20)L2_X}{L2_X-1/100L2_X+1/20L2_X} \\ &= (n_x+1)(n_y+1)\frac{9n_x+104}{104(n_x+1)}\frac{20+n_y}{20(n_y+1)} \\ &= \frac{1}{2080}(9n_x+104)(20+n_y)\end{aligned} \tag{B.13}$$

The matching equation for the sliding-L1 scratchpad is obtained based on the previous one for the case of $n_y = 0$.

$$\frac{NA_{SLIDING-L1-L0}}{NA_{STRIPE-L1-L0}} = (n_x+1)\frac{9n_x+104}{104(n_x+1)} = \frac{1}{104}(9n_x+104) \tag{B.14}$$

C

Proof of concept

THIS appendix consists of two parts. The first (larger) part presents a processor that is based on the 2-level memory hierarchy presented in this thesis, and, a mapping of a motion estimation algorithm. It also compares the achieved efficiency against the well-known media processor. The second part presents the stand-alone synthesis results of the L0 scratchpad, which show the percentage of logic used in the L0 scratchpad.

Processor and mapped algorithm characterization

To demonstrate the results achieved by applying the concepts presented in this thesis, a processor has been designed that can execute algorithms from the application domain. As an illustration, the 3DRS motion estimation algorithm is mapped onto this processor. In many applications, this estimator is an inevitable ingredient for achieving high quality output pictures. From an implementation point of view, it is a challenging algorithm, since it is recursive, implying data dependencies. The input resolution was set to 1920*1080. The 3DRS estimator evaluates 7 quarter-pel accurate motion vector candidates per block of 8*8 pixels. The sum-of-absolute-differences (SAD) was used as the error criterion comparing interpolated blocks of 8*8 pixels. This implementation did not use any algorithmic technique for operation count reduction, such as pixel subsampling at the SAD level or block subsampling at the picture level [120]. In other words, each block of 8*8 pixels within a picture was evaluated using all its pixels. In addition to the best vector computation, the visualization of computed vector values is performed as well [34]. The reason is that one of the targets of this implementation is an FPGA-based demonstrator system connected to an HDTV panel. The vector visualization requires additional computation, on-chip memory storage (double buffering) and off-chip memory bus traffic (storing the U/V components to the off-chip memory).

Table C.1 The details of the functionally correct 3DRS motion estimation algorithm implementation.

Resolution and picture-rate	1920*1080p @ 60fps
Effective search area	224*96
Algorithm details	7 quarter-pel accurate cand., 8*8 SAD, no sub-sampling of any kind, vector visualization
CMOS Technology	TSMC 90nm, WCC
Frequency	250 MHz
Program memory (PMEM)	RAM, 512 bits wide, 1024 (850 used) words
ILP at vector datapath	93%
Datapath configuration	4 scalar and 4 16-way vector issue slots
Area logic [mm^2]	1.54
Area PMEM (64 Kbytes) [mm^2]	1.1
Area 32-bit mem. (4 Kbytes) [mm^2]	0.078
Area L0 mem. (32 Kbytes) [mm^2]	0.624
Area L1 mem. (88 Kbytes) [mm^2]	1.1
Area total [mm^2]	4.442
BW towards the off-chip mem.	read: 1.29 access/pixel

The processor characteristics are summarized in Table C.1, based on scheduling results and synthesis reports. The processor's datapath consists of 4 vector and 4 scalar issue slots. Vector issue slots are 16-way. The scalar and vector instruction set implements basic instructions such as addition, subtraction, multiplication, shifting, comparison, etc. The memory hierarchy enables access to an arbitrarily positioned block of pixels, which may contain 4*4, 8*2 or 16*1 pixels. In this particular implementation, we only used 8*2 access patterns. Each pixel from the off-chip memory is on the average accessed 1.29 times. The processor was designed using a Silicon Hive technology [18].

The results of this thesis suggest that a 1-level memory subsystem offers a number of advantages, which are applicable to this particular algorithm as well. However, this processor is based on a 2-level memory hierarchy, since a functionally correct HDL description of a 1-level memory subsystem was not available at the time of writing. This means that the performance can be increased and the cost reduced by using the 1-level memory subsystem. The performance can be further boosted by further customizing the memory subsystem (for example, supporting the on-the-fly (bi)linear interpolation), and in general, adding custom instruction set.

To judge the efficiency of our implementation, we compare it against the selected

Table C.2 Stand-alone synthesis of the L0/L01$_{\mathrm{M}}$ scratchpad.

Bank depth [words]	256	512	1024	2048	4096
Total capacity [KB]	8	16	32	64	128
Example dim. [pix.]	128*64	256*64	256*128	512*128	1024*128
Total area [mm^2]	0.433	0.513	0.672	0.996	1.655
Memory area [mm^2]	0.385	0.465	0.624	0.947	1.606
Logic area [mm^2]	0.048	0.048	0.048	0.049	0.049
Percentage of logic [%]	11	9.3	7.2	4.9	3.0

state-of-the-art IC. The selected IC should execute similar algorithm, ideally the 3DRS. One of the programmable ICs that is used for video post-processing (based on the 3DRS motion estimation as well) that lived long to produce several generations, and received significant publicity, is the Trimedia family of processors [123, 124, 130]. These facts make Trimedia a suitable comparison point, and we compare our proposal against the recently published (2007) 3DRS implementation on the Trimedia TM3270 processor [130]. The cited implementation performs two motion estimation scans on progressive HDTV input film material, 1920*1080 at 24 Hz. This is equivalent to one motion estimation scan performed at 48 Hz. The motion vectors are quarter-pel accurate and 6-8 of them are evaluated per block. To process this material real-time, the Trimedia TM3270 needs 1240 Mcycles/sec. For a 60 Hz motion estimation, the requirement increases to 1550 Mcycles/sec. The Trimedia processor includes 128 Kbytes cache and occupies 8 mm^2 in 90nm CMOS technology. The synthesis of our solution resulted in 4.442 mm^2, also in 90nm technology. Assuming the layout efficiency of 50%, our solution occupies approximately 6 mm^2. To process 60 Hz material real-time, it requires 250 Mcycles/sec. Comparing the performance per square millimeter, we conclude that our solution is eight times more efficient.

In order to achieve real-time, the authors in the cited publication are forced to use algorithmic modifications to reduce the implementation cost. The following methods are used: block hopping (only 50% of the blocks are processed), pixel subsampling of the SAD window by a factor of two, and, picture size reduction by a factor of two. Each of these methods reduces the cost of implementation by approximately factor of two. Since our proposal is also based on a programmable architecture, we can implement all of the mentioned methods. The appropriate software modification would lead to the similar compute complexity reduction factors.

Stand-alone synthesis of the L0/L01$_{\mathrm{M}}$ scratchpad

To complete the story, we provide the stand-alone synthesis results of a L0/L01$_{\mathrm{M}}$ scratchpad. The scratchpad consists of four sets, each containing two banks. A bank is

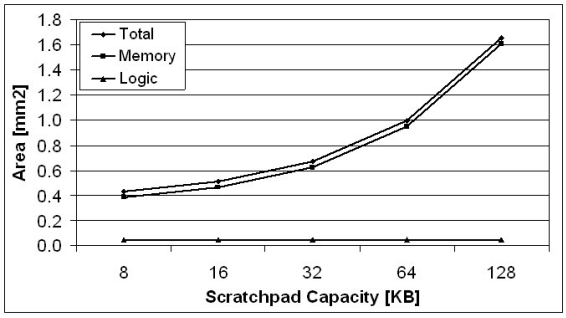


Figure C.1: Picture illustrates the synthesis results of the L0/L01$_{M}$ scratchpad.

32-bit, or four 8-bit pixels wide. The scratchpad can deliver a BOI that consists of 16*1, 8*2 or 4*4 pixels. The block was synthesized using the 90nm TSMC technology, WCC at 200 MHz. The results are summarized in Table C.2 and plotted in Figure C.1. The table is organized in five columns, each providing the data for a specific depth of a single bank. In addition to synthesis results, an example dimensions of a WOI are provided. The synthesis numbers are given for the memory and logic separately. The logic percentage stays in the range of 3% to 11%, depending on the instantiated memory capacity. This percentage supports our starting assumption from Section 4.4, that the memory banks are the dominant cost factor.

References

[1] J. P. Eckert and J. W. Mauchly, "Outline of plans for development of electronic computors", 1946.

[2] H.H. Goldstine and A. Goldstine, "The Electronic Numerical Integrator and Computer (ENIAC)", *Mathematical Tables and Other Aids to Computation, 2(15)*, pp. 97–110, 1946.

[3] C. E. McTiernan, "The ENIAC patent", *IEEE Annals of the History of Computing*, vol. 20, no. 2, pp. 54–58, 1998.

[4] Press release from Bell Telephone Laboratories, "A. M. papers of Thursday, July 1, 1948", July 1948.

[5] "The Nobel Prize in Physics 1956: For their researches on semiconductors and their discovery of the transistor effect, William Bradford Shockley, John Bardeen, Walter Houser Brattain", 1956.

[6] R. Adams, *The Transistor Enigma*, ISBN 0-473-06678-5, 2003.

[7] G. E. Moore, "Cramming more components onto integrated circuits", *Electronics*, vol. 38, no. 8, pp. 114–117, 1965.

[8] W. C. Rhines, "Moore's law is unconstitutional", in *IEEE International Conference on VLSI Design*, pp. 31–32, 2005.

[9] T. von Sydow, H. Blume, and T. G. Noll, "Performance analysis of general purpose and digital signal processor kernels for heterogeneous systems-on-chip", *Advances in Radio Science*, pp. 172–175, 2003.

[10] S. H. Unger, "A computer oriented towards spatial problems", in *Proceedings of Institute of radio engineers 46*, pp. 203–208, 1958.

[11] K. Ebcioglu, "Some design ideas for a VLIW architecture for sequential-natured software", *Parallel Processing*, pp. 3–21, 1988.

[12] J. L. Hennessy and D. A. Patterson, *Computer architecture a quantitative approach*, Morgan Kaufmann Publishers, ISBN 1-55860-724-2, 2003.

[13] V. Aue, J. Kneip, M. Weiss, M. Bolle, and G. Fetweis, "A design methodology for high performance ICs: wireless broadband radio baseband case study", in *EuroMicro Symposium on Digital System Design*, pp. 16–20, 2001.

[14] A. Hoffmann, T. Kogel, A. Nohl, G. Braun, O. Schliebusch, O. Wahlen, A. Wieferink, and H. Meyr, "A novel methodology for the design of application-specific instruction-set processors (ASIP) using a machine description language", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1338–1354, 2001.

[15] H. Meyr, "System-on-chip for communications: the dawn of ASIPs and the dusk of ASICs", in *IEEE International Workshop on Signal Processing Systems*, pp. 4–5, 2003.

[16] H. Jeng, L. Feipei, E. Naroska, and U. Schwiegelshohn, "Multimedia ASIP SoC co-design based on multicriteria optimization", in *IEEE International Conference on Consumer Electronics*, pp. 342–343, 2001.

[17] M. Mbaye, N. Belanger, Y. Savaria, and S. Pierre, "Application Specific Instruction-Set Processor Generation for Video Processing Based on Loop Optimization", in *IEEE International Symposium on Circuits and Systems*, pp. 3515–3518, 2005.

[18] Silicon Hive technology: http://www.siliconhive.com.

[19] S. Pees, A. Hoffmann, V. Zivojinovic, and H. Meyr, "LISA-machine description language for cycle-accurate models of programmable DSP architectures", in *IEEE/ACM Design Automation Conference*, pp. 933–938, 1999.

[20] A. Beric, G. de Haan, R. Sethuraman, and J. van Meerbergen, "Algorithm/architecture co-design of the generalized sampling theorem based de-interlacer", in *IEEE International Symposium on Circuits and Systems*, pp. 2943–2946, 2005.

[21] C. Ciuhu and G. de Haan, "A 2-dimensional generalized sampling theory and its application to deinterlacing", in *Visual Communications and Image Processing Conference*, pp. 700–711, 2004.

[22] E. B. Bellers, J. W. van Gurp, J. G. W. M. Janssen, R. Braspenning, and R. Wittebrood, "Solving occlusion in Frame-Rate up-Conversion", in *IEEE International Conference on Consumer Electronics*, pp. 1–2, 2007.

[23] G. de Haan, P. W. A. C. Biezen, H. Huijgen, and O. A. Ojo, "True motion estimation with 3-D recursive search block-matching", *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 3, no. 5, pp. 368–379, Oct. 1993.

[24] G. de Haan, "Video format conversion", *Journal of the SID*, vol. 8, no. 1, pp. 79–87, Oct. 2000.

[25] L.A. Vervoort, P. Yeung, and A. Reddy, "A white paper: Addressing memory bandwidth needs for next-generation digital tvs with cost-effective, high-performance consumer DRAM solutions", 2007.

[26] K. Ebcioglu, "A compilation technique for software pipelining of loops with conditional jumps", *Proceedings of MICRO-20*, pp. 69–79, 1987.

[27] K. Asanovic, *Vector Microprocessors*, Ph.D. thesis, University of California at Berkeley, 1998.

[28] R. Lee, "Accelerating multimedia with enhanced microprocessors", *IEEE Micro*, vol. 15, no. 2, pp. 22–32, 1995.

[29] Ruby B. Lee, "Subword parallelism with MAX-2", *IEEE Micro*, vol. 16, no. 4, pp. 51–59, 1996.

[30] R. Lee and J. Huck, "64-bit and multimedia extensions in the PA-RISC 2.0 architecture", in *COMPCON '96: Proceedings of the 41st IEEE International Computer Conference*, pp. 152–160, 1996.

[31] P. Clarke, "Silicon Hive launches image processor, EE Times", May 2006.

[32] A. Beric, R. Sethuraman, H. Peters, J. van Meerbergen, G. de Haan, and C. A. Pinto, "A 27mw 1.1 mm^2 motion estimator for picture-rate up-converter", in *IEEE International Conference on VLSI Design*, pp. 1083–1088, 2004.

[33] A. Beric, R. Sethuraman, J. van Meerbergen, and G. de Haan, "Memory-centric motion estimator", in *IEEE International Conference on VLSI Design*, pp. 816–819, 2005.

[34] G. de Haan, *Video processing for multimedia systems*, University press Eindhoven, ISBN 90-9014015-8, 2001.

[35] G. de Haan, J. Kettenis, B. Deloore, and A. Loehning, "IC for Motion Compensated 100Hz TV, with a Smooth Motion Movie-Mode", *IEEE Transactions on Consumer Electronics*, vol. 42, no. 2, pp. 165–174, May 1996.

[36] J. W. Woods and S. Han, "Hierarchical motion compensated de-interlacing", in *SPIE Proceedings Vis. Comm. and Image Process.*, pp. 805–810, 1991.

[37] P. Delogne, L. Cuvelier, B. Maison, B van Caillie, and L. Vandendorpe, "Improved interpolation, motion estimation and compensation for interlaced pictures", *IEEE Transactions on Image Processing*, vol. 3, no. 5, pp. 482–491, Sept. 1994.

[38] L. Vandendorpe, L. Cuvelier, B. Maison, P. Quelez, and P. Delogne, "Motion-compensated conversion from interlaced to progressive formats", *Elsevier Signal Processing: Image Communication*, vol. 6, pp. 193–211, 1994.

[39] G. de Haan, P. W. A. C. Biezen, and O. A. Ojo, "Patent: Apparatus for performing motion-compensated picture signal interpolation", US-patent 5,534,946.

[40] G. de Haan, P. W. A. C. Biezen, H. Huijgen, and O. A. Ojo, "Graceful degradation in motion compensated field-rate conversion", *Signal Processing of HDTV*, pp. 249–256, 1994.

[41] M. Hahn, P. Rieder, and C. Tuschen, "Patent: Motion-compensated frame interpolation", EU-patent 1,397,003, Mar. 2004.

[42] G. de Haan and A. Pelagotti, "Patent: Problem area location in an image signal", US-patent 6,487,313, Nov. 2002.

[43] O. A. Ojo and G. de Haan, "Robust motion-compensated video up-conversion", *IEEE Transactions on Consumer Electronics*, vol. 43, no. 4, pp. 1045–1055, Nov. 1997.

[44] Y. Okumura T. Kondo, T. Fujiwara and Y. Node, "Patent: Picture conversion apparatus, picture conversion method, learning apparatus and learning method", US-patent 6,323,905, Nov. 2001.

[45] M. Zhao and G. de Haan, "Intra-field de-interlacing with advanced up-scaling methods", in *IEEE International Symposium on Consumer Electronics*, pp. 315–319, 2004.

[46] M. A. Klompenhouwer, *Flat panel display signal processing*, Ph.D. thesis, University of Technology Eindhoven, 2006.

[47] R. M. Rast, "The dawn of digital TV", *IEEE Spectrum*, Oct. 2005.

[48] A. Navarro, "Technical Aspects of European Digital Terrestrial Television", in *IEEE MELECON*, pp. 2–6, 2002.

[49] H. Schroeder, M. Silverberg, B. Wendland, and G. Huerkamp, "Scanning modes of flicker-free colour TV-reproduction", *tce*, vol. 31, no. 4, pp. 627–641, 1985.

[50] A. M. Bock, "Motion-adaptive standards conversion between formats of similar field rates", *Signal Processing: Image Communication*, vol. 6, pp. 275–280, June 1994.

[51] P. D. Filliman, T. J. Christopher, and R. T. Keen, "Interlace to progressive scan converter for IDTV", *IEEE Transactions on Consumer Electronics*, vol. 38, no. 3, pp. 135–144, Aug. 1992.

[52] T. Doyle and M. Looymans, "Progressive scan conversion using edge information", *Signal Processing of HDTV*, vol. 2, pp. 711–721, 1990.

[53] Philips Semiconductors: Datasheet SAA4990 Prozonic, 1995.

[54] G. de Haan and R. Lodder, "De-interlacing of video data using motion vectors and edge information", in *IEEE International Conference on Consumer Electronics*, pp. 70–71, 2002.

[55] F. Wang, D. Anastassiou, and A. Netravali, "Time-recursive deinterlacing for idtv and pyramid coding", *Elsevier Signal Processing: Image Communication*, vol. 2, pp. 365–374, 1990.

[56] G. de Haan and G. F. M. de Poortere, "Patent: Method and apparatus for processing a picture signal to increase the number of displayed television lines using motion vector compensated values", US-patent 5,280,350, June 2006.

[57] G. de Haan and E. B. Bellers, "De-interlacing of video data", *IEEE Transactions on Consumer Electronics*, vol. 43, no. 3, pp. 819–825, Aug. 1997.

[58] J. L. Yen, "On Nonuniform Sampling of Bandwidth-Limited Signals", *IRE Transactions on Circuit Theory*, vol. CT-3, pp. 251–257, Dec. 1956.

[59] E. B. Bellers and G. de Haan, "Advanced motion estimation and motion compen-

sated de-interlacing", in *International Workshop on HDTV*, pp. Session A2, paper no. 3, 1996.

[60] A. Beric and R. Sethuraman, "Complexity analysis of the 2D-GST based de-interlacing algorithm, Philips Research Report, PR-TN 2004/00262", Oct. 2004.

[61] A. W. M. van den Enden and N. A. M. Verhoeckx, *Discrete-time signal processing*, Prentice Hall, 1989.

[62] P. Haavisto, J. Juhola, and Y. Neuvo, "Fractional frame rate up-conversion using weighted median filters", *IEEE Transactions on Consumer Electronics*, vol. 35, no. 3, pp. 272–278, Aug. 1989.

[63] P. Haavisto, J. Juhola, and Y. Neuvo, "Scan rate up-conversion using adaptive weighted median filtering", *Signal Processing of HDTV*, vol. 2, pp. 703–710, 1990.

[64] H. Blume, L Schworer, and K. Zygis, "Subband based upconversion using complementary median filters", in *Workshop on HDTV*, p. paper no. 1, 1994.

[65] O.L. Franzen, *Design and application of weighted median filters for poly-phase image interpolation*, Ph.D. thesis, Dortmund University (in German), 2005.

[66] M. Mertens and G. de Haan, "Motion vector field improvement for picture rate conversion with reduced halo", in *SPIE Proceedings Vis. Comm. and Image Process.*, pp. 352–362, 2001.

[67] M. M. van Ansem and R. Sethuraman, "Complexity analysis of the picture-rate up-conversion algorithm with reduced halo artifacts, Philips Research Report, PR-TN 2004/00253", Oct. 2004.

[68] D. P. Mitchell and A. N Netravali, "Reconstruction filters in computer graphics", *Computer graphics*, vol. 22, no. 4, pp. 221–228, Aug. 1988.

[69] P. Thevenaz, T. Blu, and M. Unser, *Handbook of medical imaging, processing and analysis, Chapter Image interpolation and resampling*, Academic press San Diego, CA, USA, ISBN 0120777908, 2000.

[70] Gennum: Product brief and the datasheet of the gf9320 scaling processor, 2004.

[71] C. B. Atkins, C. A. Bouman, and J.P. Allebach, "Optimal image scaling using pixel classification", in *IEEE International Conference on Image Processing*, pp. 864–867, 2001.

[72] N. Plaziac, "Image interpolation using neural networks", *IEEE Transactions on Image Processing*, vol. 8, no. 11, pp. 1647–1651, Nov. 1999.

[73] X. Li and M. T. Orchard, "New edge-directed interpolation", *IEEE Transactions on Image Processing*, vol. 10, no. 10, pp. 1521–1527, Oct. 2001.

[74] E. B. Bellers and J. Caussyn, "A high definition experience from standard definition video", in *SPIE Proceedings, the International Society for Optical Engineering*, pp. 594–603, 2003.

[75] J. Tegenbosch, P. Hofman, and M. Bosma, "Improving nonlinear up-scaling by adapting to the local edge orientation", in *SPIE Proceedings*, pp. 1181–1190, 2004.

[76] T. Kondo and K. Kawaguchi, "Patent: Adaptive dynamic range encoding method and apparatus", US-patent 5,444,487, Aug. 1995.

[77] H. Hu, P. M. Hofman, and G. de Haan, "Content-adaptive neural filters for image interpolation using pixel classification", in *Proceedings of SPIE, Applications of Neural Networks and Machine Learning in Image Processing IX*, 2005.

[78] H. Hu and G. de Haan, "Class-count reduction techniques for content adaptive filtering", in *To be published in Proceedings of the European Signal Processing Conference*, 2008.

[79] T. K. Moon, "The expectation-maximization algorithm", *IEEE Signal Processing Magazine*, vol. 13, no. 6, pp. 47–60, Nov. 1996.

[80] A. Lev, S. W. Zucker, and A. Rosenfeld, "Iterative enhancement of noisy images", *IEEE Transactions on Systems Man and Cybernetics*, vol. 7, pp. 435–442, June 1977.

[81] Y. L. Lee, H. C. Kim, and H. W. Park, "Blocking effect reduction of JPEG images by signal adaptive filtering", *IEEE Transactions on Image Processing*, vol. 7, no. 2, pp. 229–234, Feb. 1998.

[82] M. Zhao, R.E.J. Kneepkens, P.M. Hofman, and G. de Haan, "Content adaptive image de-blocking", in *IEEE International Symposium on Consumer Electronics*, pp. 299–304, 2004.

[83] B. Ramamurthi and A. Gersho, "Nonlinear space-variant postprocessing of block coded images", *IEEE International Conference on Acoustics, Speech and Signal Processing*, vol. 34, no. 5, pp. 1258–1268, Oct. 1986.

[84] H. Reeve and J. Lim, "Reduction of blocking effect in image coding", *IEEE International Conference on Acoustics, Speech and Signal Processing*, vol. 8, pp. 1212–1215, Apr. 1983.

[85] M. B. Alp and Y. Neuvo, "3-dimensional median filters for image sequence processing", in *IEEE International Conference on Acoustics, Speech and Signal Processing*, vol. 4, pp. 2917–2920, 1991.

[86] J. S. Lee, "Digital image smoothing and the sigma filter", in *Computer Vision Graphics and Image Processing*, pp. 255–269, 1983.

[87] A. K. Katsaggelos, J. N. Driessen, S. N. Efstratiadis, and R. L. Lagendijk, "Temporal motion compensated noise filtering of image sequences", in *SPIE Proceedings Vis. Comm. and Image Process.*, pp. 61–70, 1989.

[88] R. C. Triplicane, "Recursive restoration of noisy image sequences", M.S. thesis, Northwestern Univ., Evanston, IL, Oct. 1989.

[89] G. de Haan, "IC for motion compensated de-interlacing, noise reduction and picture rate conversion", *IEEE Transactions on Consumer Electronics*, vol. 42, pp. 617–624, Aug. 1999.

[90] V. Zlokolica, A. Pizurica, and W. Philips, "Wavelet-domain video denoising based on reliability measures", *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 16, pp. 993–1007, Aug. 2006.

[91] J. O. Drewery, R. Storey, and N. E. Tanton, "Video Noise Reduction, BBC Research Department Report, BBC RD 1984/7", July 1984.

[92] G. de Haan, T. G. Kwaaitaal-Spassova, and O. A. Ojo, "Automatic 2-D and 3-D noise filtering for high-quality television", in *International Workshop on HDTV*, pp. Session 4–B, paper no. 2, 1994.

[93] D. C. C. Wang, A. H. Vagnucci, , and C.C. Li, "Gradient inverse weighted smoothing scheme and the evaluation of its performance", in *Computer Vision Graphics and Image Processing*, pp. 167–181, 1981.

[94] J. C. Brailean, R. P. Kleihorst, S. Efstratiadis, A. K. Katsaggelos, and R. L. Lagendijk, "Noise reduction filters for dynamic image sequences", *Proceedings of the IEEE*, vol. 83, no. 9, pp. 1272–1292, Sept. 1995.

[95] Ed. T. S. Huang, *Image Sequence Analysis*, Berlin: Springer-Verlag, 1981.

[96] M. C. Q. Farias, J. M. Foley, and S. K. Mitra, "Detectability and annoyance of synthetic blockiness, blurriness, noisiness and ringing in video sequences", in *IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 553–556, 2005.

[97] Y. K. Lai, J. Li, and C. C. J Kuo, "Image enhancement for low bit-rate jpeg and mpeg coding via postprocessing", in *Proceedings of the SPIE*, pp. 1484–1494, 1996.

[98] C. Tomasi and R. Manduchi, "Bilateral Filtering for Gray and Color Image", in *IEEE International Conference on Computer Vision*, pp. 839–846, 1998.

[99] H. Hu and G. de Haan, "Simultaneous Coding Artifact Reduction and Sharpness Enhancement", in *IEEE International Conference on Consumer Electronics*, pp. 213–214, 2007.

[100] J. Chen, J. Liu, X. Wang, and G. Chen, "Modified edge-oriented spatial interpolation for consecutive blocks error concealment", in *IEEE International Conference on Image Processing*, pp. III – 904–907, 2005.

[101] J. Shu, "A new heuristic edge extraction technique", in *IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 285–288, 1987.

[102] K.T. Tan and M. Ghanbari, "Frequency domain measurement of blockiness in MPEG-2 coded video", in *IEEE International Conference on Image Processing*, pp. 977–980, 2000.

[103] A. Zakhor, "Iterative procedures for reduction of blocking effects in transform image coding", *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 2, no. 1, pp. 91–95, Mar. 1992.

[104] S. Yang, S. Kittitornkun, Y. H. Hu, T.Q. Nguyen, and D.L.Tull, "Blocking artifact free inverse discrete cosine transform", in *IEEE International Conference on Image Processing*, pp. 869–872, 2000.

[105] Y. Yang, N. P. Galatsanos, and A. K. Katsaggelos, "Regularized reconstruction to reduce blocking artifacts of block discrete cosine transform compressed images", *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 3, no. 6, pp. 421–432, Dec. 1993.

[106] H. Paek, R.-C. Kim, and S.-U. Lee, "On the POCS-based postprocessing technique

to reduce the blocking artifacts in transform coded images", *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 8, no. 3, pp. 358–367, June 1998.

[107] S. Minami and A. Zakhor, "An optimization approach for removing blocking effects in transform coding", *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 5, no. 2, pp. 74–82, Apr. 1995.

[108] W. B. Pennebaker and J. L. Mitchell, *JPEG Still Image Data Compression*, New York: Springer-Verlag, ISBN: 0442012721, 1993.

[109] C. Wang, W.-J. Zhang, and X.-Z. Fang, "Adaptive reduction of blocking artifacts in DCT domain for highly compressed images", *IEEE Transactions on Consumer Electronics*, vol. 50, no. 2, pp. 647–654, May 2004.

[110] T. Jarske, P. Haavisto, and I. Defe'e, "Post-filtering methods for reducing blocking effects from coded images", *IEEE Transactions on Consumer Electronics*, pp. 521–526, Aug. 1994.

[111] Y. Yang, N. P. Galatsanos, and A. K. Katsaggelos, "Projection-based spatially adaptive reconstruction of block-transform compressed images", *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 4, no. 7, pp. 896–908, July 1995.

[112] D. Youla, "Generalized Image Restoration by the Method of Alternating Orthogonal Projections", *IEEE Transactions on Circuits and Systems*, vol. 25, no. 9, pp. 694–702, Sept. 1978.

[113] P. Rieder and G. Scheffler, "New concepts on denoising and sharpening of video signals", *IEEE Transactions on Consumer Electronics*, vol. 47, no. 3, pp. 666–671, Aug. 2001.

[114] M. Schu, D. Wendel, C. Tuschen, M. Hahn, and U. Langenkamp, "System-on-silicon solution for high quality consumer video processing - the next generation", *IEEE Transactions on Consumer Electronics*, vol. 47, no. 3, pp. 412–419, Aug. 2001.

[115] P. Rieder and G. Scheffler, "New concepts on denoising and sharpening of video signals", *IEEE Transactions on Consumer Electronics*, vol. 47, no. 3, pp. 666–671, Aug. 2001.

[116] Philips Semiconductors: Datasheet tda9170 yuv picture improvement processor based on histogram modification, 1994.

[117] Philips Semiconductors: Datasheet tda4780 rgb video processor with automatic cut-off control and gamma adjust, 1997.

[118] Philips Semiconductors: Datasheet tdas480 rgb-gamma for lcd displays, 1997.

[119] Philips Semiconductors: Datasheet saa4978h Picture improved combined network IC (picnic), 1999.

[120] M. Braun, M. Hahn, J. R. Ohm, and M. Talmi, "Motion-compensating real-time format converter for video on multimedia displays", *IEEE Transactions on Image Processing*, vol. 1, pp. 125–128, Oct. 1997.

[121] M. Berekovic, H.-J. Stolberg, P. Pirsch, and H. Runge, "A programmable co-

processor for MPEG-4 video", in *IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 1021–1024, 2001.

[122] H.-J. Stolberg, M. Berekovic, L. Friebe, S. Moch, S. Flugel, M. Xun, M. B. Kulaczewski, H. Klussmann, and P. Pirsch, "HiBRID-SoC: a multi-core system-on-chip architecture for multimedia signal processing applications", in *Design, Automation and Test in Europe Conference*, pp. 8–13, 2003.

[123] S. Rathnam, G. Slavenburg, S. Nayak, E. Bellers, and J. Janssen, "A single-chip hybrid media processor for CRT and matrix displays-based televisions", in *IEEE International Conference on Consumer Electronics*, pp. 418–419, 2003.

[124] J.W. van der Waerdt, S. Vassiliadis, E. Bellers, and J.G.W.M. Janssen, "Motion Estimation and Temporal Up-Conversion on the TM3270 Media-Processor", in *IEEE International Conference on Consumer Electronics*, pp. 315–316, 2006.

[125] G. de Haan, P. W. A. C. Biezen, and O. A. Ojo, "An evolutionary architecture for motion-compensated 100 Hz television", *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 5, no. 3, pp. 207–217, June 1995.

[126] M. Schu, G. Scheffler, C. Tuschen, and A. Stolze, "System on silicon-IC for motion compensated scan rate conversion, picture-in-picture processing, split screen applications and display processing", *IEEE Transactions on Consumer Electronics*, vol. 45, no. 3, pp. 842–850, Aug. 1999.

[127] J. N. Driessen, L. Boroczki, and J. Biemond, "Pel-recursive motion field estimation from image sequences", *Journal on visual communication and image restoration*, 1991.

[128] R. J. Schutten and G. de Haan, "Real-time 2-3 pull-down elimination applying motion estimation/compensation on a programmable device", *IEEE Transactions on Consumer Electronics*, vol. 44, no. 3, pp. 930–938, Aug. 1998.

[129] G. de Haan and P. W. A. C. Biezen, "Sub-pixel motion estimation with 3-D recursive search block-matching", *Elsevier Signal Processing: Image Communication*, vol. 6, pp. 229–239, 1994.

[130] E. B. Bellers, J. G. Janssen, and M. Penners, "Efficient frame rate conversion for high frame rate lcd panels", in *International Workshop on Video Processing and Quality Metrics for Consumer Electronics VPQM-07*, 2007.

[131] T. Koga, K. Iinuma, A. Hirano, Y. Iilima, and T. Ishiguro, "Motion-compensated interframe coding for video conferencing", in *Proceedings of the NTC*, p. paper no. G5.3.1., 1981.

[132] Y. Ninomiya and Y. Ohtsuka, "A motion compensated interframe coding scheme for television pictures", *IEEE Transactions on communications*, vol. 30, no. 1, pp. 201–211, 1982.

[133] R. Srinivasan and K. Rao, "Predictive coding based on efficient motion estimation", *IEEE Transactions on Communications*, vol. 33, no. 8, pp. 888–896, Aug. 1985.

[134] G. de Haan, *Motion estimation and compensation, an integrated approach to con-*

sumer display field rate conversion, Ph.D. thesis, Delft University of Technology, 1992.

[135] T. Reuter, "Improved TV standards conversion with 3-dimensional motion compensating interpolation filter", in *Club de Rennes Young TV Researchers Conference*, 1988.

[136] M. Goetze, "Generation of motion vector fields for motion compensated interpolation of HDTV signals", in *Signal Processing of HDTV*, pp. 383–392, 1988.

[137] B. Chupeau, "Multi-resolution motion estimation", in *International Workshop on HDTV and Beyound*, pp. Session 5, paper no. 2, 1991.

[138] C. D. Kuglin and D. C. Hines, "The phase correlation image alignment method", in *IEEE International Conference on Cybernetics and Society*, pp. 163–165, 1975.

[139] G. A. Thomas, "Television motion measurement for datv and other applications, BBC Research Department Report, BBC RD 1987/11", Nov. 1987.

[140] F. Kelly, *Fast probabilistic inference and GPU video processing*, Ph.D. thesis, University of Dublin, 2005.

[141] A. Beric, G. de Haan, R. Sethuraman, and J. van Meerbergen, "Technique for reducing the complexity of the recursive motion estimation", in *IEEE International Workshop on Signal Processing Systems*, pp. 195–200, 2003.

[142] P. Csillag and L. Boroczky, "Frame rate conversion based on acceleration and motion-based segmentation", in *Proceedings of the SPIE*, pp. 438–448, 1996.

[143] J. C. Greiner, R. Sethuraman, J. van Meerbergen, and G. de Haan, "A cost-effective implementation of object-based motion estimation", in *IEEE International Workshop on Signal Processing Systems*, pp. 148–153, 2003.

[144] K. Denolf, C. de Vleeschouwer, R. Turney, G. Lafruit, and J. Bormans, "Memory centric design of an mpeg-4 video encoder", *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 15, no. 5, pp. 609–619, May 2005.

[145] P. R. Panda, N. D. Dutt, A. Nicolau, F. Catthoor, A. Vandecappelle, E. Brockmeyer, C. Kulkarni, and E. de Greef, "Data memory organization and optimizations in application-specific systems", *"Design and Test of Computers, IEEE"*, vol. 18, no. 3, pp. 56–68, 2001.

[146] C. Kulkarni, F. Catthoor, and H. D. Man, "Advanced data layout organization for multimedia applications", in *Proceedings of the Workshop on Parallel and Distributed Computing in Image Processing, Video Processing, and Multimedia*, pp. 186–193, 2000.

[147] J-W. van der Waerdt, G. A. Slavenburg, J. P. van Itegem, and S. Vassiliadis, "Motion Estimation Performance of the TM3270 Processor", in *ACM symposium on Applied Computing*, pp. 850–856, 2005.

[148] A. Beric, R. Sethuraman, H. Peters, G. Veldman, J. van Meerbergen, and G. de Haan, "Streaming scratchpad memory organization for video applications", in *IAESTED International Conference on Circuits, Signals and Systems*, pp. 427–432, 2004.

[149] P. R. Panda, F. Catthoor, N. D. Dutt, K. Danckaert, E. Brockmeyer, C. Kulkarni, A. Vandecappelle, and P. G. Kjeldsberg, "Data and memory optimization techniques for embedded systems", *ACM Transactions on Design Automation of Electronic Systems*, vol. 6, no. 2, pp. 149–206, 2001.

[150] Texas Instruments: http://www.ti.com.

[151] P. Ranganathan, S. Adve, and N. P. Jouppi, "Reconfigurable caches and their application to media processing", in *IEEE International Symposium on High-Performance Computer Architecture*, pp. 214–224, 2000.

[152] N. Park and N. K. Prasanna, "Reconfigurable caches and their application to media processing", in *IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 1205–1208, 2001.

[153] T. F. Chen and J. L. Baer, "A performance study of software and hardware data prefetching schemes", in *IEEE International Symposium on Computer Architecture*, pp. 223–232, 1994.

[154] F. Dahlgren and P. Stenstrom, "Effectiveness of hardware-based stride and sequential prefetching in shared-memory multiprocessors", in *IEEE International Symposium on High-Performance Computer Architecture*, pp. 68–77, 1995.

[155] D.F. Zucker, R. B. Lee, and M. J. Flynn, "Hardware and software cache prefetching techniques for MPEG benchmarks", *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 10, no. 5, pp. 782–796, July 2000.

[156] R. Cucchiara, M. Piccardi, and A. Prati, "Neighbor cache prefetching for multimedia image and video processing", *IEEE Transactions on Multimedia*, vol. 6, no. 4, pp. 539–552, Aug. 2004.

[157] H. Sbeyti, S. Niar, and L. Eeckhout, "Adaptive prefetching for multimedia applications in embedded systems", in *Design, Automation and Test in Europe Conference*, pp. 1350–1351, 2004.

[158] P. Ranganathan, S. Adve, and N. P. Jouppi, "Performance of image and video processing with general-purpose processors and media ISA extensions", in *IEEE International Symposium on Computer Architecture*, pp. 124–135, 1999.

[159] Ed. R. Cravotte, "Exploring Memory Architectures: Pillars of processing performance", *EDN*, pp. 44–54, June 2007.

[160] C. Kulkarni, C. Ghez, M. Miranda, F. Catthoor, and H. de Man, "Cache conscious data layout organization for embedded multimedia applications", in *Design, Automation and Test in Europe Conference*, pp. 686–691, 2001.

[161] P. R. Panda, N. D. Dutt, and A. Nicolau, "Efficient Utilization of Scratch-Pad Memory in Embedded Processor Applications", in *Proceedings of the IEEE European conference on Design and Test*, 1997.

[162] W. Hinrichs, J.P. Wittenburg, H. Lieske, H. Kloos, M. Ohmacht, and P. Pirsch, "A 1.3-GOPS parallel DSP for high-performance image-processing applications", *IEEE Journal of Solid-State Circuits*, vol. 35, no. 7, pp. 946–952, July 2000.

[163] A. Beric, R. Sethuraman, J. van Meerbergen, and G. de Haan, "Algo-

rithm/Architecture co-design of a picture-rate up-conversion module", in *Proceedings of ProRISC/IEEE conference*, pp. 203–208, 2002.

[164] A. Beric, G. de Haan, R. Sethuraman, and J. van Meerbergen, "An efficient picture-rate up-converter", *Journal of VLSI Signal Processing*, vol. 41, no. 1, pp. 49–63, Aug. 2005.

[165] J.-C. Tuan, T.-S. Chang, and C.-W. Jen, "On the Data Reuse and Memory Bandwidth Analysis for Full-Search Block-Matching VLSI Architecture", *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 12, no. 1, pp. 61–72, Jan. 2002.

[166] S. Yang, W. Wolf, and N. Vijaykrishnan, "Power and performance analysis of motion estimation based on hardware and software realizations", *IEEE Transactions on Computers*, pp. 714–726, June 2005.

[167] P. Budnik and D. J. Kuck, "The organization and use of parallel memories", *IEEE Transactions on Computer*, vol. 20, no. 12, pp. 1566–1569, Dec. 1971.

[168] D. H. Lawrie, "Access and alignment of data in an array processor", *IEEE Transactions on Computer*, vol. 24, no. 12, pp. 1145–1155, Dec. 1975.

[169] D. H. Lawrie and C. R. Vora, "The prime memory system for array accesst", *IEEE Transactions on Computer*, vol. 31, no. 5, pp. 1145–1155, May 1982.

[170] J. W. Park, "An Efficient Buffer Memory System for Subarray Access", *IEEE Transactions on parallel and distributed systems*, vol. 12, no. 3, pp. 316–335, Mar. 2001.

[171] AMBA AHB bus http://www.arm.com/products/solutions/AMBAHomePage.html.

[172] G. Berry, L. Charpentier, and M.Duranton, "Specification document for Philips' CAT-IP DMA, Esterel technologies", May 2005.

[173] A. Beric and M. Cocco, "Silicon Hive DMA, Implementation document", June 2007.

[174] NXP memory modules.

[175] A. Beric, G. de Haan, J. van Meerbergen, and R. Sethuraman, "Towards an efficient high quality picture-rate up-converter", in *IEEE International Conference on Image Processing*, pp. 363–366, 2003.

[176] A. Beric, B. van der Waal, R. Sethuraman, and G. de Haan, "Low-bandwidth dynamic aspect ratio region-based motion estimation", in *IEEE International Workshop on Signal Processing Systems*, pp. 393–398, 2005.

[177] "Fork in the road for Moore's law", *IEEE Electronics Systems & Software*, vol. 2, no. 5, pp. 6–7, Oct. 2004.

[178] N. Forbes and M. Foster, "The End of Moore's Law?", *IEEE Computing in Science and Engineering*, vol. 5, no. 1, pp. 18–19, Jan. 2003.

Publications

Journal publications

1. A. Beric, J. van Meerbergen, G. de Haan and R. Sethuraman, "Memory-Centric Video Processing", in *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 18 no. 4, April 2008.
2. A. Beric, G. de Haan, R. Sethuraman and J. van Meerbergen, "An efficient picture-rate up-converter", in *Journal of VLSI Signal Processing*, vol. 41 no. 1, pp. 49-63, Aug. 2005.
3. H. Peters, R. Sethuraman, A. Beric, P. Meuwissen, S. Balakrishnan, C. A. Pinto, W. Kruijtzer, F. Ernst, G. Alkadi, J. van Meerbergen and G. de Haan, "Application Specific Instruction-Set Processor Template for Motion Estimation in Video Applications", in *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 15, no. 4, pp. 508-527, April 2005.

Conference publications

1. A. Beric, R. Sethuraman, C. A. Pinto, H. Peters, G. Veldman, P. van de Haar and M. Duranton, "Heterogeneous Multiprocessor for High Definition Video", in *IEEE International Conference on Consumer Electronics*, pp. 401-402, 2006.
2. A. Beric, G. de Haan, R. Sethuraman and J. van Meerbergen, "Algorithm/architecture co-design of the generalized sampling theorem based de-interlacer", in *IEEE International Symposium on Circuits and Systems*, pp. 2943-2946, 2005.
3. A. Beric, B. van der Waal, R. Sethuraman and G. de Haan, "Low-bandwidth dynamic aspect ratio region-based motion estimation", in *IEEE International Workshop on Signal Processing Systems*, pp. 393-398, 2005.
4. A. Beric, R. Sethuraman, J. van Meerbergen and G. de Haan, "Memory-centric motion estimator", in *IEEE International Conference on VLSI Design*, pp. 816-819, 2005.

5. A. Beric, R. Sethuraman, H. Peters, G. Veldman, J. van Meerbergen and G. de Haan, "Streaming scratchpad memory organization for video application", in *IAESTED International Conference on Circuits, Signals and Systems*, pp. 427-432, 2004.
6. A. Beric, R. Sethuraman, H. Peters, J. van Meerbergen, G. de Haan and C. Alba Pinto, "A 27mw 1.1 mm^2 motion estimator for picture-rate up-converter", in *IEEE International Conference on VLSI Design*, pp. 1083-1088, 2004.
7. A. Beric, G. de Haan, J. van Meerbergen and R. Sethuraman, "Towards an efficient high quality picture-rate up-converter", in *IEEE International Conference on Image Processing*, pp. 363-366, 2003.
8. A. Beric, G. de Haan, R. Sethuraman and J. van Meerbergen, "Technique for reducing the complexity of the recursive motion estimation", in *IEEE International Workshop on Signal Processing Systems*, pp. 195-200, 2003.
9. A. Beric, G. de Haan, J. van Meerbergen and R. Sethuraman, "Parameters analysis of a picture-rate up-conversion module", in *ASCI conference*, pp. 246-251, 2003.
10. A. Beric, R. Sethuraman, J. van Meerbergen and G. de Haan, "Algorithm/Architecture co-design of a picture-rate up-conversion module", in *ProRISC/IEEE conference*, pp. 203-208, 2002.

Patent applications

1. WO 2007/113767: A. Beric and R. Sethuraman, "Video processor comprising a pixel filter", October, 2007.
2. WO 2007/091213: A. Beric and R. Sethuraman, "Video processor comprising a motion estimator of the recursive type", August, 2007.
3. WO 2007/052203: A. Beric and R. Sethuraman, "Data processing system", May, 2007.
4. WO 2006/109209: A. Beric and R. Sethuraman, "Video processing with region-based multiple-pass motion estimation and update of temporal motion vector candidates", October, 2006.
5. WO 2006/109205: A. Beric and R. Sethuraman, "Region-based 3DRS motion estimation using dynamic aspect ratio of region", October, 2006.
6. EP1864492A1: A. Beric and R. Sethuraman, "Processing a data array with a meandering scanning order using a circular buffer memory", September, 2006.
7. EP1817907A1: A. Beric and R. Sethuraman, "Reducing the latency of a motion estimation based video processing system", June, 2006.
8. EP1741296A2: R. Sethuraman, A. Beric, C. A. Pinto, H. Peters, P. Meuwissen, B. Srinivasan and G. Veldman, "Data processing apparatus that provides parallel access to multi-dimensional array of data values", November, 2005.

Biography

Aleksandar Berić was born on August the 13th, 1976 in Belgrade, Serbia. He has enrolled the University of Belgrade, School of Electrical Engineering in 1995 and completed his studies in 2001. The graduation project was a design of the digital motor control system based on a Texas Instruments DSP. During his studies, in 1999, he spent half a year at Tokyo Metropolitan Institute of Technology, Japan where he worked on research projects concerning signal processing and emotional engineering. In 2002, he started his PhD studies at the Eindhoven University of Technology, The Netherlands, in collaboration with Philips Research Eindhoven. The topic was architectural research within the video post processing application domain. This research resulted in this book, 3 journal and 10 conference papers, and 8 patent applications. The results from this work are currently being productized by Silicon Hive.