Parallel Video Decoding

Mauricio Álvarez Mesa

A thesis submitted in fulfillment of the requirements for the degree of Doctor of Philosophy in Computing Engineering

Advisors:
Alex Ramírez and Mateo Valero

Department of Computer Architecture
Universitat Politècnica de Catalunya (UPC)
June, 2011

To Luna, Claudia, Carolina and Mariela.
The women of my life.
Without their support and love this work would have been meaningless.

Abstract

Digital video is a popular technology used in many different applications. The quality of video, expressed in the spatial and temporal resolution, has been increasing continuously in the last years. In order to reduce the bitrate required for its storage and transmission, a new generation of video encoders and decoders (codecs) have been developed. The latest video codec standard, known as H.264/AVC, includes sophisticated compression tools that require more computing resources than any previous video codec. The combination of high quality video and the advanced compression tools found in H.264/AVC has resulted in a significant increase in the computational requirements of video decoding applications.

The main objective of this thesis is to provide the performance required for real-time operation of high quality video decoding using programmable architectures. Our solution has been the simultaneous exploitation of multiple levels of parallelism. On the one hand, video decoders have been modified in order to extract as much parallelism as possible. And, on the other hand, general purpose architectures has been enhanced for exploiting the type of parallelism that is present in video codec applications.

First, we made a scalability analysis of two different Single Instruction Multiple Data (SIMD) extensions: a 1-dimensional (1D) extension and a 2-d matrix extension (2D). We have shown that scaling the 2D extension results in higher performance and lower complexity than the 1D extension for MPEG-2 video coding and decoding.

We performed a workload characterization of H.264/AVC for High Definition (HD) applications. We identified the main kernels and compared them with the kernels of previous video codecs. Due to the lack of a proper benchmark for HD video decoding we developed our own one, called HD-VideoBench. This benchmark includes complete applications for video coding and decoding together with a set in of input videos in HD resolutions.

After that, we optimized the most relevant kernels of the H.264/AVC decoder using SIMD instructions. However, we were not able to reach the maximum performance due to the overhead of data re-alignment. As a solution, we implemented and evaluated the required hardware and software for supporting unaligned accesses in SIMD extensions. This support resulted in significant performance gains both for kernels and the complete application.

Because SIMD extensions were not enough to provide all the required performance, we developed an investigation on how to extract Thread-Level-Parallelism. We found that none of the existing mechanisms could scale to massive parallel systems. As a solution, we developed a new algorithm, called the dynamic 3D-Wave, that is able to reveal thousands of independent tasks exploiting macroblock-level parallelism.

We implemented intra-frame macroblock-level parallelism on a Distributed Shared Memory (DSM) parallel machine but this implementation was not able to reach the maximum performance due to the negative impact of thread synchronization and the effect of the entropy decoding kernel.

In order to eliminate these bottlenecks we proposed a parallelization of the entropy decoding stage at the frame-level combined with a parallelization of the other kernels at the macroblock-level. A further performance increase was obtained by using different type of processors for each type of kernel. The overhead of thread synchronization was almost eliminated with a special hardware support for synchronization operations.

With all the presented enhancements we were able to process, in real-time, video decoding applications at high definition and at high frame-rate. We created a scalable solution that is able to use the growing number of cores in emerging multicore architectures.

Acknowledgments

This thesis has been the result of a long effort during many years of my life. During that time I received the support of many people in many different ways.

First, I would like to express my gratitude to my thesis advisors Mateo Valero and Alex Ramírez. I want to thank Mateo for giving me the opportunity of making the PhD at the High Performance Computing Group at UPC under his direction. You also suggested me the idea of working on architectures for multimedia applications and, specifically, the idea of doing my research on high definition video processing using H.264/AVC. This field now constitutes my main area of interest and, it seems that, I am going to continue working on that for some time. I only regret for not being able to have more meetings like those of the first period of the doctorate. Also, I want to express my gratitude to Alex Ramírez for all his efforts and dedication during these years. You suggested me to cooperate with other research groups that were working on similar topics. These collaborations (with researchers from TU-Delft, NXP and others) gave me the opportunity to work in an international collaborative environment. Some of the main contributions of this thesis have been the result of these collaborations.

I am also thankful to Esther Salamí for working with me during the first years of the PhD. I learned a lot working with you, especially in those practical issues that are difficult on the day-by-day research. Also, I want to recognize your words of support which helped me a lot when things were not clear.

I am specially grateful to my friend and colleague Friman Sánchez. During these years we have supported each other and we have forged a great friendship. Apart from the technical discussions I have enjoyed very much the shared readings and conversations about literature, philosophy, politics and history. They helped me to understand what are the important things in life.

I thank Cor Meenderinck and Arnaldo Azevedo from TU-Delft for the collaboration that we developed about parallelization of video codecs. I learned a lot from you and enjoyed our meetings at different places of Europe.

I am specially grateful with Ben Juurlink (now in TU-Berlin) for the fruitful collaboration we have had over the years about parallel H.264 decoding. I thank your welcome in Berlin during the spring of 2011 and your dedication and help with my thesis. I also want to thank Chi Ching Chi for the good time working together on HEVC parallelization during my visit to TU-Berlin.

I thank Ayal Zaks and Uzi Shvadron for helping me during my visit to IBM Haifa Labs. It was a really good experience to work with you during these months. I am very grateful for your kind welcome in Israel.

I also want to express my gratitude to my colleagues of the PhD program at DAC UPC. Stefan Bieschewski, Nikolaos Galanis, Miquel Pericàs, Germán Rodríguez and others. My experience doing the PhD and living in Barcelona would not have been the same without our shared moments in the C6 room and other places.

I thank Felipe Cabarcas and Alejandro Rico for their support with the TaskSim

simulator. A part of this thesis has been possible thanks to your work.

I also want to thank the people from System Administration at DAC (LCAC) for their professionalism, responsiveness and for their commitment to free-software. Also, thanks to the administration people at DAC, specially Trinidad Carneros. Your work made my life easier during my PhD.

I also want to express my gratitude to Pau Bofill for reading and commenting on the first versions of this manuscript. Also, and specially, for sharing with me his humanistic perspective of engineering and scientific research.

I would like to show my gratitude to Juan Felipe Osorio, Clara Osorio and Carolina Mora for being my family during some years in Barcelona. Felipe has been a good friend since we were studying Calculus and Physics at the Universidad de Antioquia during our Bachelor. My deepest feelings of gratitude for him.

Also, I want to express my gratitude to Pierre Leroux, Gabriela Sobel, Jordi Vilar, Leonardo Boccaccio and other people from the Dojo Zen de Barcelona. Most of the strength and peace required to finish this thesis came from the time we spend together practicing zazen. In a similar way, I would like to thank Carlos Candre from Leticia, Amazonas (Colombia). My visit to your house in the jungle in the summer of 2010 gave me strength and courage to finish this project.

Finally, I want to express my gratitude to my family. To my daughter Luna. During all this time in Barcelona you has been my source of inspiration. I hope that we can enjoy more time together after my graduation. To my sister Carolina. Your intelligence and hard work have been an important motivation in my life. To my mother Mariela for her unconditional love. I want to acknowledge you specially for this afternoon where I was trying to make a difficult homework for the school about factorization. You learned algebra just to teach me how to solve these problems. But what you taught me is that I can solve big problems, and that with love everything can be learned. This thesis is the demonstration of that. Finally, to my wife and partner Claudia. For all the love and patience you have had with me. I'm very happy that we can continue making our lives together. Also, special thanks to you for the thesis cover design. I learned from you to appreciate the beauty of butterflies and with them to admire the beauty of all life.

You may ask—you are bound to ask me now: what, then, is in your opinion the value of natural science? I answer: It's scope, aim and value is the same as that of any other branch of human knowledge. Nay, none of them alone, only the union of all of them, has any scope or value at all, and that is simply enough described: it is to obey the command of the Delphic deity, get to know yourself. Or, to put it in the brief, impressive rhetoric of Plotinus, "And we, who are we anyhow?" he continues: "Perhaps we were there already before this creation came into existence, human beings of another type, or even some sort of gods, pure souls and mind united with the whole universe, parts of the intelligible world, not separated and cut off, but at one with the whole."
Erwin Schrödinger. Science and Humanism.

Contents

1 Introduction **1**
- 1.1 Trends in Multimedia Applications . . . 2
- 1.2 Trends in Computer Architecture . . . 3
  - 1.2.1 Scalability limits of single core architectures . . . 3
  - 1.2.2 The trend to multicore architectures . . . 5
- 1.3 Definition of the problem . . . 6
- 1.4 Thesis contributions . . . 7
  - 1.4.1 A benchmark for HD video applications . . . 7
  - 1.4.2 Scalability of multidimensional vector architectures . . . 7
  - 1.4.3 Efficiency of SIMD extensions for exploiting DLP . . . 7
  - 1.4.4 Thread-level parallelization of video decoding . . . 7
  - 1.4.5 Scalability of Macroblock-level parallelization . . . 8
  - 1.4.6 Scalability of heterogeneous multicore architectures . . . 8
- 1.5 Historical Context . . . 8
- 1.6 Organization of this document . . . 10

2 Video Coding Technology **13**
- 2.1 Video Compression Objectives . . . 13
- 2.2 Video Coding Standards . . . 13
- 2.3 Block-based Hybrid Generic Video Codec . . . 15
  - 2.3.1 Block-based Structure and Color Coding . . . 16
  - 2.3.2 Prediction . . . 16
  - 2.3.3 Transform . . . 17
  - 2.3.4 Entropy Coding . . . 18
  - 2.3.5 Decoding Process . . . 19
- 2.4 H.264/AVC Video codec . . . 19
  - 2.4.1 Entropy Decoding: CAVLC and CABAC . . . 20
  - 2.4.2 Inverse Transform . . . 21
  - 2.4.3 Quantization . . . 22
  - 2.4.4 Inter-prediction . . . 23
  - 2.4.5 Intra-prediction . . . 26
  - 2.4.6 In-loop Deblocking Filter . . . 28
  - 2.4.7 Comparison with Previous Video codecs . . . 30
- 2.5 Characteristics of Video Decoding Applications . . . 31
  - 2.5.1 Real-time Operation . . . 32
  - 2.5.2 Integer Small Data Types with Saturating Arithmetic . . . 32
  - 2.5.3 Block Processing . . . 32
  - 2.5.4 Heterogeneous Kernels . . . 32
  - 2.5.5 Hierarchy of Data Dependencies . . . 33

2.6 Summary . . . 33

3 Architectures for Video Decoding **35**
3.1 Dedicated Hardware Architectures . . . 35
3.2 Multimedia Processors . . . 37
3.3 General Purpose Processors (GPPs) . . . 38
3.3.1 SIMD Extensions . . . 38
3.3.2 Vector Processors . . . 42
3.4 Chip Multiprocessor (CMP) Architectures . . . 44
3.4.1 General Purpose Multicores . . . 44
3.4.2 Heterogeneous Media-processors . . . 45
3.4.3 Graphics Processing Units: GPUs . . . 46
3.4.4 Specialized Data-intensive Multicores . . . 48
3.5 Summary . . . 49

4 Scalability of Vector ISAs **51**
4.1 Scaling SIMD Extensions . . . 51
4.1.1 Scaling 1-Dimensional SIMD Extensions . . . 52
4.1.2 Scaling 2-Dimensional Extensions . . . 52
4.1.3 Hardware Cost of Scaling . . . 54
4.1.4 A Case of Study: Motion Estimation . . . 55
4.2 Experimental Methodology . . . 55
4.2.1 Workload . . . 55
4.2.2 Simulation Framework . . . 56
4.2.3 Processor Models . . . 57
4.2.4 Memory Hierarchy Model . . . 58
4.3 Simulation results . . . 59
4.3.1 Kernels Speedup . . . 59
4.3.2 Complete Applications Speedup . . . 60
4.3.3 Cycle Breakdown . . . 61
4.3.4 Dynamic Instruction Count . . . 61
4.4 Analysis of New SIMD Extensions . . . 62
4.5 Summary . . . 63

5 Workload Characterization **65**
5.1 Related Work . . . 65
5.2 Methodology . . . 66
5.2.1 Processor and Tools . . . 66
5.2.2 Codec Configuration . . . 67
5.2.3 Test Sequences . . . 67
5.3 Analysis . . . 68
5.3.1 Profiling of the H.264/AVC Decoders . . . 68
5.3.2 Instructions and Cycles . . . 69
5.3.3 Cache Analysis . . . 72
5.3.4 Branch Prediction . . . 74
5.4 Performance on Recent High Performance Processors . . . 75
5.5 Summary . . . 76

6 A Benchmark for HD Video Applications **77**
6.1 Benchmarking Video Codecs . . . 77
6.2 Related Work . . . 78
6.3 The HD-VideoBench Applications . . . 79
6.3.1 MPEG-2 . . . 80
6.3.2 MPEG-4 . . . 80
6.3.3 H.264/AVC . . . 81
6.4 HD-VideoBench Input Sequences and Coding Options . . . 81
6.5 Running HD-VideoBench . . . 82
6.6 HD-VideoBench Performance . . . 82
6.6.1 Coding Efficiency . . . 82
6.6.2 Decoding Performance: Frame Rate . . . 84
6.7 Summary . . . 86

7 Unaligned Accesses in SIMD architectures **89**
7.1 Motivation: Impact of Overhead Instructions . . . 89
7.2 Current Support for Unaligned Accesses . . . 90
7.2.1 Compiler Optimizations Related to Memory Alignment . . . 94
7.2.2 Unaligned Accesses in Video Applications . . . 94
7.3 Adding Support for Unaligned Loads and Stores . . . 95
7.4 Methodology . . . 97
7.5 Performance Evaluation . . . 99
7.5.1 Dynamic Instruction Count . . . 100
7.5.2 Kernels Speedup . . . 100
7.5.3 Impact of the Latency of Unaligned Load and Stores . . . 101
7.5.4 Complete Application Speedup . . . 103
7.6 Summary . . . 103

8 Thread-level Parallelism in Video Decoding **105**
8.1 Function-level Decomposition . . . 105
8.2 Data-level Decomposition . . . 107
8.2.1 GOP-level Parallelism . . . 107
8.2.2 Frame-level Parallelism for Independent Frames . . . 107
8.2.3 Slice-level Parallelism . . . 107
8.2.4 Macroblock-level Parallelism . . . 109
8.2.5 Block-level Parallelism . . . 113
8.3 Parallel Scalability of the Static 3D-Wave . . . 114
8.3.1 Estimating the Maximum Parallelism . . . 115
8.3.2 Theoretical Results . . . 116
8.4 Parallel Scalability of the Dynamic 3D-Wave . . . 117
8.4.1 Maximum Parallelism . . . 118
8.4.2 Effect of Limited Resources . . . 119
8.5 Related Work . . . 120
8.5.1 Function-level Parallelism . . . 120
8.5.2 GOP-level Parallelism . . . 120
8.5.3 Frame-level Parallelism . . . 121
8.5.4 Slice-level Parallelism . . . 121

8.5.5 Macroblock-level Parallelism . . . 121
8.6 Summary . . . 123

9 Scalability of Macroblock-level Parallelism 125
9.1 Theoretical Analysis . . . 125
9.1.1 Formal Model . . . 125
9.1.2 Abstract Trace-driven Simulation . . . 126
9.1.3 Effects of Variable Decoding Time . . . 127
9.1.4 Effects of Thread Synchronization Overhead . . . 128
9.2 Performance Analysis on a Parallel Architecture . . . 129
9.2.1 Implementation Methodology . . . 129
9.2.2 Evaluation Platform . . . 130
9.2.3 Scheduling Strategies . . . 131
9.2.4 Static Scheduling . . . 132
9.2.5 Dynamic Scheduling . . . 132
9.2.6 Dynamic Scheduling with Tail-submit . . . 133
9.2.7 Impact of the Sequential Part of the Application . . . 135
9.3 Summary . . . 136

10 Scalability of Heterogeneous Architectures 137
10.1 Scalable H.264/AVC Parallelization . . . 137
10.2 Solving the Scalability Bottlenecks . . . 138
10.2.1 Parallelism in the Entropy Decoding Stage . . . 138
10.2.2 Thread Synchronization . . . 139
10.3 Experimental Methodology . . . 140
10.3.1 H.264/AVC Decoder . . . 140
10.3.2 Instrumentation and Trace generation . . . 140
10.3.3 Trace-driven Simulation . . . 141
10.3.4 The SARC Architecture . . . 141
10.4 Experimental Results . . . 142
10.4.1 Dynamic 3D-Wave with Multiple CABAC Processors . . . 143
10.4.2 Case Study: Heterogeneous Manycore Architecture . . . 144
10.4.3 Impact of Thread Synchronization . . . 145
10.4.4 Memory Requirements . . . 145
10.5 Related Work . . . 147
10.6 Summary . . . 148

11 Conclusions 149
11.1 Contributions . . . 149
11.1.1 Scalability of Multidimensional Vector Architectures . . . 149
11.1.2 A Benchmark for High Definition Video Codec Applications . . . 150
11.1.3 Efficiency of SIMD extensions for Exploiting DLP . . . 151
11.1.4 Thread-level Parallelization of Video Decoding . . . 151
11.1.5 Scalability of Macroblock-level Parallelism . . . 152
11.1.6 Scalability of Heterogeneous Manycore Architectures . . . 153
11.1.7 Other Publications . . . 153
11.2 Open Areas for Research . . . 154

11.2.1 Modifications to Video Codecs 154
11.2.2 Modifications to the Architecture 155

Bibliography **157**

List of Figures

1.1 Architecture of a generic multimedia player 1
1.2 Moore's law 4
1.3 Relationship between different "walls" in processor design 5
1.4 Technological evolution during the thesis 9

2.1 Evolution of video codecs 14
2.2 General structure of a hybrid video codec 15
2.3 Data elements of a video sequence 17
2.4 Type of frames 18
2.5 General structure of the H.264/AVC decoder 19
2.6 CABAC arithmetic codec 20
2.7 Variable block size for motion compensation 23
2.8 Interpolation for the luma component 24
2.9 Half-pixel interpolation for chroma components 25
2.10 Motion vector prediction 26
2.11 Intra-prediction sample labels 27
2.12 Intra-prediction modes for 4×4 blocks 27
2.13 Data dependencies due to intra-prediction 28
2.14 Deblocking filter 29
2.15 Macroblock dependencies due to the deblocking filter 30

3.1 Hardware H.264 decoder 36
3.2 SIMD code example 40
3.3 Sample Operation using μSIMD instructions 40
3.4 Sample DLP operation using vector instructions 43
3.5 Example of DLP operation using a matrix instruction 43
3.6 OpenCL generic view of a GPU architecture 47

4.1 Register files for SIMD architectures 53
4.2 Motion estimation code example 56
4.3 Simulated microarchitecture 57
4.4 Kernels speedup 59
4.5 Full applications speedup 60
4.6 Cycle count distribution 61
4.7 Dynamic instruction count 62

5.1 Profiling of H.264 decoder 68
5.2 Performance impact of the different types of frames 70
5.3 IPC in P- and B-frames for 1008_blue_sky 71
5.4 IPC sampling 71

5.5 L1 data cache impact of frame type . . . 73
5.6 L1 data cache performance in P and B frames . . . 74
5.7 L1 data cache sampling . . . 74
5.8 Branch misprediction sampling . . . 75

6.1 Rate distortion for 1088p25_blue_sky . . . 84
6.2 Decoding performance . . . 85

7.1 Total dynamic instructions . . . 90
7.2 Altivec dynamic instructions . . . 90
7.3 Vector load from an unaligned address . . . 91
7.4 Altivec alignment code for a vector load . . . 92
7.5 Vector load from an unaligned address with stride one . . . 94
7.6 Alignment in luma and chroma interpolation kernels . . . 95
7.7 Altivec alignment code for a vector store . . . 96
7.8 Load store pipeline . . . 96
7.9 Realignment unit using a two-bank interleaved cache . . . 97
7.10 General microarchitecture of the simulated processors . . . 99
7.11 Kernels speedup . . . 101
7.12 Latency impact of unaligned accesses . . . 102
7.13 Complete application speedup . . . 103

8.1 Parallelization strategies . . . 106
8.2 Bitrate increase due to slices . . . 108
8.3 Data dependencies at the macroblock level . . . 109
8.4 MB parallelism with 2D-Wave approach . . . 111
8.5 MB-level parallelism in the temporal domain . . . 112
8.6 Parallel scalability of X264 encoder . . . 113
8.7 3D-Wave strategy . . . 114
8.8 Reference area in the static 3D-Wave . . . 115
8.9 Parallel MBs for static 3D-Wave . . . 116
8.10 Parallel MBs for dynamic 3D-Wave . . . 118
8.11 Dynamic 3D-Wave with limited frames in flight . . . 119

9.1 Directed Acyclic Graph (DAG) of macroblocks . . . 126
9.2 Histograms of MB processing time . . . 127
9.3 Sppedup per frame with variable decoding time . . . 128
9.4 Synchronization overhead . . . 129
9.5 Dynamic task model diagram . . . 129
9.6 cc-NUMA architecture . . . 130
9.7 Speedup for different scheduling approaches . . . 132
9.8 Ready to process MBs with 2D-Wave . . . 134
9.9 Profiling of enqueue and dequeue operations . . . 135
9.10 Complete application speedup . . . 136

10.1 H264 decoder with CABAC decoupling . . . 138
10.2 CABAC frame-level parallelism . . . 139
10.3 Baseline heterogeneous multicore architecture . . . 142

10.4 Multiple CABAC processors in the 3D-Wave 143
10.5 Parallel H.264 decoder with different type of cores 145
10.6 Latency effect of synchronization operations 146
10.7 Memory requirements 146

List of Tables

1.1 Bitrate requirements for video decoding 2

2.1 Comparison of video coding standards 31

3.1 Comparison of hardware architectures 36
3.2 Comparison of VLIW-based media-processors 37
3.3 Comparison of SIMD Extensions 39
3.4 Comparison of SIMD optimizations for video applications 41
3.5 H.264 decoding using SIMD 41
3.6 General purpose multicore architectures 45
3.7 Heterogeneous media-processors 46
3.8 Graphic Processing Units (GPUs) 47
3.9 Specialized data intensive multicores 48

4.1 Scaling register files for SIMD extensions 54
4.2 Benchmark set description 57
4.3 Modeled processors 58
4.4 Memory hierarchy configuration 59

5.1 Experimentation platform 67
5.2 Cycles, instructions and IPC per frame 70
5.3 L1 data cache performance 72
5.4 Branch prediction 75

6.1 Existing multimedia benchmarks 78
6.2 HD-VideoBench applications 80
6.3 Input sequences 81
6.4 Execution commands 83
6.5 Rate distortion results 83
6.6 Experimentation platform 84
6.7 Speedup of SIMD optimizations 86

7.1 Support for unaligned loads 93
7.2 Processor configurations used in simulation analysis 98
7.3 Dynamic instruction count for kernels 99

8.1 Parallel MBs with 2D-Wave approach 110
8.2 Maximum parallel MBs for static 3D-Wave 116
8.3 Parallel MBs for dynamic 3D-Wave 117

9.1 Maximum speedup for 2D-Wave 126

9.2 Effect of variable decoding time 128
9.3 Experimentation platform 131
9.4 Profiling of the dynamic scheduling approach 133
9.5 Profiling of tail-submit approach 134

10.1 Instructions for task synchronization 140
10.2 Baseline simulation parameters 143
10.3 Heterogeneous system configuration 144

1 Introduction

In recent years, digital multimedia has become a widely extended application. Multimedia can be defined as the use of different types of media (text, pictures, graphics, sounds, animations and videos) in order to enrich the quality of information [104]. With the recent advances in processor and memory technology, communication networks, mobile devices and display technology, the production, consumption and distribution of digital multimedia content have become possible for the average user of digital devices. In that sense, it is said that digital multimedia has empowered individuals [182].

Digital multimedia can be seen as the integration of computing and other forms of traditional display technologies like motion pictures. The consequences of that integration are changes in the forms and possibilities of language, communication and human expression [67].

One of the most widely extended application of digital multimedia is the combination of digital audio and video. Applications of that technology include: mobile multimedia [104], Digital Video Disc (DVD), Internet Protocol Television (IPTV) (and other forms of Digital Television), and streaming of audio and video content from the Internet.

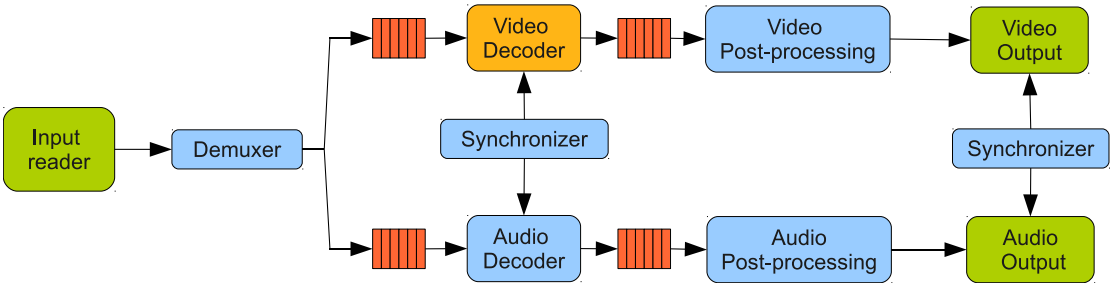


Figure 1.1: Architecture of a generic multimedia player

Digital multimedia can be reproduced on a variety of computing devices, ranging from powerful desktop PCs to set-top boxes and low-power mobile devices [52]. In order to reproduce multimedia content the user needs a multimedia player. The multimedia player can be a standalone application or a player embedded in another application like a web browser. Figure 1.1 shows the generic architecture of a multimedia player. The input content is received from a file or from a network connection. A demuxer (demultiplexer) is applied to the input bitstream for separating the audio and video content streams. After that, separated processing pipelines are applied to each stream. Those pipelines are composed of decoding and post-processing stages. At the end, the decoded and processed output streams are sent to the corresponding output devices. An additional module is in charge of maintaining the proper synchronization between audio and video streams. In a multimedia player that is embedded in a web browser the video output has to be blended with other sources of information such as text and graphic elements.

Application	Acronym	Resolution (Luma) [pixels]	Frame rate [fps]	Uncompressed bit rate [Mbps]	Compressed bit rate [Mbps]
Ultra High Definition	UHDp60	7680×4320	60	23888	> 500
Quad Full High Definition	QHDp50	3840× 2160	50	4977	100-200
Full High Definition	FHDp50	1920×1080	50	1244	20-40
Full High Definition	FHDp25	1920×1080	25	622	8-20
High Definition	HDp25	1280×720	25	277	2-8
Standard Definition	SDp25	720×576	25	124	1-2
Common Intermediate Format	CIFp25	352×288	25	37	0.128-1
Quarter CIF	QCIFp15	176×144	15	4.6	0.05-1

Table 1.1: Bitrate requirements for video decoding. Video in progressive YUV format, with 4:2:0 chroma subsampling [196]

1.1 Trends in Multimedia Applications

From the computational point of view, one the most demanding parts of a digital multimedia player is the processing of digital video, specially video decoding (also called video decompression) [196]. This is due to the high information bandwidth, the complexity of operations, and the real-time requirements. In addition to that, multimedia applications are increasing their quality with each generation requiring, in turn, more computational resources.

The observed trends in the digital multimedia domain, at least for the video domain, can be summarized in three groups:

- A trend towards high quality content
- A trend towards mobile multimedia
- A trend towards multiple formats and extensions

The first trend is towards applications with higher quality video content. To provide better viewing experiences different type of enhancements are being proposed. The first one is the increase in spatial frame resolution: higher resolutions of video are being adopted like High Definition (HD) digital video [127], and there are proposals for higher definitions like Quad-Full High Definition (QFHD) or Ultra High Definition (UHD) [231]. The second technique consists of increasing the temporal frame resolution (or frame rate), like going from 25 to 50 frames per second (fps) in HD systems and even higher frames rates like 100 fps and 300 fps are being considered [19]. Other techniques include the increase in color fidelity and amplitude resolution. Also, alternative displaying techniques like three-dimensional TV (3D-TV) are becoming common [161]. The main consequence of this trend, at least from the computational point of view, is the continuous increase in the computing demands of digital video processing.

Table 1.1 shows the bandwidth (uncompressed bit rate) of different video resolutions starting from the smallest resolution used in mobile devices (QCIF) and going up to the currently proposed highest definition (UHD). As an example, video at Full High Definition at 50 fps (FHDp50) produces 270 times more information per unit of time than video at QCIF resolution at 15 fps (QCIFp15).

Although in recent years there has been an enormous increase in computer network bandwidth and in storage device capacity, the amount of information produced by digital

video is so huge that compression technology is completely necessary to enable practical systems for transmission and storage of video data. To provide higher compression efficiency without sacrificing quality, a new generation of advanced Video COders and DECoders (CODECs) like H.264/AVC has been created [225]. The combination of the complexity of these video CODECs and the higher quality of emerging video applications has resulted in an important increase in the computational requirements of multimedia applications [232].

The second trend is towards mobile multimedia, which is the result of the convergence of mobile phone technology and digital multimedia. Current and emerging applications in this area are: video phone calls, multimedia messages and video streaming from the Internet [182]. As a result, the computing platforms of mobile devices are required to support real-time audio and video playing under severe power and cost constraints.

The final trend is towards multiple standards and formats. In the current state of video technology there is not a single dominant format or standard [225, 232]. In the case of video compression there are multiple formats in use like MPEG-2, MPEG-4, H.264/AVC, VC-1, etc. Some of these formats are not completely fixed because some extensions appear from time to time, and new formats are being developed constantly. The main consequence of this diversity is that it makes very difficult to create application specific solutions for video decoding. In order to support multiple video formats, the platforms for processing video applications should be programmable and flexible.

1.2 Trends in Computer Architecture

The popularization of digital multimedia in the last years have been the effect of positive feedback between computing technology and application trends. The possibility of processing digital audio and video in real-time in consumer equipment has resulted in the popularization of digital multimedia. As users have started to play with multimedia applications the demands for higher quality and new applications has increased pushing an enormous pressure on the computing technology. This co-evolution of applications and technology has resulted in a change in which computing platforms are designed and, at the same time, in the emergence of new applications [142, 128, 70, 53].

The computing demands of video applications have been provided in general purpose processors with a specific support in the architecture for multimedia data and taking advantage of technology scaling. Back in 1993, the highest performance workstations were unable to process low resolution (QCIF) MPEG-1 compressed video in real-time [157]. To solve this limitation processor architectures were enhanced with Simple Instruction Multiple Data (SIMD) capabilities that allowed to process multiple video samples with a single instruction [30]. By using this technique it was possible to process MPEG-1 and MPEG-2 low resolution video using General Purpose Processors (GPPs) [140]. By taking advantage of the performance increase that results from technology scaling it was possible to increase the resolution and frame rate at which digital video can be processed in real-time in GPPs [193].

1.2.1 Scalability limits of single core architectures

The continuous increase in integration density of electronic circuits (also known as Moore's law) states that the number of basic components (i.e. transistors) on a chip will

double every 18 months[1] [162]. During 30 years the increase in the number of transistor has been accompanied by a similar increase in the clock frequency. The net result has been a boost in application performance that comes from frequency scaling (smaller cycle times) and microarchitecture (more operations per cycle) [34]. Figure 1.2 shows the evolution of transistor count and frequency of (some) Intel microprocessors in the last 35 years [62].

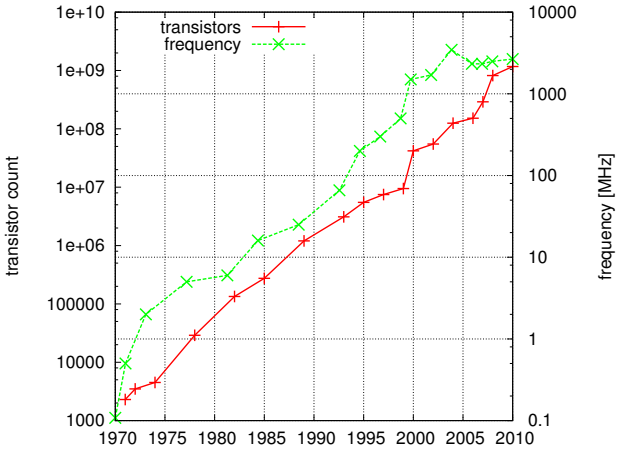


Figure 1.2: Evolution of transistor count and frequency of Intel microprocessors

Due to several technology limits frequency can not longer scale at the same rate than transistor count [1]. This is known as the "frequency wall". Clock frequency can not increase because power density is reaching a limit (this is known as the "power density wall"). Dynamic power (P_D) on a CMOS circuit, as shown in Equation 1.1 (where α is the activity factor; C switched capacitance and V voltage supply) is directly proportional to the frequency (f). When frequency increases beyond a threshold the power per unit of area (power density) can not be dissipated for a given packaging technology. As a result, the frequency of the last generation of processors has remained constant (in some cases it has decreased). A recent estimation made by the International Technology Roadmap for Semiconductors is a frequency increase of 1.05X per year [81].

$$P_D = \alpha C V^2 f \tag{1.1}$$

There is also the "Instruction Level Parallelism (ILP) wall". That means that the effort in terms of design, complexity and power to include support for aggressive ILP (deep pipelines, high issue width, dynamic scheduling, speculation, out-of-order execution, etc) does not pay for the small gains in performance. This is the result of the low ILP that is present in many applications and the difficulties of finding it.

Apart there is the "memory wall". Which means that processor is faster than the memory and, as a consequence, the former has to wait for data to come from the latter. When frequency increases the same happens with the miss penalty for accessing data in the lower levels of the memory hierarchy. On the the hand, an increase in memory size results on an increase in the power consumption due to static power.

[1] Originally expressed as doubling every year, later corrected as doubling every two years, but its common use is that the performance of a computer system will double every 18 months

Finally, there are some manufacturing issues that affect the scaling of processor architectures: One is the "wire delay walls", which limits the maximum distance that a signal can travel on the chip on a single clock cycle. And, other issue is the "variability and reliability wall" that means that circuits fabricated with deep submicron technologies have more divergence and are more susceptible to soft errors and device degradation.

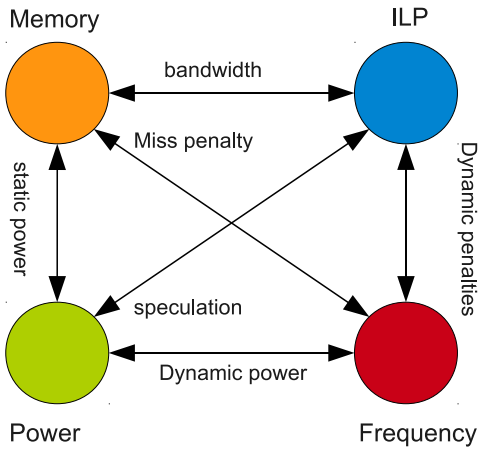


Figure 1.3: Relationship between different "walls" in processor design

The main "walls" are interrelated as shown in Figure 1.3 [264]. For example, an increase in frequency will lead to an increase in the penalties due to ILP support; an increase in the miss penalty for accessing memory and an increase in the dynamic power.

Chip Multi-Processor (CMP) architectures appears as the main solution to all the design and technology issues presented before. With a multicore architecture it is possible to use the density increase to increase the number of cores but decreasing the frequency in such a way that it is possible to obtain (theoretically) the same performance but with a fraction of the power consumption. The scaling of the CMP architecture is done with the number of cores rather than frequency. Usually cores are simpler in terms of their support of ILP in order to further reduce power consumption and complexity, which in turn, allows to include more cores. The "ILP wall" is addressed by raising the level of abstraction from instructions to tasks and letting the programmer to find task-level parallelism in applications. The "memory wall" is addressed with the use of distributed memory hierarchies that include local caches or scratchpad memories. CMPs also alleviate the design and manufacturing issues by reducing the complexity of each core and allowing reuse and replication.

1.2.2 The trend to multicore architectures

The evolution of multicore architectures can be summarized with three groups of trends:

- A trend towards multicore and manycore systems
- A trend towards simple unicore architectures
- A trend towards heterogeneous and asymmetric systems

The result of the first trend is that the increase in performance from one generation of processors to the next one is based mainly in the exploitation of Thread(task)-level Parallelism (TLP). If the increase in the number of cores continues with every generation, in the near future there will be systems with hundreds of cores (also called manycores) [35]. In order to exploit the performance of those parallel architectures fine-grain parallelization of applications is required.

The second trend in multicore architectures is towards systems with simpler cores that usually have extensive support for Data-level Parallelism (DLP) and limited support for ILP [88]. In this new scenario, single core performance is not going to increase at the same rate than in the past, and in multiple cases simpler (and low performance) cores are being used to construct architectures with many cores. As a result, in order to exploit the full potential performance inside a single core, applications need to exploit DLP explicitly using SIMD or similar techniques.

A final trend is towards heterogeneous multicore architectures, that is, architectures with different types of cores. Those architectures can have cores with the same Instruction Set Architecture (ISA) but different microarchitecture (asymmetric cores); or cores with different ISAs (heterogeneous cores), some of them optimized for application specific domains. Those specialized architectures have better performance per unit of power and unit of area than homogeneous architectures and are widely used in the embedded domain [258].

As a conclusion, in order to extract the performance that the new processor architectures provide, applications (in the case of this work: video decoding applications) have to exploit efficiently multiple levels of parallelism, specially DLP and TLP. That constitutes the main objective of this thesis.

1.3 Definition of the problem: Challenges of software video decoding

The main problem is how to provide the performance required by high quality video decoding applications under the following restrictions:

High performance: The system should be able to deliver the performance required to decode high quality video in real-time.

Scalability: Performance should scaled with new "generations" of video decoding applications, specially with the increase in resolution and frame rate.

Flexibility: The architecture should be able to support multiple video formats.

Efficiency: The architecture should be able to provide the required performance using limited resources, specially low area (low cost) and low power consumption.

Taking that into account, we can define the main objective of this thesis as: the efficient exploitation of multiple levels of parallelism for providing the required performance of high definition video decoding applications.

The solution to this problem problem requires a combination of hardware and software approaches. On the software side video decoders need to be redesigned for exploiting

different levels of parallelism. On the hardware side, the computer architecture has to be modified in order to allow a efficient exploitation of the type of parallelism that is present in these applications.

1.4 Thesis contributions

This thesis has three main contributions. First, there is the definition and characterization of a benchmark for high quality video decoding. Second, the evaluation of two techniques for improving SIMD extensions for video decoding. And, finally, some hardware and software optimizations for exploiting Thread-Level Parallelism in a scalable way.

Below we present a most detailed description of each of them.

1.4.1 A benchmark for HD video applications

In order to perform computer architecture studies it is necessary to have a representative set of applications (not just synthetic benchmarks or small kernels) from the target domain. For doing that, first, we defined the expected characteristics of a benchmark; then, we analyzed the existing benchmarks according to these criteria and, based on that, we proposed a new benchmark. These new benchmark includes complete applications from the multimedia domain optimized for high performance; a set of input sequences in HD; and a set of coding options best suited for HD content.

Additionally, we performed a characterization of the computational behavior of H.264 decoding for high definition environments. This analysis allowed us to identify the most time consuming parts of the application, its execution phases and the potential for parallelization.

1.4.2 Scalability of multidimensional vector architectures

As a preliminary study, we analyzed the scalability of two different SIMD extensions: a sub-word extension (like Intel MMX) and a 2-dimensional vector extension. We have shown that scaling the latter results in higher performance and lower complexity than scaling the former for MPEG-2 video coding and decoding.

1.4.3 Efficiency of SIMD extensions for exploiting DLP

We studied the inefficiencies of SIMD architectures for processing video data and proposed a mechanism to remove or, at least, minimize them. In particular, we analyzed the impact of unaligned accesses to memory and proposed architectural and microarchitectural solutions for minimizing the performance loss due to this type of operations.

1.4.4 Thread-level parallelization of video decoding

We performed a detailed analysis of the different thread-level parallelization strategies that can be applied to a software video decoder. These techniques were compared in terms of potential parallelism, scalability, granularity of tasks, communication and synchronization overhead. A new parallelization strategy, that exploits dynamically, both temporal and spatial fine-grain data-level parallelism was proposed.

1.4.5 Scalability of Macroblock-level parallelization

We identified the limitations of current parallel architectures for exploiting the available TLP in video decoding. The limitations and overheads of parallel video decoding were analyzed on a real parallel machine and the potential for software and hardware improvements was identified.

1.4.6 Scalability of heterogeneous multicore architectures

We proposed some hardware and software techniques for improving the scalability of parallel H.264/AVC decoder. That includes a proposal for exploiting frame-level parallelization of the entropy decoder combined with macroblock-level parallelization of macroblock decoding and pipeline-level parallelism between these two stages.

1.5 Historical Context

The main two driving forces of this thesis have been the evolution of video coding technology and the evolution of computer architecture. Figure 1.4 shows a timeline diagram with some of the most important developments in processor technology, video codec design, and the tools and projects related to this thesis.

Our project started in 2003, the same year of the first public release of the H.264/AVC international standard for video coding. At that time the most common video CODECs were MPEG-2-Video and MPEG-4-Visual. Our first step was to conduct a performance characterization of this new CODEC for HD applications. In order to do that we required a software implementation of the standard, but after its release the only available implementation was the reference decoder [114]. We worked initially with this code, but after a careful analysis it was clear that it was not suitable for complexity analysis and computer architecture studies. That is due to its low performance, than in turn, is a consequence of a design made for demonstrating standard features but without applying any significant performance optimization. At the end of 2003, the FFmpeg project [76] published the first version of a H.264/AVC decoder implementation, but their implementation did not include all standard features. Due to that, our first work was done using a mixture of the reference code and the FFmpeg code. At the end of 2004 the FFmpeg project released a complete and highly optimized version of the H.264 decoder. This code was, at that time, at least 10 times faster than the reference code. Even that, the reference code continued to be the most popular implementation used in computer architecture studies and it was included in some benchmarks like SPEC-2006 [93]. That lead us to the definition of our own benchmark devoted to high definition video applications. This benchmark, called *HD-VideoBench* has been used in the rest of our work and we have knowledge of other research groups that are using it for research in computer architecture and multimedia applications.

The other driving force has been the evolution of computer architecture. At the time of the release of the H.264/AVC standard the most common strategy for dealing with the performance requirements of video applications on general purpose computers was the use of SIMD extensions [140]. Most of the works at that time consisted in different techniques for improving the efficiency of SIMD architectures including approaches such as multi-dimensional vector and streaming architectures [132]. In an initial step,

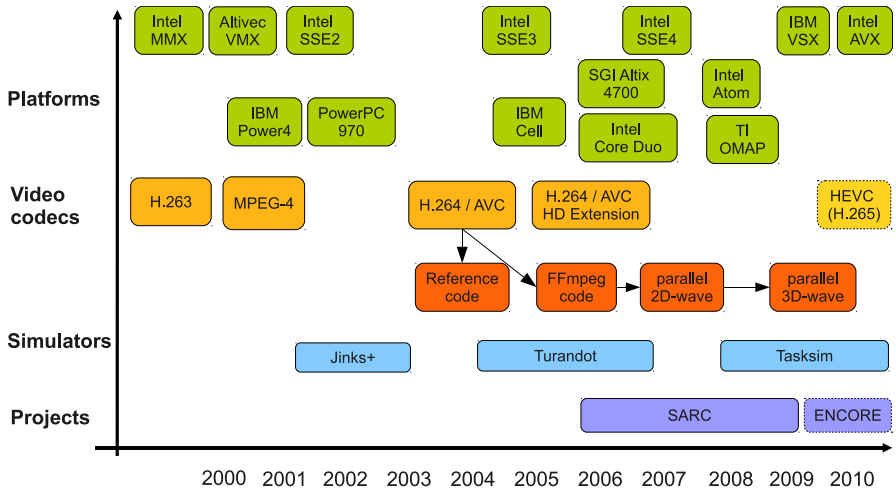


Figure 1.4: Timeline diagram of the technological evolution during the development of this thesis

we worked on an extension for an existing proposal of a 2-dimensional vector architecture called Matrix Oriented Multimedia (MOM) [57] using the MPEG-2 video CODEC. In another work we also perform a study with a conventional SIMD extension (Altivec for the PowerPC architecture [71]) analyzing the sources of inefficiency in SIMD computations, specially in the memory operations, and removing part of them with support for unaligned memory operations.

Our first results suggested that the exploitation of DLP using SIMD instructions and ILP with superscalar architectures on single core processor were insufficient to provide the performance required for real-time HD video decoding. At the same time, in the computer architecture industry it was happening a tremendous paradigm change with the switch to multicore processors. In 2007 we switched from DLP and SIMD extensions in single core processors to TLP in multicore processors. We started with the study of different alternatives for parallelizing H.264 decoding. First, we conducted a theoretical study of the available TLP in the H.264 decoder and a comparison between different parallelization techniques. To carry out this study we started a collaboration within the European Network of Excellence on High Performance and Embedded Architecture and Compilation (HiPEAC). This joint effort included different research groups such as the Computer Engineering Laboratory at the Delft University of Technology, a group from NXP semiconductors and our group at UPC. The main result was a deep study of the parallelization opportunities of H.264 decoding and the proposal of a new parallelization technique called the 3D-Wave decoder [160].

Parallelization of H.264/AVC decoding has been an important research topic in industry and academy. Our work in this area has been inspired by the early work of der Tol et al. [69] and has been done in parallel with many other works and also have influenced other proposals and research [45, 160, 269, 51, 24, 48]. After having our own parallel version of the H.264 decoder, we perform an evaluation on real parallel machines, measured its performance and estimated its bottlenecks. Based on that, we developed proposals for improving the parallelization efficiency. Part of this work was done in the context of the SARC European project [192].

It is important to mention that during the development of our work important changes occurred in both video application and computer architecture domains. As a consequence, we have used different benchmarks (MPEG-2-video and two different versions of H.264/AVC); different target architectures such as single cores with SIMD and vector extensions, and different parallel and multicore processors; and finally different simulators and measurement tools. The heterogeneity that results from the combination of different applications, architectures and tools reflects the rapid evolution of the field during these years and our attempt to follow it within the scope of our research.

1.6 Organization of this document

In this chapter we have presented the motivation of this work. The importance of digital multimedia and video processing have been justified and some trends in this application domain have been presented. Based on these trends a set of challenges for computer architectures for multimedia applications have been described and trends in computer architecture have also been presented. The efficient exploitation of multiple levels of parallelism is proposed as the solution for matching the requirements of video decoding applications with the performance offered by emerging computer architectures. We also presented the main contributions of the thesis and the historical context in which it was developed. The rest of this document is organized as follows:

In Chapter 2 an overview of the video decoding application domain is presented. That includes a short revision of the evolution of video coding technology and a description of the main characteristics of the H.264/AVC video decoder.

A description of the architectures that have been used for video decoding is presented in Chapter 3. They are organized by their degree of programmability into application specific architectures, programmable multimedia processors, and General Purpose Processors.

Chapter 4 describes a scalability analysis of SIMD extensions for video decoding applications. A comparison of 1- and 2-dimensional SIMD extensions is presented using the MPEG-2 codec as a benchmark.

Chapter 5 presents an analysis and a performance characterization of H.264/AVC decoding for HD applications.

Chapter 6 presents a benchmark for evaluating HD video coding and decoding applications. It includes a description of the applications, test videos and coding parameters. An analysis of the performance of the benchmark in different architectures is also presented.

Chapter 7 presents a proposal for improving the efficiency of SIMD extensions for video decoding applications based on the support for unaligned memory accesses.

Chapter 8 presents a study on the different techniques for exploiting TLP in video decoding applications. Different techniques like frame-level, slice-level and macroblock-level parallelization are described and compared.

Chapter 9 describes the implementation of macroblock-level parallelism on a real parallel machine. It also includes a scalability analysis that reveals the limits and bottlenecks of this parallelization.

Chapter 10 presents a study on enhancing the scalability of parallel video decoding using heterogeneous manycore architectures.

Finally, Chapter 11 presents the conclusions, highlights the contributions and suggests ideas for future research in the field.

2 Video Coding Technology and the H.264/AVC Standard

In this chapter we present on overview of video coding technology, define the basic elements of the "hybrid video codec" and make a description of the H.264/AVC video coding standard which represents the state of the art in video coding. The main focus in on the characteristics that has some influence in performance and the potential for exploiting different forms of parallelism.

2.1 Video Compression Objectives

Although the capacity of the network and storage systems has increased in recent years, they remain insufficient to transmit and store uncompressed video. That is due to the fact that along with the increase in bandwidth and disk size there has been an increase in the demanded quality of video applications as it has been shown in Chapter 1.

As a result, video compression continues to be a key technology for creating practical applications. For HD systems, video compression makes feasible the transmission and storage of high bandwidth video signals. For low resolution mobile systems, video compression allows to make an efficient use of limited bandwidth [196].

Video compression is a trade-off between bitrate reduction, quality, delay and complexity. Compression comes at the expense of a reduction in quality and high compression ratios (in video) can only be achieved with lossy compression, in which the decompressed (decoded) video is not equal to the original one. Lossy compression appears as a result of information elimination from the original signal. The result of lossy compression is distortion in the decoded content. Compression also increases latency. This is a result of the computational work and buffering that has to be done for encoding and decoding. Related to that, another effect of compression is an increment in the computational complexity. Maintaining high quality while achieving a significant reduction in bitrate require to apply a set of complex kernels to the input content.

Taken all these factors into account, the main objective of a video codec can be defined as: "Given a maximum allowed delay and a maximum allowed complexity, achieve an optimal trade-off between bitrate and distortion for the range of network environments envisioned in the scope of the applications" [232]

2.2 Video Coding Standards

In order to ensure interoperability between different systems some video codecs have been standardized by international organizations like the International Standard Organization (ISO) and the International Telecommunication Union (ITU-T). The Motion Pictures Expert Group (MPEG) from ISO has developed a series of video standards

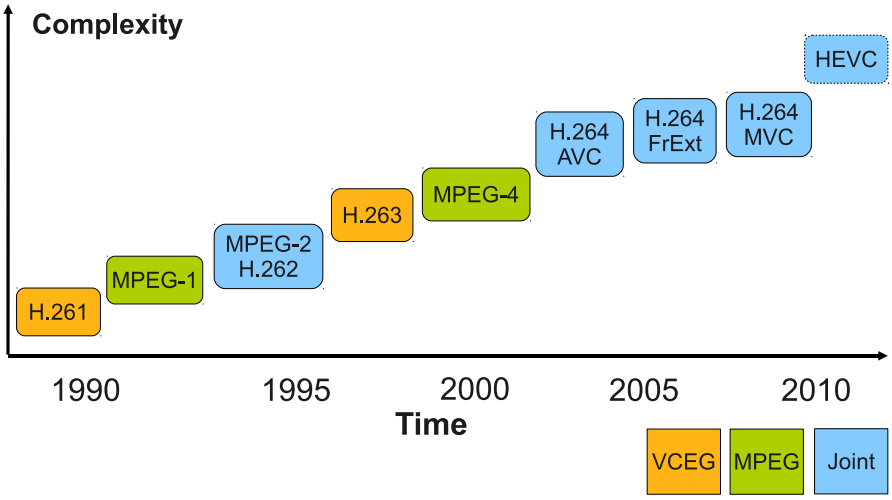


Figure 2.1: Time-line of standard video codecs developed by ISO MPEG and ITU-T VCEG groups

specially oriented to video playback applications. Alongside, the ITU-T has a video committee, called Video Coding Experts Group (VCEG), that has created another family of video codecs oriented to video conferencing applications. To further increase adoption and interoperability of video codecs some standards have developed jointly by both committees. Figure 2.1 presents a time-line description of the video codec standards released by the ISO and ITU-T in the last 20 years.

The firsts widely used international video standards were H.261 [90] (1988) and MPEG-1 [166] (1993). They were developed for low bitrate (less than 2 Mbps) and low resolution (QCIF and CIF) applications like Video-CD and video over telephone networks. In 1995, a joint standard called MPEG-2 and H.262 was published targeting applications with "standard resolution" (720x576 at 25 fps). One of the most successful applications of MPEG-2 has been the DVD [167, 224]. Later on, in 1995, the H.263 standard appeared adding different coding tools for improving compression efficiency[91]. Two additional revisions, known as H.263+ (1998) and H.263++ (2000) were developed for improving compression efficiency and error resilience. In 2001, the MPEG group published another standard called MPEG-4 targeting many multimedia applications, a video codec was published in the MPEG4-Visual part of the standard [168]. Although H.263 and MPEG-4 are different standards they have a compatibility mode that allow basic interoperability. Their main objective was to reduce the bitrate of standard definition video and to improve the quality of low bandwidth applications compared to MPEG-2 and MPEG-1. In 2003, another international standard for video coding was announced. It was a joint effort of both committees, and is called MPEG-4 Part 10 - AVC (Advanced Video Coding) by ISO and H.264 by ITU-T [92]. The main objective was to improve compression efficiency and quality compared to previous codecs. This relatively new standard has a very broad spectrum of applications ranging from low resolution mobile multimedia to high definition playback. It is included as one of the supported codecs in the High Definition Blu-Ray disc (with MPEG-2 and VC-1); and it is widely used for video streaming on Internet and in High Definition Television

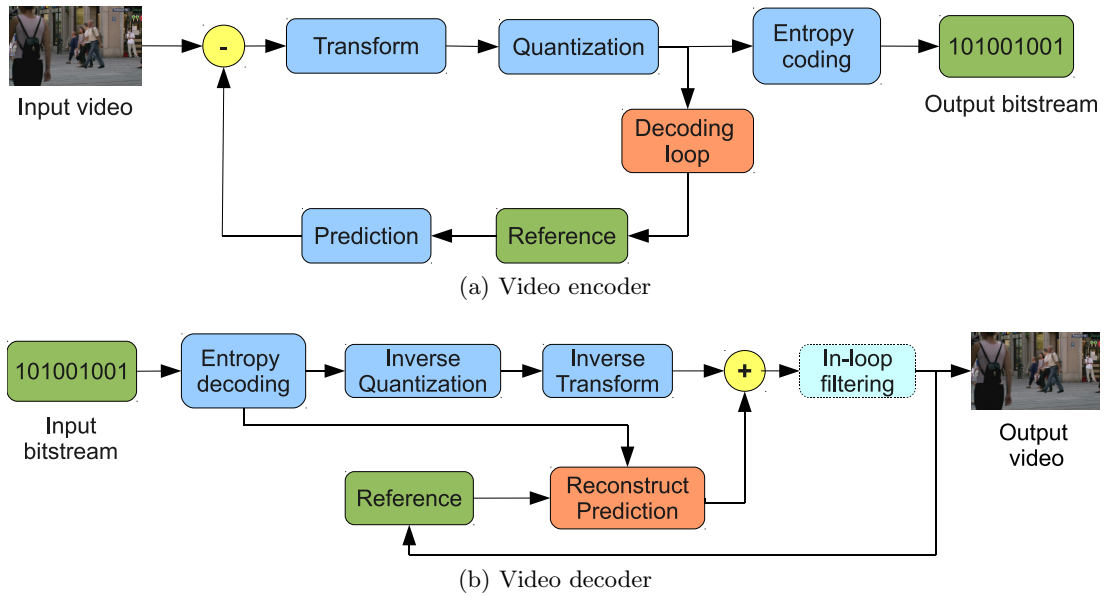


Figure 2.2: General structure of a hybrid video codec

(HDTV) [195]. In 2004 an extension of the H.264/AVC standard appeared, called Fidelity Range Extensions (FREXt) for improving the compression of HD content [233]. In 2007 and 2009 two extensions to the H.264/AVC standard have been defined: one for Scalable Video Coding (SVC) [210] and the other one for Multiview Video Coding (MVC) [253].

Currently there is an ongoing effort for defining a new video coding standard with higher quality and compression efficiency than H.264/AVC. It is being developed by both VCEG and MPEG groups under the Joint Collaborative Team on Video Coding (JCT-VC). They are planning to release a new standard called High Efficiency Video Codec (HEVC) in 2012 or 2013 [110, 234].

As Figure 2.1 suggests qualitatively, the computational complexity of video codecs has increased over time. This is the result of adding more coding tools for improving compression and quality.

2.3 Block-based Hybrid Generic Video Codec

A video codec reduces the size of a digital video signal by exploiting redundancies on the video data [196]. Over the years, different video coding tools have been designed to exploit those redundancies, and some of them are common in most video codecs, specially in those of the MPEG and VCEG families. The use of similar coding tools defines a generic structure of a video codec that has been called the "block-based hybrid generic video codec" [225].

The hybrid video codec is composed by three main stages: prediction, transformation/quantization and entropy coding. Prediction consists of exploiting spatial or temporal redundancies for deriving a block based on previous ones. The predicted block is then subtracted from the current one forming the error signal. A transform is next applied to the error signal in order to change from the spatial domain to the frequency

domain. Then, Quantization (Q) is applied to reduce the dynamic range of the signals. As a result, some of the high frequency components become zero, reducing the number of coefficients to be transmitted. Quantization is what makes a hybrid video codec lossy, because it is not possible to reconstruct the original video from a reduced set of frequency components. The quantized components and other prediction information are then passed to the entropy coding stage in which they are represented with a code that reduce its size by exploiting statistical redundancies. The result of entropy coding is a compressed bitstream. Figure 2.2a shows the general structure of the hybrid video encoder.

2.3.1 Block-based Structure and Color Coding

In a block-based codec each frame of the video sequence is divided in small rectangular blocks called MacroBlocks (MBs). In this way the coding tools are applied to MBs rather than to whole frames reducing the computational complexity. Figure 2.3 shows a generic view of the data elements of a video sequence. It starts from the sequence of frames that compose a whole video. Several frames can form a Group of Pictures (GOP) which is an independent set of frames. Each frame can be composed of independent sections called slices, and these ones, in turn, are composed of MBs. Each MB is formed by blocks, which in turn, are composed of pictures samples.

A MB is composed of separated blocks for luma (or relative luminance, denoted by Y) and chroma signals (color information, denoted by Cb for blue and Cr for red). A pre-processing step has to be applied to convert video from a color component format (like RGB) to a difference color format (like YCbCr). By coding the video signals in this format it is possible to reduce the amount of color information in such a way that is not noticeable for the human eye [189]. The most common color format is called 4:2:0 in which chroma is sub-sampled by 2 in both spatial dimensions. In most MPEG and ITU-T video codecs, each MB has one 16×16 luma block and two 8×8 chroma blocks.

2.3.2 Prediction

Prediction can be done in the spatial or in the temporal domains. Prediction in the spatial domain consists of estimating the values of the current block by using only samples from the same frame. Prediction in the temporal domain consists of estimating the value of the current macroblock using blocks from other frames, called reference frames. One form of inter-prediction is called Motion Compensation (MC) in which the predicted block in the current frame is generated as if it was an area in a reference frame that have moved to a different position in the current frame. This block displacement is expressed and transmitted as a Motion Vector (MV). In order to find such blocks (and vectors) the encoder should perform a search process called Motion Estimation (ME).

The prediction strategy defines the type of each frame. In MPEG and VCEG video codecs there are three type of frames: Intra-coded frame (I), Predicted frame (P) and Bi-predicted frames (B) [224].

In I-frames all the MBs are coded using intra-prediction. I-frames are inserted by the encoder as random-access points or when the amount of changes between frames do not allow to create an efficient temporal prediction. I-frames are also used as references for P- and B-frames.

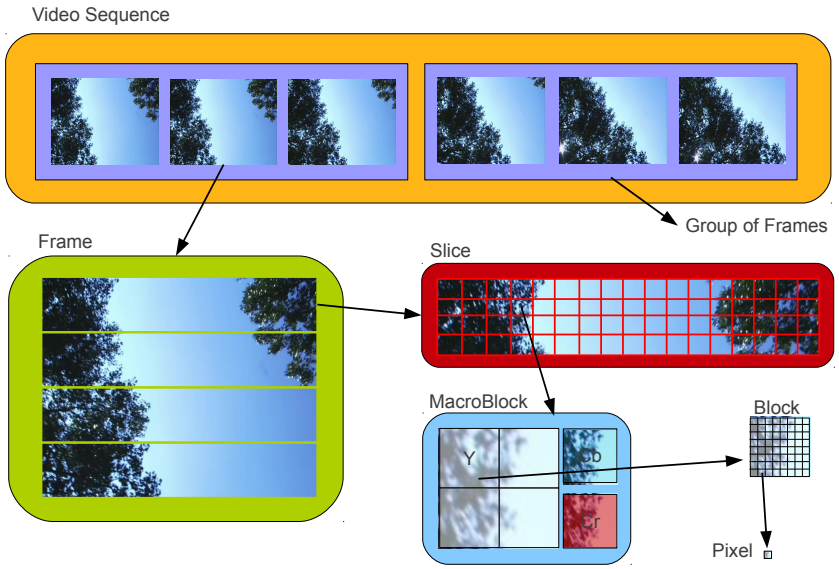


Figure 2.3: Data elements on a generic video sequence: GOPs, frames, slices, and macroblocks

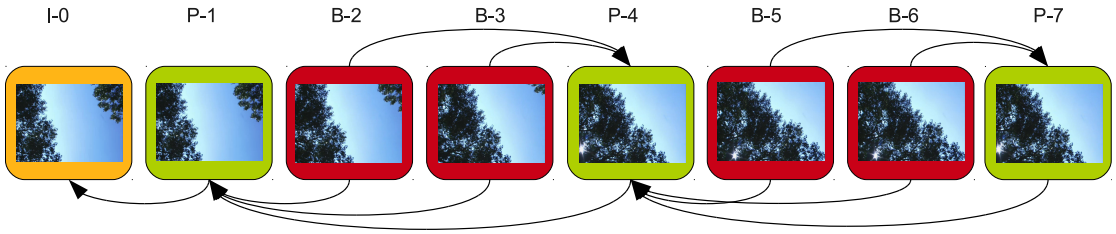
In P-frames MBs can be temporal predicted using one frame (or one frame direction) as reference. That means that the decoding of a P-frame requires that the reference frame has already been decoded, defining an order constraint in the decoding process due to data dependencies. P-frames can be used as reference frames for other P-frames or B-frames.

In B-frames MBs can be predicted using samples from two reference frames (or frame directions). A B-type macroblock can be predicted in three ways: with a forward reference, backward reference or with a combination of forward and backward references combined to produce a single prediction. Decoding a B-frame requires that all reference frames have been already decoded, creating an order constraint involving past and future frames. Figure 2.4a shows a diagram of a 7 frame sequence in display order with a I-P-B-B-P-B-B structure in which arrows indicate the data dependencies at the frame level. For transmission and processing B-frames require a change in the order of frames as depicted in Figure 2.4b.

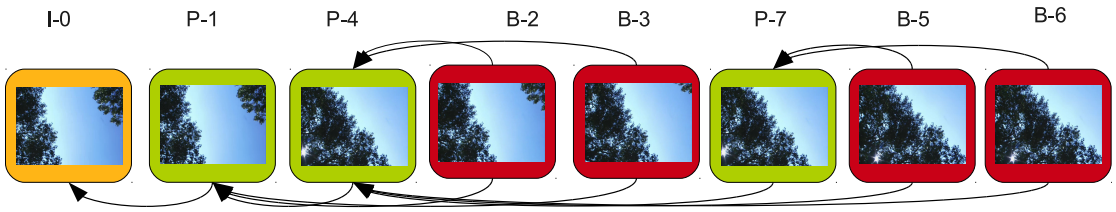
2.3.3 Transform

The main purpose of the transform is to separate the input data into frequency components (decorrelation) and concentrate most of the energy of the signal in a small set of coefficients (compaction) [195]. The 2-dimensional Discrete Cosine Transform (2D-DCT) is the most widely used transform for block-based video codecs because its mathematical properties allow efficient implementations [3].

The 2D-DCT maps a $N \times N$ matrix of samples X into a new $N \times N$ matrix of transform coefficients Y by a linear transformation. The inverse transform (2D-IDCT) maps a matrix of transform coefficients Y into a matrix of samples X:



(a) Frames in display order



(b) Frames in processing order

Figure 2.4: Type of frames in a MPEG/VCEG hybrid video codec

$$\begin{aligned} \mathbf{Y} &= \mathbf{A}\mathbf{X}\mathbf{A}^{\mathbf{T}} \\ \mathbf{X} &= \mathbf{A}^{\mathbf{T}}\mathbf{Y}\mathbf{A} \end{aligned} \tag{2.1}$$

where A is a $N \times N$ transform matrix. The transform matrix for a 4×4 block is defined as follows:

$$\mathbf{A} = \begin{bmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & c \end{bmatrix} \tag{2.2}$$

where $a = \frac{1}{2}$, $b = \sqrt{\frac{1}{2}}\cos\left(\frac{\pi}{8}\right)$ and $c = \sqrt{\frac{1}{2}}\cos\left(\frac{3\pi}{8}\right)$.

2.3.4 Entropy Coding

In a video codec entropy coding is used to compress the data that results from the prediction and transform processes including: transform coefficients, motion vectors, and side information. Compression is achieved by exploiting statistical redundancies in the set of input symbols, and works by representing frequently occurring symbols with a small number of bits and infrequently occurring symbols with a larger number of bits [196].

The two main techniques used for entropy coding are Huffman coding [98] and arithmetic coding [257]. In Huffman coding symbol of the input sequence is represented with a variable length code that uses an integral number of bits. In arithmetic coding a complete sequence of symbols is represented with a real number in the range $[0, 1)$ allowing to assign fractional number of bits to input symbols based on its probability. Because of that arithmetic coding achieves a better compression performance than Huffman coding, and it can reach the optimal data compression as defined by Shanon's source coding theorem [217].

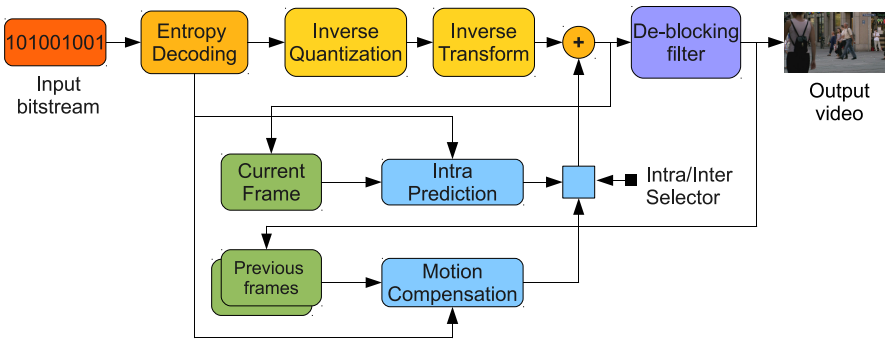


Figure 2.5: General structure of the H.264/AVC decoder

2.3.5 Decoding Process

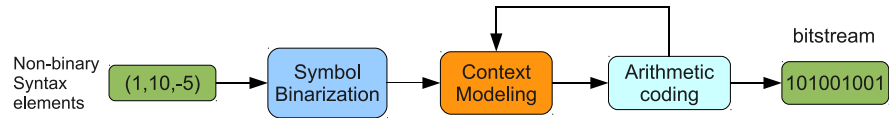
In order to decompress the video, the decoder has to perform similar stages than encoder but in a reverse order. In Figure 2.2b, the structure of a general video decoder is shown. The input is a compressed bitstream that first has to be entropy decoded. The output of entropy decoding are the quantized transformed coefficients and prediction information. By applying an "inverse quantization" (or more accurately, a re-scaling operation) to the quantized coefficients, a reconstructed version of the transformed coefficients is derived. Then, an inverse transform is applied to the reconstructed coefficients to obtain the error signal. The predicted blocks are reconstructed either with intra- or inter-prediction. After that, the error signal is added to the prediction signal forming a decoded video signal. An optional in-loop filter (deblocking, de-ringing, etc) is applied for improving the visual quality of the output video.

The variations between different video codecs that implement the structure of the hybrid video codec are in the coding tools applied in each one of the mentioned generic stages. In the next section we are going to present a detailed description of the coding tools available in the H.264/AVC standard along with a comparison with other standards.

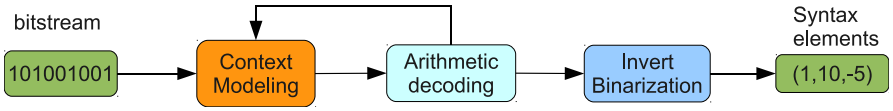
2.4 H.264/AVC Video codec

H.264/AVC is based on the same block-based motion compensation and transform-based coding framework of prior MPEG and ITU-T video coding standards. It provides higher coding efficiency through added features and functionality that, in turn, entail additional complexity. H.264/AVC includes a lot of new coding tools for different application scenarios but here we present only a summary of the tools that have more effect on the performance of the video decoder which are the focus of this work.

A generic structure of the H.264/AVC decoder is presented in Figure 2.5. Below, we present some details of each decoder stage. The full specification can be found in the standard [92].



(a) CABAC encoder



(b) CABAC decoder

Figure 2.6: CABAC arithmetic codec

2.4.1 Entropy Decoding: CAVLC and CABAC

H.264/AVC includes two different algorithms for entropy decoding. The first one is Context Adaptive Variable Length Coding (CA-VLC), which is an adaptive variant of Huffman coding oriented to applications that require less computational complexity in the entropy decoder. The second one is Context Adaptive Binary Arithmetic Coding (CABAC), which is based on arithmetic coding techniques and results in compression ratios between 10% and 14% bigger than CAVLC [156]. In this work we have focused on H.264/AVC decoders that uses CABAC because it results in higher compression ratios and it is widely used in HD applications. Below we present some details of the CABAC coding algorithm.

Context Adaptive Binary Arithmetic Coding

CABAC uses binary coding, which means that the source alphabet is composed of just two symbols, requiring that all non-binary inputs have to be converted to binary symbols before the actual arithmetic coding. Binary arithmetic coding simplifies the coding of each binary symbol at the cost of a reduced performance because the final throughput can not be larger that one bit per few CPU cycles [205]. CABAC also uses adaptive coding with context models, which means that the estimation of source symbols probabilities is done during the coding process. A context model in CABAC is composed of the probability of the Least Probable Symbol (LPS) and the value of the Most Probable Symbol (MPS) [204].

The coding process of CABAC involves the following three stages as described in Figure 2.6a:

Binarization : non-binary input symbols are mapped to a variable length binary code (bin string).

Context model selection : Select a context model for the current bin. Context models are probability models for bins in the binarized symbol.

Binary arithmetic encoder : Encode the current bin using the probability model defined by the context model.

CABAC Decoding Algorithm

The decoding process involves a similar set of operations than encoding but in a reverse order, as shown in Figure 2.6b. For every syntax element a series of bins are decoded to make a bin string. The CABAC decoder has a special unit for detecting when a decoded sequence of bins matches a valid codeword, if that occurs the original syntax element is reconstructed with a de-binarization process.

To decode a bin, a proper context model should be used. A context selection process is performed to obtain the context model for each particular bin. This is done by accessing a context model memory using a context model index γ_i that depends on the syntax element being decoded and the value of previous decoded bins.

The arithmetic decoding starts with some bits from the bitstream (called the offset), an initial range (R), and the context model. The range R is divided into two sub-ranges: R_{MPS} and R_{LPS}, where R_{LPS} is calculated by multiplying the range by the probability of the LPS symbol. R_{MPS} is computed as the difference between the range and the R_{LPS}. If the offset is less than R_{MPS} the bin is decoded as the MPS and the range for the next bin is set to R_{MPS}, otherwise the bin is decoded as LPS and the next range is R_{LPS}. It is worth to mention that for performance reasons in CABAC all the multiplications have been replaced by tables of precomputed values. During arithmetic decoding a renormalization process is applied to keep the range and the offset in a fixed precision. After decoding each bin, the context model is updated, which includes an update of the LPS symbol probability and the MPS value [156, 119, 267].

The context selection process requires both the current data being decoded and previous decoded data. This creates a strong data dependency between bins in the the decoding process restricting the exploitation of data-level parallelism inside the binary decoder. CABAC contexts are maintained for all the macroblocks of a complete slice. At the end of a slice context probabilities are reset to an initial state. Because of that slices are the smallest data partitions for the parallelization of the CABAC algorithm.

2.4.2 Inverse Transform

H.264/AVC transform is an integer orthogonal approximation of the DCT allowing bit exact implementations for decoders and encoders. The main transform is applied to 4×4 blocks and is useful for reducing ringing artifacts. The FRExt extension allows another DCT transform for 8×8 blocks which is useful in HD video for the preservation of fine details and textures [233]. The encoder is allowed to select between the 4×4 and 8×8 transforms on a macroblock by macroblock basis. The transforms employ only integer arithmetic without multiplications, with coefficients and scaling factors that allow for 16-bit arithmetic computation [155].

In H.264 the transform is formed by rounding and scaling the matrix A (See Equa-

tion 2.2). The inverse 4×4 transform of H.264 is described in equation (2.3):

$$\begin{aligned}\mathbf{X} &= \mathbf{A_i^T}(\mathbf{Y} \otimes \mathbf{E_i})\mathbf{A_i} \\ &= \begin{bmatrix} 1 & 1 & 1 & \frac{1}{2} \\ 1 & \frac{1}{2} & -1 & -1 \\ 1 & -\frac{1}{2} & -1 & 1 \\ 1 & -1 & 1 & -\frac{1}{2} \end{bmatrix} \left([Y] \otimes \begin{bmatrix} a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \\ a^2 & ab2 & a^2 & ab \\ ab & b^2 & ab & b^2 \end{bmatrix} \right) \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \frac{1}{2} & -\frac{1}{2} & -1 \\ 1 & -1 & -1 & 1 \\ \frac{1}{2} & -1 & 1 & \frac{1}{2} \end{bmatrix}\end{aligned} \tag{2.3}$$

The matrix of coefficients Y is pre-scaled multiplying it by each element of the scaling matrix E_i ($\otimes$ means scalar multiplication rather than matrix multiplication). The factors in the transform matrices A_i and A_i^T allow implementations with shifts and adds integer arithmetic operations.

As with other DCTs, in H.264 DCT it is possible to exploit data-level parallelism because the same operations are applied to a block of data. The main difference is that H.264/AVC uses a small data set compared to previous video codecs and also allows variable block sizes (4×4 and 8×8) which introduces control dependencies to detect the different modes.

2.4.3 Quantization

H.264 uses a scalar quantizer. In its general form a scalar quantizer can be expressed as:

$$X_q(i,j) = \frac{X(i,j)}{Q_s} \tag{2.4}$$

where i and j are the row and column indexes of the sample and Q_s is the quantizer step size. In H.264/AVC a total of 52 values of Q_s are supported and these are indexed by a Quantization Parameter (QP).

Direct Quantization and Scaling

The definition and implementation of the H.264 quantizer is such that it avoids division or floating point arithmetic, it also incorporates the post- and pre-scaling matrices used for simplifying the transform. The post-scaling factor (PoF) that appears in the direct transform is incorporated into the forward quantizer:

$$X_q(i,j) = X(i,j)\frac{PoF}{Q_s} \tag{2.5}$$

$\frac{PoF}{Q_s}$ is implemented as a multiplication and a right shift avoiding any division operations in the quantization process.

Inverse Quantization and Rescaling

The decoder performs inverse quantization by re-scaling the quantized data:

$$X_r = X_q(i,j)Q_s \tag{2.6}$$

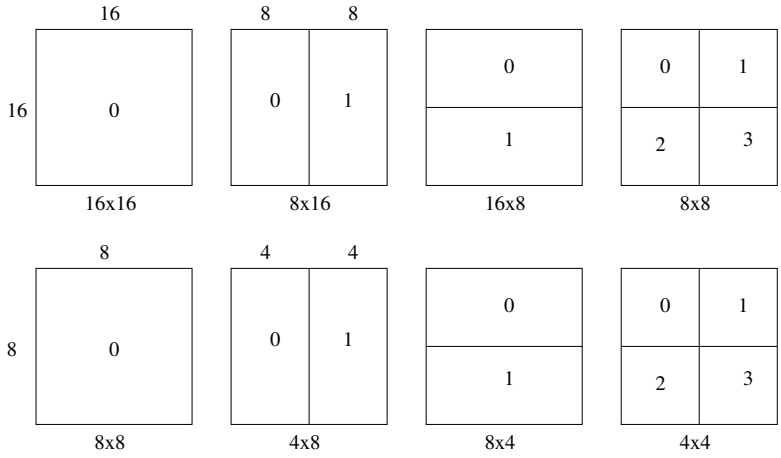


Figure 2.7: Variable block size for motion compensation

The pre-scaling factor (PreF) for the inverse transform is incorporated in this operation, together with a constant scaling factor to avoid rounding errors:

$$X_r = X_q(i,j) \cdot Q_s \cdot PreF \cdot 64 \tag{2.7}$$

2.4.4 Inter-prediction

H.264/AVC introduces a lot of improvements in the inter-prediction stage like variable block-size motion compensation with fractional pixel precision; the use of multiple reference frames for prediction with a weighted combination of the prediction signals, and the use of B-frames as references for prediction [79]. We present a review of these features and their potential impact on decoder performance.

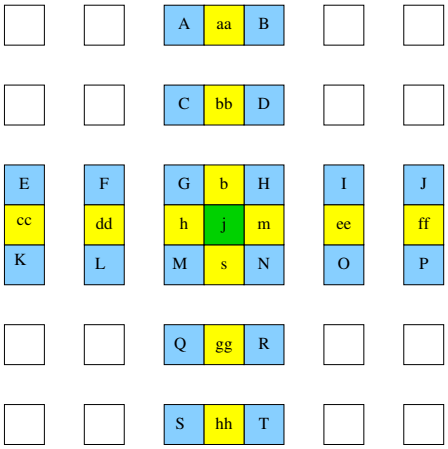
Variable Block Size

H.264/AVC supports block sizes that range from 16×16 to 4×4 samples, as shown in Figure 2.7. The encoder can select big block sizes for areas with low spatial detail or coarse grain motion resulting in less bits for encoding the motion vectors. Small block sizes are useful for reducing the prediction residual of areas with high spatial detail or fine grain motion. Variable block size motion estimation allows the encoder to adapt the block size to the properties of the input video allowing higher compression ratios while, at the same time, reducing the blocking noise [195].

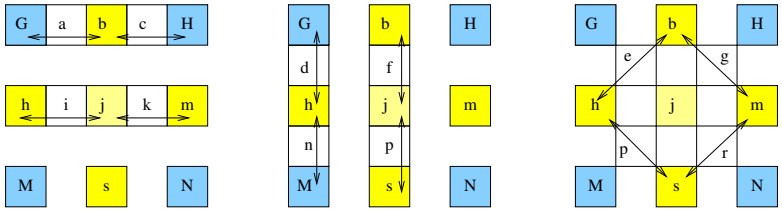
From the performance point of view the use of variable block size reduces the efficiency of data-level parallelization. On the one hand, it is necessary to include control code to detect the size of each block in a macroblock. On the other hand, in the smaller blocks (like 4×4) the amount of DLP is reduced, which leads to a bigger impact of overheads in DLP processing.

Fractional Motion Compensation

Fractional motion estimation allows the encoder to define motion vectors with fractional pixel values, this has the advantage of finding prediction signals with less energy. In



(a) Half-pel



(b) Quarter-pel

Figure 2.8: Interpolation for the luma component

the decoder, an interpolation filter has to be applied to the reference area in order to generate fractional picture samples. In H.264/AVC interpolation is defined separately for luma and chroma signals [254].

Luma Interpolation

Luma interpolation is performed in two stages. In the first one, half-sample positions are calculated using a 6-tap FIR filter, as shown in Figure 2.8a. The FIR filter is defined in Equation 2.8 and its coefficients are $coeff[-2..3] = [1, -5, 20, 20, -5, 1]$. After filtering, the results are rounded and clipped. In the second stage, quarter-sample position are generated by linear interpolation, as shown in Figure 2.8b.

$$a(i+2) = \Big(\sum_{i=-2}^{3} coeff[i] \cdot x_i \Big) / 32 \tag{2.8}$$

In total there are 16 modes in which luma interpolation can be applied: no interpolation, half-pel interpolation (with three different modes) and quarter-pel interpolation (with 12 different modes). For example, according to figure 2.8a sub-sample position 'j' is generated with horizontal and vertical interpolation as: $j = round\Big((cc - 5dd + 20h + 20m - 5ee + ff)/32\Big)$. After this, quarter pel position 'i' is generated with horizontal linear interpolation as $i = round\Big((h + j)/2\Big)$.

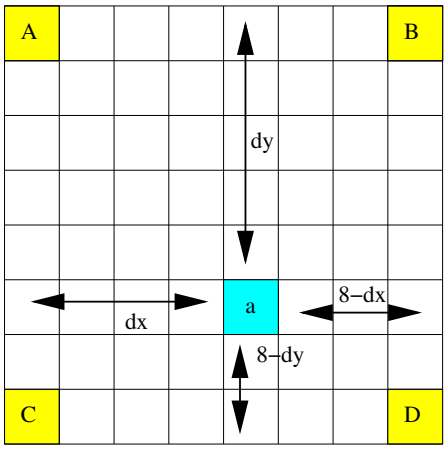


Figure 2.9: Half-pixel interpolation for chroma components

Chroma Interpolation

In chroma interpolation, each sub-sample position is generated as a linear combination of the neighboring integer sample positions, as shown in Figure 2.9. The linear interpolation is defined in Equation 2.9

$$a = round\Big(\big((8-dx)\cdot(8-d_y)A + d_x\cdot(8-d_y)B + (8-d_x)\cdot C + d_x\cdot d_y D\big)/64\Big) \quad (2.9)$$

Luma and chroma interpolation filters are amenable for DLP processing because they are applied to complete data blocks. But the definition of multiple interpolation modes combined with the variable block size requires the inclusion of mode detection code that introduces a significant amount of branches, which in turn, reduce the efficiency of DLP processing.

Multiple Reference Frames and Weighted Prediction

In previous video standards, in B-frames the final prediction is obtained with a linear interpolation of the predictions from the backward and forward predictions. In H.264/AVC, B frames have more flexibility: the final prediction can be obtained with a prediction from two reference frames regardless of their temporal direction [79]. In order to separate the temporal directions from the frames used for prediction, in H.264/AVC the two sources of prediction are called lists. A B-frame is predicted using reference frames from list0 and list1, as shown in Equation 2.10.

$$pred(i,j) = \Big(pred0(i,j) + pred1(i,j) + 1\Big) >> 1 \quad (2.10)$$

Additionally, H.264/AVC allows the encoder to define a weighted combination of the two predictions, as shown in Equation 2.11.

$$pred(i,j) = \Big(w0\cdot pred0(i,j) + w1\cdot pred1(i,j) + 1\Big) >> 1 \quad (2.11)$$

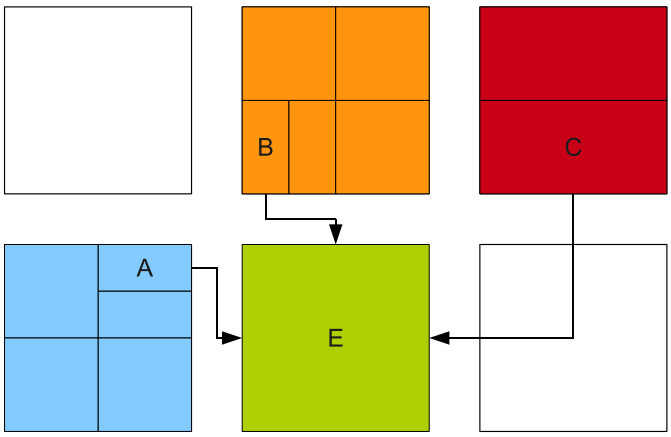


Figure 2.10: Motion vector prediction in neighboring macroblocks

Finally, another improvement compared to previous video codecs is the use of a large window for the reference frames, that allows the use of up to 16 reference frames. In some cases, this allows for a significant reduction of the bitrate compared to the traditional model of just one or two reference frames.

In terms of performance, B-frames require more computing power than processing P- and I-frames because multiple reference areas have to be computed and combined. At the same time, B-frames in H.264/AVC increases application memory usage because more reference frames are active at any given time. From the point of view of data-parallelism, B-frames generate a more complicated dependency graph.

Motion Vector Prediction

In order to reconstruct an inter-predicted block it is necessary to access an area in the reference frame pointed by a motion vector. In H.264/AVC motion vectors are predicted from vectors of nearby, previously coded blocks [195]. In the decoder, the motion vector is reconstructed by forming a combination of motion vectors from neighbor blocks and adding them to the motion vector difference coded in the bitstream. Figure 2.10 shows an example of the vector prediction of a macroblock (E) from neighbor partitions on the left (A), top (B) and top-right (C).

One consequence of motion vector prediction is that it creates data dependencies between macroblocks. In order to decode an inter-macroblock it is necessary to have access to the motion vectors of the MBs on the left, top and top-right.

2.4.5 Intra-prediction

H.264/AVC uses spatial intra-prediction in which the predicted MB is created using previously decoded pixels. The standard supports three different types of spatial prediction depending of block size: 4×4 luma prediction, 8×8 luma prediction and 16×16 luma (and corresponding chroma block) prediction.

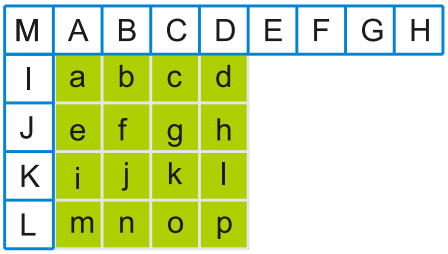
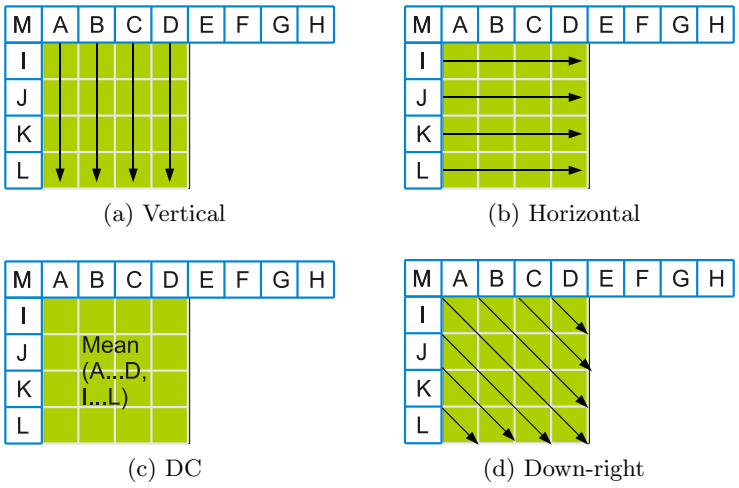


Figure 2.11: Intra-prediction sample labels for 4×4 blocks



(a) Vertical (b) Horizontal

(c) DC (d) Down-right

Figure 2.12: Intra-prediction modes for 4×4 blocks

4x4 Intra-prediction

This type of prediction is well suited for coding of parts of a picture with significant spatial detail [256]. The whole 16×16 MB is divided into sixteen 4×4 sub-blocks and intra-prediction is applied to each one of them. The standard defines nine prediction modes that the encoder can choose independently for each sub-block. Prediction of samples $[a...p]$ is made using previously decoded samples from the top block, top-right, and left blocks $[A...M]$, as shown in Figure 2.11. Four of the nine prediction modes are shown in Figure 2.12. Vertical and Horizontal modes are extrapolation of the previous samples as indicated by the arrows in the Figures 2.12b and 2.12a. DC-mode (Figure 2.12c) is made from an average of samples $[A, B, C, D, I, J, K, L]$. The other six modes are variations of diagonal interpolation, one of which is illustrated in Figure 2.12d. In this mode, the prediction is formed from a weighted average of prediction samples [195].

8x8 Intra-prediction

The FRext extension defined intra-prediction for 8×8 blocks. The prediction modes are basically the same as in 4×4 intra-prediction with the addition of low-pass filtering to improve prediction performance [233].

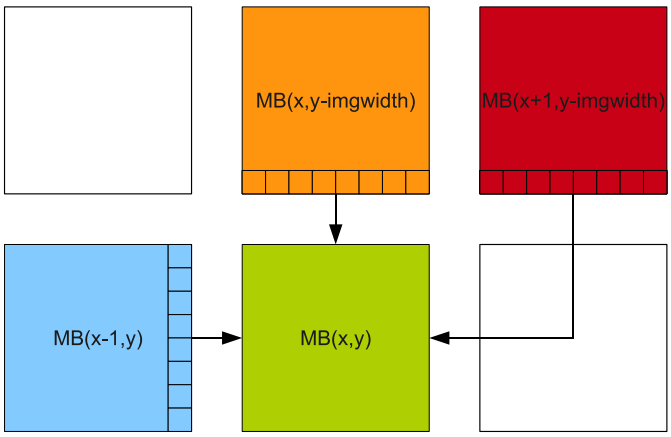


Figure 2.13: Data dependencies due to intra-prediction

16x16 Intra-prediction

The Intra 16 × 16 mode, performs the prediction of the whole MB and is well suited for coding smooth areas of the frame with gently changing luminance. Four different prediction modes are available for this type of prediction: vertical prediction, horizontal prediction, DC-prediction and plane-prediction. The first three ones are very similar to the modes available for 4 × 4 blocks. Plane-prediction uses a linear function between the neighboring samples [256, 178].

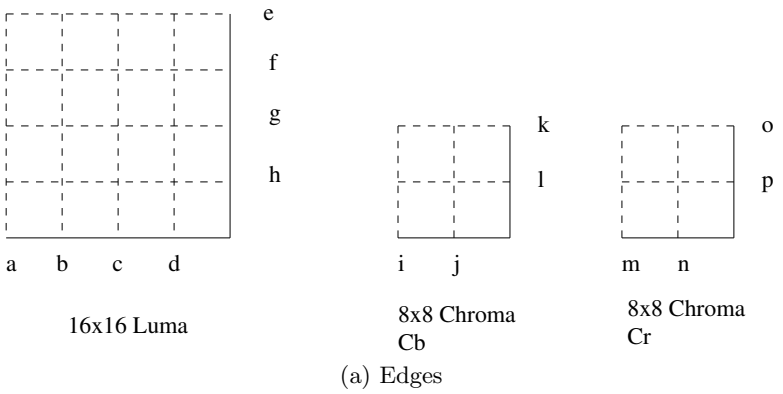
Data Dependencies due to Intra-prediction

One consequence of intra-prediction is that it creates data dependencies between macroblocks. In order to decode an intra-macroblock it is necessary to decode first the macroblocks on the left, top and top-right. Figure 2.13 shows the dependencies between the current macroblock MB(x,y) and its neighbor macroblocks on the left MB(x-1,y), top MB(x,y-imgwidth) and top-right MB(x+1,y-imgwidth). Where *imgwidth* is the frame width in macroblocks.

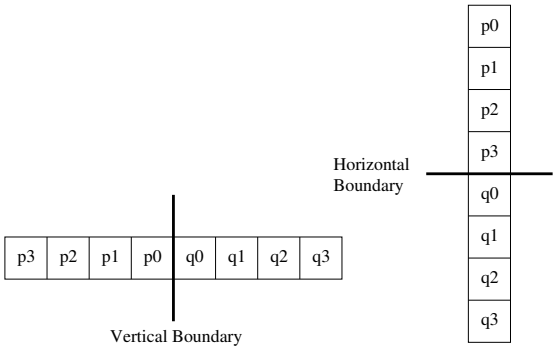
2.4.6 In-loop Deblocking Filter

H.264/AVC includes an in-loop adaptive deblocking filter. It was added in order to reduce the artifacts produced by the block-based structure of the coding process [152]. The filter is located in-loop which means that filtered frames are used as reference frames for motion compensation.

The deblocking filter is adaptive which means that filtering is applied to the block edges with more probable effects of blocking distortion and is reduced (or not apply at all) on the block boundaries that has original input content. The filter is adaptive at the slice, block and sample levels. At the slice level, the encoder can influence the amount of filtering applied for all blocks in the slice. At the block-level, the filter is adjusted depending on the MB type (intra, inter), motion differences and the presence of transform coefficients. Finally, at the sample level, the filter that is applied depends on the sample values at the edge of two adjacent blocks.



(a) Edges



(b) Block boundaries

Figure 2.14: Deblocking filter

The filter is applied on a macroblock basis in scan order. It is applied to vertical and horizontal edges of 4×4 blocks, both for luma and chroma components. Figure 2.14a shows the block edges used for filtering. The amount of filtering at block-level is controlled by the BoundaryStrength (BS) parameter, which depends on the coding modes of the adjacent blocks. It has four values: 0 means no filter, 1-3 is normal filter, and 4 is strong filter.

As an example, the basic filter (for $BS = [1, 3]$) affects the p0 and q0 samples depending on the values of p0,p1,q0 and q1, as shown in Figure 2.14b. The filter is applied if the following condition is true:

$$|p_0 - q_0| < \alpha \quad and \quad |p_1 - p_0| < \beta \quad and \quad |q_1 - q_0| \leq \beta$$

where α and β are thresholds defined in the standard that depend on the Quantization Parameter (QP). They are used to switch-off the filter when QP is low.

In Equations 2.12 and 2.13 the operations of the basic filter are shown. An important feature of the filter is the clipping process expressed with the Min-Max operations, where c_0 is a parameter defined in the standard that allows to adjust the level of filtering. The adaptivity of the filter is obtained, in part, by controlling this parameter when clipping [152].

$$p'_0 = p_0 + \Delta_0 \tag{2.12}$$

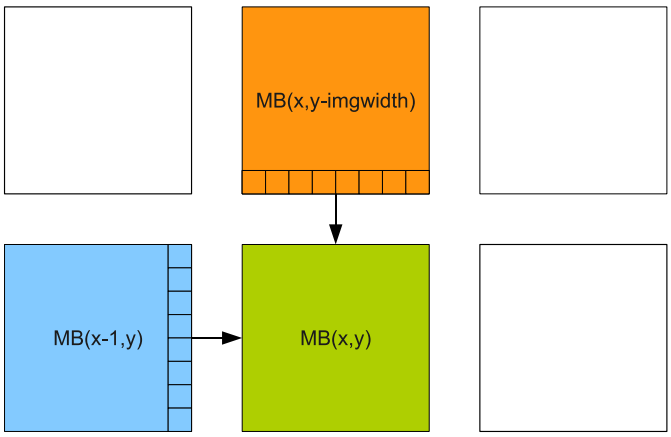


Figure 2.15: Macroblock dependencies due to the deblocking filter

$$q'_0 = q_0 - \Delta_0 \tag{2.13}$$

$$\Delta_{0i} = \big(4(q_0 + p_0) + (p_1 - q_1) + 4\big) \gg 3 \tag{2.14}$$

$$\Delta_0 = Min\big(Max(-c_0, \Delta_{0i}), c_0\big) \tag{2.15}$$

Implications on Performance and Parallelization

An important consequence of adaptivity is an increase in the computational complexity of the decoding process, due to the presence of conditional branches in the innermost loop of the deblocking filter. Memory complexity is also increased because all samples in a 4×4 block are read in order to determine the amount of filtering, and then, up to three samples are modified and stored back to memory. Similarly, exploitation of DLP is affected by the filter adaptivity because the operation that is applied to each sample depends on the input values.

At the macroblock granularity the deblocking filter introduces data dependencies because filtering of the current block requires data from the left and top macroblock as shown in Figure 2.15

2.4.7 Comparison with Previous Video codecs

In Table 2.1 the main features of H.264/AVC are summarized and compared with MPEG-2 and MPEG-4 video codecs [240, 178, 233, 256, 224, 123]. One of the main differences between H.264/AVC and the other video codecs is that the former allows a variable block size for motion compensation; while MPEG-2 only supports 16x16 pixel blocks and MPEG-4 16x16 down to 8x8 pixel blocks. Additionally, H.264/AVC uses a quarter sample resolution for motion estimation, a feature that is optional in MPEG-4 and not available in MPEG-2. Another important difference is that H.264/AVC supports multiple reference frames for motion compensation compared to a single one in the other two codecs.

Features	MPEG-2	MPEG-4 ASP	H.264/AVC High
Macroblock size	16×16	16×16	16×16
Block size	8×8	16×16,16×8,8×8	16×16,16×8,8×16, 8×8,4×8,8×4,4×4
Transform	8×8 DCT	8×8 DCT	8×8,4×4 integer DCT
Pel-Accuracy	$1, 1/2$ pel	$1, 1/2, 1/4$ pel	$1, 1/2, 1/4$ pel
Reference frames	One frame	One frame	Multiple frames (up to 16 frames)
Bidirectional prediction	forward/backward	forward/backward	forward/backward forward/forward forward/backward
Intra-prediction	DC-prediction	coefficient prediction	$4x4$ spatial $16x16$ spatial
Deblocking filter	No	No	Yes
Weighted prediction	No	No	Yes
Entropy Coding	VLC	VLC	CAVLC, CABAC

Table 2.1: Comparison of video coding standards

In the case of intra-prediction H.264/AVC uses several intra-prediction modes that results in better intra-compression than the DC-prediction of MPEG-2 and the prediction of transformed coefficients of MPEG-4-Visual.

H.264/AVC also includes a mandatory in-loop deblocking filter that is not available in MPEG-2 and MPEG-4 and is optional in H.263.

H.264/AVC includes a binary arithmetic coder (CABAC) which is more powerful than the traditional entropy coders based on Variable Length Coding (VLC) of previous standards.

For the transformation stage H.264/AVC has an adaptive transform size (4×4 and 8×8) and an integer transform that is simpler to implement compared to the fractional transform of previous standards.

From the point of view of performance and parallelization the new coding tools of H.264/AVC have two main consequences. First, the availability of multiple block sizes and kernel operation modes requires the insertion of control code to detect, at runtime, the final operation that has to be applied. This reduces the efficiency of data-level parallelization. And second, some prediction techniques create multiple levels of data dependencies. At the level of MBs, some kernels uses data from neighbor MBs, creating a data dependency that reduces the opportunities for data-level parallelization. In some kernels such as entropy decoding, there are dependencies at the bit-level that inhibit fine-grain data-level parallelization.

2.5 Characteristics of Video Decoding Applications

From the previous analysis of the internal algorithms of the H.264/AVC decoder it is possible to extract some intrinsic characteristics of this type of applications that make them different from other application domains and that are required for making architectural optimizations.

2.5.1 Real-time Operation

The most common application of video decoding is to display motion video on a screen. The underlying architecture should provide the required performance to decode and display a fixed amount of frames per second.

2.5.2 Integer Small Data Types with Saturating Arithmetic

Video decoding applications use small integer data types. Pixel information is usually represented with 8-bits for consumer applications and 10 or 12-bit for professional applications. Arithmetic operations of video filters and transforms are done with integer 16-bit arithmetic, and finally, the results are converted back to 8-bit precision.

Most of the operations use saturating arithmetic. In saturation arithmetic the result of an arithmetic operation that exceeds the maximum representation of the binary base the result is set to the maximum value (and a similar process is applied for underflow).

2.5.3 Block Processing

In H.264/AVC the basic processing unit is the MacroBlock which in turn, can be divided into smaller blocks down to 4×4 pixels. Sub-block size defines the maximum data-level parallelism that can be extracted from a particular kernel and influences the memory access pattern of the application.

2.5.4 Heterogeneous Kernels

As shown in Figure 2.2 a video codec consists of a set kernels applied applied to the input bitstream. These kernels can be classified in three main groups[214]:

Highly Data-parallel Operations

The deblocking filter, luma and chroma interpolations for Motion Compensation, and IDCT fit in this category. They consist of a sequence of operations that have to be applied to a complete block of data. Because of that, they fit well for architectures that support fine-grain DLP.

Bit Serial Operations

This type of operations is common in the entropy decoding and parsing stages. Operations of this type are applied to a bitstream bit by bit with data and control dependencies between them. DLP is not applicable, but ILP can be exploited to some extent.

Control Operations

H.264/AVC have a lot of operation modes. Before applying the highly data-parallel kernels it is necessary to detect or decide many coding options. Also it is necessary to set-up the data structures used for storing compressed and uncompressed video elements, and to send information to the input and output devices. Usually, this type of operations fit very well on a general purpose processor running an operating system.

Data Movement and Formatting Operations

In order to apply the different kernels data has to be moved from main memory to the different processing units. Some processing units require the data to be in a specific format requiring a pre- or post-formatting stage. In some cases, Memory Transfer Parallelism can be exploited by overlapping memory operations with computations.

2.5.5 Hierarchy of Data Dependencies

As shown in Figure 2.3 a compressed video sequence is composed by a hierarchy of data elements. At each level in the data hierarchy there are different data dependencies. At the frame-level, for example, P- and B-frames have data dependencies with their respective reference frames. At the macroblock level, motion vector prediction, intra-prediction and deblocking filtering require data from neighbor macroblocks. As a consequence, the parallelization at this granularity requires to handle this type of dependencies.

2.6 Summary

In this chapter we have introduced the importance of video compression for allowing the storage and transmission of digital video signals. The objective of a video codec has been defined as a tradeoff between the reduction of bitrate and maintaining high quality and low latency and low complexity. The structure of the block-based hybrid video codec has been presented. Prediction, Transformation and Entropy decoding were presented as the main stages of the video codec. Taking this structure as a reference, the main kernels of the H.264/AVC video codec were described, including a comparison with previous video codecs like MPEG-2 and MPEG-4. Finally, we have derived some general characteristics of the video decoding applications that will be used to present the different architectural optimizations in the rest of this work.

3 Architectures for Video Decoding

This chapter presents a description of the existing work on computer architectures for video decoding. Most of the techniques and architectures that are going to be presented are applicable for most "hybrid video CODECs" but we concentrate on those that are applicable for H.264/AVC decoding.

Architectures are classified by the degree of programmability in three groups: dedicated hardware architectures, media processors and general purpose processors [66, 131]. And they are evaluated in terms of its performance, scalability, efficiency and flexibility. Because some of these objectives can not fulfilled simultaneously, the design and evaluation of architectures for video decoding results is a complex tradeoff between them.

3.1 Dedicated Hardware Architectures

Dedicated hardware architectures, also called Application Specific Integrated Circuits (ASICs), for video decoding consist of a direct mapping of a particular video CODEC into specific hardware circuits. Such architectures offer the performance required by a specific real-time target (like FHDp25) at the maximum efficiency. In the one hand, due to its specific nature, they use less area and have a very low power consumption; making them a good solution for low power and mobile devices. But, on the other hand, they have a poor scalability and minimal flexibility. Low scalability is the result of being designed to support a maximum real-time performance: when the application demands more performance, for example going to a higher definition, they can not scale or need to be redesigned. The other one is the minimal flexibility: hardware modules are extremely application specific and they can not support extensions or new video CODECs; or for supporting multiple formats some hardware units have to be replicated with the corresponding loss in efficiency.

Most of the proposed solutions are based on a streaming architecture [138] that includes hardware modules optimized for each type of kernel in the video decoder. Usually a control processor is included for high level syntax parsing and general control tasks. A complete system also includes a memory architecture and an interconnection architecture. Those are also optimized to the specific data sets and communication patterns of video decoding applications [37].

A survey of dedicated architectures for previous video CODECs, like MPEG-1, MPEG-2 and MPEG-4, can be found in [188, 186, 187]. Below, we present a general description of some works on dedicated hardware architectures for H.264/AVC decoding.

A typical example of this type of architectures is presented by Lin et al. [150]. They target H.264/AVC decoding for FHD input videos at 30 fps. The architecture includes 4 hardware modules: one for bitstream decoding, one for inverse quantization and inverse transform, one for (both intra- and inter-) prediction, and another one for the deblocking filter. The system also includes a DMA engine with a special purpose buffer and two

external memory interfaces. The on-chip interconnection is done with a hierarchical Advanced Microcontroller Bus Architecture (AMBA) bus and a special purpose internal bus. A general view of this architecture is depicted in Figure 3.1. This system is able to provide the required real-time performance operating at 120 MHz with a power consumption of 320 mW [150].

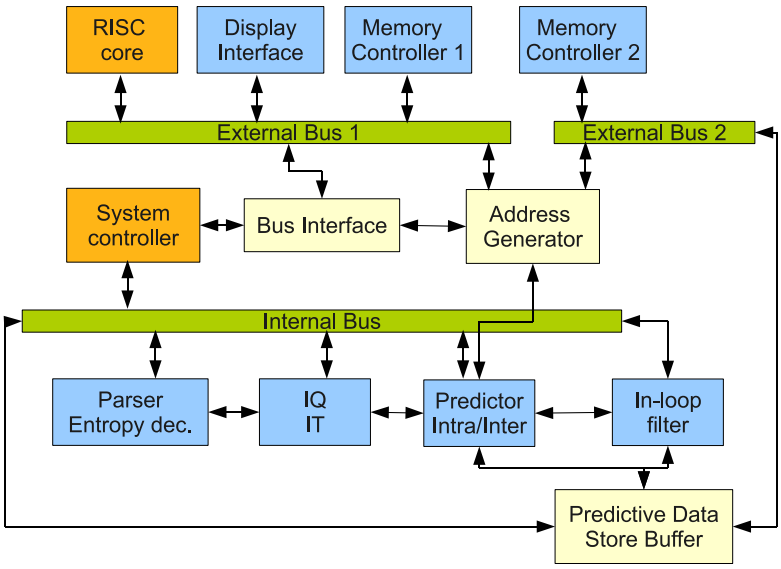


Figure 3.1: General diagram of a H.264 hardware decoder

Table 3.1 presents a comparison of different dedicated architectures for H.264/AVC decoding, including parameters like technology, frequency, power and area. All of them are based on the same heterogeneous streaming architecture but differ in the hardware optimizations that have been applied. From the published results it can be noted that dedicated hardware architectures can provide the performance for FHDp30 real-time decoding with a power consumption less than 350mW and an area less than $10mm^2$.

It is important to mention that some dedicated architectures also include programmable processors optimized for specific kernels that allows to adapt the system for different video CODECs but with a limited programmability [120]. The cost of this flexibility is a bigger area and power consumption.

Architecture	Technology (nm)	Voltage (V)	Frequency (MHz)	Power (mW)	Area (mm^2)	Video Capabilities
Hu et al.[97]	130	1.2	200	160	n.a.	$2048 \times 1024p30$
Chien et al.[49]	130	1.2	120	71	5.04	$1920 \times 1080p30$
Lin et al.[150]	180	1.8	120	320	8.41	$1920 \times 1080p30$
Liu et al.[153]	180	1.8	16.6	12.4	15.21	$720 \times 576p30$
Sze et al.[235]	65	0.7	14-25	1.8-3.2	7.62	$720 \times 576p30$
Kimura et al.[120]	65	1.2	162	342	29.7	$1920 \times 1080p30$

Table 3.1: Comparison of hardware architectures for H.264/AVC decoding

3.2 Multimedia Processors

Multimedia processors (also known as media-processors) are programmable processors that have been designed to address the requirements of multimedia applications specially video coding and decoding. They have an ISA and a memory architecture optimized for video applications.

Most current media-processors use a Very Long Instruction Word (VLIW) architecture [78] that allows to exploit ILP with a more efficient hardware implementation that superscalar processors, making them suitable for low power and low cost devices. The programmability of media processors give them a flexibility advantage compared to dedicated hardware solutions while, at the same time, they offer a high efficiency in terms of area and power consumption.

Two examples of VLIW-based media-processors are Texas Instruments' VelociTI architecture [216] and Phillips TriMedia architecture [194].

Current version of the VelociTI architecture is called TMS320C6X for fixed point processors. This architecture includes two datapaths each of which contains 32 32-bit registers for a total of 64 registers. The number of functional units per datapath is implementation dependent, going up to 8 in total in the most recent processors. The processor can execute 8 32-bit instructions per cycle and support μSIMD operations on 8- and 16-bit operands. The architecture has a two-level memory hierarchy composed of level-1 instruction and data caches with support for unaligned accesses and a shared level-2 cache. It also features a multichannel DMA controller for handling on-chip data transfers between the L2 cache and the peripherals [216, 2]. Reported results show that with a TI VLIW processor running at 160 MHz it was possible to decode MPEG-4 CIF resolution videos at 55 fps [271].

The TM3270 is a media processor based on the TriMedia architecture [194] designed for high performance video decoding. The architecture support five operations per VLIW instruction. The ISA include special operations like "collapsed load with interpolation" that implements interpolation filters in the load operation; and it also includes special instructions for doing CABAC entropy decoding. In addition, it includes optimized prefetching mechanisms adapted for 2D-memory accesses [252]. Published results show that the TM3270 processor running at 33.5 MHz can encode MPEG-2 CIF videos at 25 fps [251].

Table 3.2 presents a comparison of two media processors from the VelociTI and TriMedia architectures.

Architecture/ Processor	Technology (nm)	Frequency (MHz)	Power (mW)	Area (mm^2)	Instructions per cycle	Register file
TI TMS320C6414 [2]	130	600	718	n.a	8	2 x (32 32-bit)
Phillips TM3270 [252]	90	450	n.a.	8.08	5	1 x (128 32-bit)

Table 3.2: Comparison of VLIW-based media-processors

3.3 General Purpose Processors (GPPs)

General Purpose Processors (GPPs) are in the other extreme compared to dedicated hardware solutions. They have higher flexibility that results from the software implementations. Performance usually is very high as a result of the high frequency of operation and extensive support for ILP. The main limitation is the low efficiency in terms of high power consumption and big area. In this section we review the techniques applied to support video decoding applications on GPPs including SIMD extensions and thread-level parallelism.

3.3.1 SIMD Extensions

After the release of the MPEG-1 video CODEC, a lot of interest was devoted to the development of portable software-only video decoders. With the workstations available at that time it was not possible to decode low resolution MPEG-1 videos in real-time. For example, with the HP-750 workstation equipped with the PA-7000 processor running at 66 MHz, it was only possible to decode a CIF resolution (352x288p25) MPEG-1 sequence at 15 fps [157].

In order to increase the performance of video decoding on GPPs designers started to look at hardware and software optimizations. One of the inefficiencies that was detected was that when processing video data stored on the wide registers of workstations (32- or 64-bit) the ALUs perform useless operations on part of the data. As a way to overcome this limitation processor designers included a limited form of SIMD processing in GPPs pipelines. The integer data path was modified to support the simultaneous execution of the same operation on different data inside a register, this technique is sometimes called SIMD within a register or μSIMD.

In the classical definition of SIMD computing there is a control processor that issues instructions; several data processors that execute the same instruction in parallel; and a parallel memory unit that support multiple memory accesses at the same time [80]. In μSIMD, the control processor is composed by the fetch, decode and dispatch units of the processor; the data processors are partitions of the functional units; and the parallel memory is composed by the regular load and store units which transfer complete words containing multiple subwords. The changes to the processor were inexpensive because the same functional units were used with a simple modification to avoid the propagation of carry signals [140].

By including μSIMD extensions to a base ISA, it was possible, in 1994, to decode MPEG-1 streams in real-time using a software-only decoder. A report shows that a workstation based on the PA-RISC architecture (HP-712) equipped with the PA-7100LC processor running at 80 MHz was able to decode CIF MPEG-1 videos at 25 fps [30, 143].

Comparison of SIMD Extensions for GPPs

Multimedia extensions on the PA-RISC architecture were the starting point of the now common SIMD extensions for general purpose and embedded computing platforms. That includes (in a non-exhaustive list): MAX-1 and MAX-2 (Multimedia Acceleration eXtensions) to the mentioned PA-RISC architecture [140], VIS (Visual Instruction Set) to the SPARC architecture [248], MVI (Motion Video Instructions) to the Alpha architecture [40], MMX to the X86 (IA-32) architecture [181], MDMX (MIPS Digital Media

Extension	Base ISA	Vendor	Release Date	Instructions	Register file
MAX	PA-RISC	HP	1994	9	Integer (31x32b)
VIS	SPARC	Sun	1995	121	FP (32x64b)
MAX-2	PA-RISC	HP	1995	8	Integer (32x64b)
MVI	Alpha	DEC	1996	13	Integer (31x64b)
MMX	X86	Intel	1996	56	FP (8x64b)
MDMX	MIPS-V	MIPS	1996	74	FP (32x64b)
3DNow!	X86	AMD	1998	21	FP (8x64b)
Altivec/VMX	PowerPC	Motorola/IBM	1998	162	32x128b
MIPS-3D	MIPS-64	MIPS	1999	13	FP (32x64b)
SSE	X86	Intel	1999	70	8x128b
SSE2	X86	Intel	2000	144	8x128b
SSE3	X86	Intel	2004	13	8x128b
NEON	ARMv7	ARM	2005	88	32x64b (16x128b)
SPU	Cell	IBM/Sony/Toshiba	2005	??	32x128b
SSSE3	X86	Intel	2006	16	8x128b
SSE4	X86	Intel	2007	54	8x128b
VSX	Power v2.06	IBM	2010	143	64x128b
AVX	x86 + SSE4	Intel	2011	12	16x256b

Table 3.3: Comparison of SIMD Extensions to General Purpose and Embedded Processors [227]

eXtension) to the MIPS architecture [222], Altivec to the PowerPC architecture [71], 3DNow to the x86 architecture [175], SSE and its revisions (SSE2, SSE3, SSSE3, SSE4) to the X86 architecture [246, 191], Advanced SIMD extensions (NEON) to the ARM Architecture, Vector-Scalar Extension (VSX) to the Power architecture [101] and Advanced Vector Extensions (AVX) to the x86-64 architecture [63].

In Table 3.3, a comparison of different SIMD extensions is presented. That includes the base ISA in which they are implemented, the year of introduction, the number of new instructions and the type and size of the SIMD register file (number of registers $\times$ size of each register). This is an updated version of the table presented by Slingerland and Smith in 2000 [227].

Performance of SIMD Extensions for Video Coding and Decoding Applications

The main effect of using SIMD extensions in video decoding (and other) applications is a reduction in the number of arithmetic, branch and memory instructions [193]. The reduction in the number of arithmetic operations is the result of processing multiple data with a single instruction. The reduction in branch instructions is the effect of loop unrolling in which multiple iterations of a loop are replaced by a single iteration that processes multiple data in one instruction. The reduction of memory instructions is the effect of transferring multiple small data size operands with a single load or store. Figure 3.2a shows an example of a simple code with 2 loops that exhibits straightforward DLP. Figure 3.2b shows the same example using μSIMD instructions using an architecture with 64-bit registers and 16-bit operands. Figure 3.3 shows this operation graphically. In the μSIMD version the inner loop is replaced with a single instruction that perform 4 operations in parallel.

The first works on μSIMD extensions used small kernels to test the new instructions. That includes the optimization of kernels like: FIR (Finite Impulse Response) filters, DCT transforms and block matching. Some evaluations were performed for Altivec[171],

```
for (i=0; i<4; i++){
  for (j=0; j<4; j++){
    d[i][j]= c[i][j] + a[i];
  }
}
```

(a) Scalar version

```
for (i=0; i<4; i++){
        tmp_a = simd_load_and_replicate(a[i]);
        tmp_c = simd_load(c[i]);
        tmp_d = simd_add(tmp_a, tmp_c);
        simd_store(tmp_d, d[i]);
}
```

(b) SIMD version

Figure 3.2: Addition of a vector to a matrix: example of a kernel that exhibits DLP

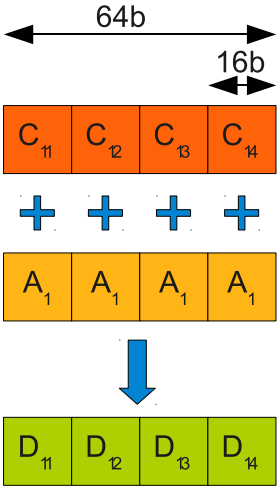


Figure 3.3: Sample Operation using μSIMD instructions

MAX [144] and MMX [29, 238] exhibiting speedups in the range of 1.5 to 12 depending on the architecture.

Some time later the first works appeared with the evaluation of complete applications. They show the real effect of using μSIMD extensions at the cost of a higher programming complexity [132]. These works present the optimization of different video CODECs using different μSIMD extensions. By using μSIMD extensions it was possible to perform real-time decoding video decoding in software for MPEG-1, MPEG-2, MPEG-4 and similar video CODECs for low to medium resolution videos. Table 3.4 presents a summary of the works on μSIMD extensions for video CODECs before H.264/AVC, it includes the resolution of the input videos, processor operating frequency and resulting frame rate. A complete review of these works is presented by Lappalainen et al. [132].

Application	Year	ISA	Resolution	Processor	Frequency [MHz]	Frame rate [fps]
MPEG-1 dec. [157]	1993	no-SIMD	CIF	PA-7000	66	15
MPEG-1 dec. [30]	1995	MAX-1	SIF	PA-7100	80	33
MPEG-2 dec. [270]	1995	VIS	720x480	Ultra SPARC	n.a	30
H.261 dec. [164]	1996	VIS	CIF	Ultra-1	167	60-243
MPEG-1 dec. [107]	1997	MMX	n.a.	Pentium-II	200	115
MPEG-2 dec. [107]	1997	MMX	n.a.	Pentium-II	200	25
MPEG-2 dec. [68]	1999	VIS	n.a.	Ultra SPARC	360	15.6 - 25
MPEG-1 dec. [250]	1999	MMX	n.a	Pentium-II	200	72
MPEG-2 dec. [250]	1999	MMX	n.a	Pentium-II	200	20
MPEG-4 dec. [41]	1999	MMX	CIF	Pentium-II	266	50-62
H.263 enc. [73]	2000	MMX	QCIF	Pentium-II	200	14-17
MPEG-4 codec [163]	1999	MMX	CIF	Pentium-II	450	30
H.263 enc. [4]	2000	MMX	QCIF	Pentium-II	233	45.68
H.263 enc. [4]	2000	VIS	QCIF	Ultra SPARC	167	12.15

Table 3.4: Performance of video coding and decoding applications with μSIMD optimizations [132]

Application	Year	ISA	Processor	Frequency [GHz]	Resolution	Frame rate [fps]	Speedup
H.264 decoder [272]	2003	SSE2	Pentium-IV	2.4	720×480	48	2.5–3
H.264 decoder [139]	2004	MMX	Pentium-IV	n.a	n.a	n.a	1.26
	2004	SSE3	Pentium-IV	3.4	1280×720	60	n.a.
H.264 decoder [111]	2004	SSE2	Pentium-M	1.7 GHz	1280×720	30	n.a.
	2004	WMMX	PXA270	0.62	352×288	30	n.a.
H.264 decoder [221]	2006	SSE2	Pentium-IV	2.8	352×288	n.a.	1.496

Table 3.5: Performance of H.264 decoder with μSIMD optimizations

SIMD Extensions for H.264/AVC Coding and Decoding

H.264/AVC at High Definition requires more computing performance than previous codecs and the first works were dedicated to analyze the optimization level that can be obtained using μSIMD instructions. In this section we present a review of the most relevant ones. Table 3.5 summarizes their main characteristics.

Zhou et al. present an optimization of the H.264 decoder using SSE2 instructions on a Pentium-IV processor running at 2.4 GHz. They used three different input videos in QCIF, CIF and SD resolutions. The optimized kernels were luma and chroma interpolation, IDCT and deblocking filter. Reported speedups were: 2.9 for luma MC, 10.2 for chroma MC, 4.3 for IDCT and 1.1 for DF. The complete application speedup ranges from 2.0 to 4.0, depending on coding options and input sequences [272, 44].

Lee et al. present an evaluation of the H.264 decoder using MMX instructions on a Pentium-IV processor. They optimized the Luma interpolation, deblocking filter and IDCT with speed-ups of 1.93, 1.91 and 5.08 respectively [139]. There are not details of the input sequences.

Iverson et al. present an evaluation on the H.264 encoder and decoder of three different Intel processors: Pentium-IV, Pentium-M and PXA270. Optimized kernels are Luma interpolation, IDCT and deblocking filter. The decoder only support the baseline profile of the standard which do not include support for B-frames, CABAC and other important coding tools. Taking this into account, the decoder is able to reach real-time processing

for different resolutions: 60 fps for HD videos on the Pentium-IV running at 2.8 GHz and 30 fps for CIF videos on the embedded PXA270 running at 403 MHz [111].

Shojania et al. present μSIMD optimizations to the deblocking filter of the H.264 decoder on a Pentium-IV machine running at 2.8 GHz. They report a speedup of 1.5 at the kernel level using CIF input videos.

Limitations of SIMD Extensions

Although the potential performance improvements of SIMD processing within a register are high (a speedup of 8 for 16-bit arithmetic on an architecture with 128-bit registers) the actual performance is considerably below that maximum point. Some experiments reported less than 2X performance improvement for the MPEG-2 decoder using VIS instructions [193]. Or sometimes there is a considerable speedup in inner kernels but not noticeable speedup in the complete application [29].

This is the result of limitations in the application, data layout, the architecture or the implementation. Limitations in the application include the existence of kernels with limited or nonexistent DLP, for example parsing of bitstreams and entropy decoding. Limitations in the data layout are usually related with the overhead of data rearrangements, for example: packing, unpacking, alignment, transposing, etc. Limitations in the ISA are the related to not having enough parallelism (in 64-bit register files), reduced number of registers (in MMX) or the lack of support for certain data types (8-bit processing in MAX and VIS). Finally, limitations in the implementation include the lack of functional units, limitations in the data paths, scheduling of SIMD instructions or memory bandwidth.

As a way to overcome those limitations different architectural techniques have been proposed. In this section we review some of them, including vector and streaming processors.

3.3.2 Vector Processors

Vector processors have been used for many years mostly in the supercomputing domain for exploiting the parallelism available in scientific applications. But also they have been proposed as an effective way of exploiting DLP in multimedia applications [75].

Conventional Vector Processors

In a conventional vector processor a single vector instruction specifies multiple independent operation on a linear array of data operands [180]. Vector instructions can be used to exploit data-level parallelism executing multiple data elements simultaneously. μSIMD can be seen as a limited form of vector processing with short and fixed length vectors.

Figure 3.4 shows an example of the use of a conventional vector architecture for optimizing the code presented in Figure 3.2. The architecture uses 4 64-bit registers to process the inner loop of the code with a single vector instruction. It is notorious that a vector architecture does not use efficiently its long registers when processing small data size operands.

One example of vector architecture proposed for multimedia applications is the *VIRAM*.

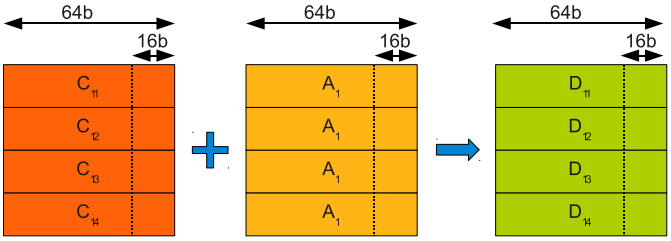


Figure 3.4: Sample DLP operation using vector instructions

VIRAM is a vector microprocessor that combines vector processing with DRAM modules on a single chip oriented to multimedia applications. It contains a scalar MIPS core extended with a Vector Unit (VU). The VU includes a vector register file (VRF), some Vector arithmetic Functional Units (VFUs) and a vector memory unit. The VRF has 32 registers each one containing 32 64-bit elements. The architecture supports μSIMD allowing to process 64 32-bit elements or 128 16-bit elements per vector register. An implementation of the processor includes two VFUs, 4 lanes and a vector load-store unit with four address generators connected to a multi-bank on-chip DRAM memory [129]. An evaluation of the H.263 encoder using a cycle-accurate simulator of the VIRAM architecture has been reported [172]. That includes an optimization of the motion estimation and the IDCT kernels for which speedups of 5.88 and 8.7 over a Pentium-II with MMX has been obtained respectively. An average encoding performance of 22.5 fps is obtained for QCIF videos on a system running at 200 MHz.

Multidimensional Vector Processors

An alternative approach to traditional vector processors and μSIMD comes from the combination of vector registers and sub-word computation in such a way that registers can be seen as matrices [56]. One of such approaches is called Matrix Oriented Multimedia (MOM).

Figure 3.5 shows how MOM ISAs exploit DLP using the simple code example of Figure 3.2. μSIMD ISAs (show in Figure 3.3) perform multiple sub-word operations within a single register. Vector processors (shown in Figure 3.4) perform multiple operations by using long vectors. MOM (shown in Figure 3.5) combines both approaches using vector registers with subword capabilities.

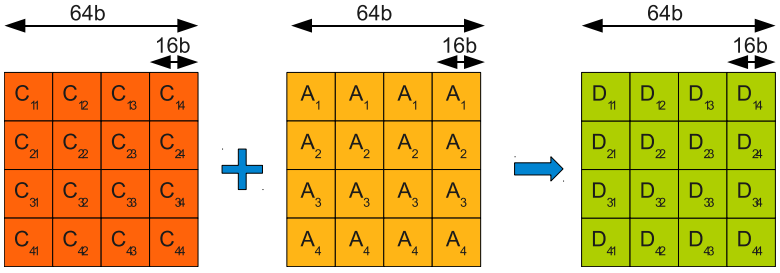


Figure 3.5: Example of DLP operation using a matrix instruction

MOM architecture includes a matrix register file, some matrix functional units and

a vector memory unit. MOM register file includes 16 registers, each one composed of 16 64-bit elements. Each element within a matrix register supports μSIMD with 32 32-bit and 64 16-bit sub-elements. MOM functional units operate on whole matrices and support reductions and reorganization operations [57]. Memory organization use a high bandwidth vector cache that bypass the L1 data cache. Evaluations of MOM on a cycle-accurate simulator for a MPEG-2 encoder and decoder shows speedups of 4.3 and 3.5 respectively compared to a scalar processor with μSIMD [56].

Another approach that removes some limitations of vector and μSIMD architectures is proposed on the Complex Streamed Instructions architecture (CSI). CSI is a memory-to-memory architecture that supports two-dimensional data streams of arbitrary length [115]. In CSI hardware is responsible for dividing the data streams into sections which are processed in parallel. There are not internal registers and data is read and written directly to memory. Finally data conversion and rearrangement is performed automatically by hardware avoiding overhead instructions. A cycle-accurate simulation evaluation shows speedups of 1.40 and 1.93 for the MPEG-2 encoder and decoder respectively (using QCIF input videos) compared to a processor with the VIS SIMD extension.

3.4 Chip Multiprocessor (CMP) Architectures

The performance required by real-time video decoding has been usually bigger that the performance offered by the general purpose architectures at any given point in time [158]. An alternative to obtain the required performance has been the use of parallel computers. In the past, parallel systems were scarce, available only in high performance or supercomputing centers and used mainly for computational science and engineering. With the trend towards Chip MultiProcessors (CMPs) the availability of parallel computers has increased and it is expected that almost all computers will be parallel computers in the near future [21].

In the last years, many different CMP architectures have appeared for different application domains such as embedded systems, desktop applications, and server computing. The architecture of those multicores is diverse and depends on many factors like the application domain, architecture of each core, memory organization, interconnect architecture and power consumption. In this section, a review of some existing multicore architectures is presented covering different application scenarios and architecture styles.

3.4.1 General Purpose Multicores

In desktop, server and some embedded applications it is necessary to offer high performance for single core applications. This is achieved with multicore architectures based on a reduced number of high performance cores. Typically they are homogeneous (i.e. all the cores have the same ISA and microarchitecture) and each one includes extensive support for ILP, uses a memory hierarchy with many levels of cache, implements a shared memory model with cache coherency and consume a lot of power (more than 100W).

One example of a general purpose multicore is the IBM Power7 [118]. It includes 8 cores each one capable of executing four threads in Simultaneous Multi-Threading (SMT) fashion for a total of 32 hardware threads executing in parallel. Each core has

Architecture	ISA	Total cores	Total threads	Freq. $[GHz]$	Power $[W]$	Tech. $[nm]$	Transist. $[\times 10^9]$	Area $[mm^2]$
Intel Core Duo	X86	2	2	2.33	31	65	151	n.a.
Intel Core 2 Quad	X86-64	4	4	2.66	105	65	582	286
Intel Core i7-970 [64]	X86-64	6	12	3.2	130	32	n.a.	n.a.
AMD Phenom II X6 [17]	X86-64	6	6	3.2	125	45	n.a.	n.a.
IBM Power4 [243]	Power	2	2	1.3	115	180	174	n.a.
IBM Power5 [117]	Power	2	4	1.5	n.a	130	276	389
IBM Power6 [135]	Power	2	4	4.2	n.a	65	790	341
IBM Power7 [118]	Power	8	32	3.5	n.a	45	1200	567
Sun UltraSPARC-T1 [125]	SPARC-V9	8	32	1.4	72	90	279	378
Sun UltraSPARC-T2 [170]	SPARC-V9	8	64	1.4	84	65	503	342
Sun UltraSPARC-T3 [220]	SPARC-V9	16	128	1.65	139	40	$\tilde{1}000$	371
ARM Cortex-A15 [18]	ARMv7	4	4	2.5	n.a.	n.a.	n.a.	n.a.

Table 3.6: General purpose multicore architectures

extensive ILP support in the form of out-of-order execution and branch prediction; each core being able of decoding, issuing and executing eight instructions per cycle. On the memory side, each core includes a small fast private L1 cache, a private L2 cache and all the cores share a big L3 cache built with eDRAM technology. The chip also includes two DDR3 memory controllers, each one supporting 4 channels, providing a total of 100 GB/s of memory bandwidth. This multicore implements a shared memory system with coherent caches using a broadcast mechanism.

Table 3.6 shows a selection of some of general purpose multicores. This is an updated version of the table presented by Blake et al. [32]. As an example of technological evolution, the IBM Power architecture are depicted: multicore architectures start with Power4 that has 2 cores, Power5 included two cores with 2-way SMT, Power6 has the same number of cores but features a higher frequency per core and finally Power7 includes 8 cores each capable of 4-way SMT.

3.4.2 Heterogeneous Media-processors

A common strategy used in embedded systems consist on including a heterogeneous mixture of programmable and dedicated processing units. Those architectures usually include one or more RISC processors for control tasks, one or more media processors (usually a VLIW optimized for media applications) for flexible media processing and some dedicated hardware modules for specific video processing tasks. These architectures provide higher flexibility in terms of programmability, have a reduced area and power consumption compared to general purpose multicores, but the main limitation is the programming of such heterogeneous environments.

Some examples of these architectures are the Open Multimedia Application Platform (OMAP) platform from Texas Instruments [42], the Nexperia platform from NXP [124] and the Nomadik platform from ST Microelectronics [179]. The OMAP platform is oriented to high performance mobile multimedia applications and combine general purpose processors with media processors and hardware accelerators. A recent example of this platform is the OMAP4440 application processor that consist of a dual-core ARM Cortex A9 processor, a VLIW media processor based on the TI TMS320C64X architec-

ture, a set of hardware modules for accelerating video encoding and decoding and an accelerator for 3D graphics among other peripherals [109].

Architecture Processor	Technology (nm)	Frequency (MHz)	Power (W)	Control Cores	Media Cores
TI OMAP4440 [109]	45	1000	n.a.	2 ARM Cortex-A9	1 C64X
NXP TV550 [124]	45	500	0.7	1 MIPS-32	2 TM3260
STM STn8820 [179]	65	576	n.a	1 ARM-11	n.a.

Table 3.7: Heterogeneous media-processors

Table 3.7 shows some details of three heterogeneous media architectures. It is important to note that the NXP platform is used for set-top-box applications and the TI OMAP and ST Nomadik ones are used for mobile devices, which have different design requirements.

3.4.3 Graphics Processing Units: GPUs

GPUs are CMP architectures specialized on video games and graphics processing, and more recently they have been used for General Purpose computing in what has been called GPGPU. GPUs are designed as floating-point numerical intensive architectures and are optimized for execution throughput of a massive number of threads. In order to take advantage from its huge computing capabilities applications must exhibit massive and regular data-level parallelism.

A programmable GPU contains a hierarchical organization of many simple cores connected to a hierarchical set of memories. Using the terminology defined by the OpenCL standard we describe a general architecture of a GPU [87]. A heterogeneous computing system is composed of one *host* and one or more GPUs called Compute Devices (CD). Each CD is composed of one or more Compute Units (CUs). Each CU is is further divided into one or more Processing Elements (PEs). Each PE has access to a small and fast private memory. All the PEs in a CU have access to a local memory and all the CUs in the CD have access to a global memory and a constant memory. The *host* memory is located in a separated address space. Figure 3.6 shows this conceptual view of a GPU architecture [121].

All the PEs on a CU execute the same instruction sequence on different data items in what is known as Single Program Multiple Data (SPMD) mode. Different CUs can execute different instruction sequences. In GPUs memory management is explicit, which means that the programmer must move data from host to global memory, and from there to local memory before performing computations. After calculations data should be moved back to the host memory.

Current GPUs contain hundreds or thousands of single processors, and provide hundreds of Gigabytes per second of memory bandwidth. Some examples of high performance GPUs are the NVIDIA GeForce series and ATI/AMD Radeon series. Table 3.8 presents some features of a selection of modern GPUs.

GPUs for Video Decoding

There are two ways of using GPUs for video decoding: using fixed-function hardware modules or using programmable SIMD cores.

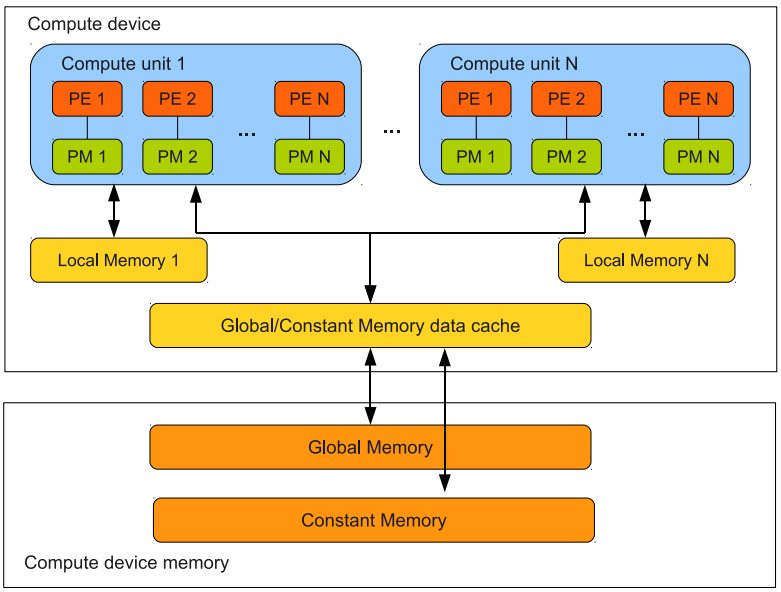


Figure 3.6: OpenCL generic view of a GPU architecture

Architecture	ISA	Total cores	Compute Units	Cores/ CU	Freq. [GHz]	Power [W]	Tech. [nm]
Nvidia GTX480 [60]	n.a.	480	15	32	1.4	250	40
AMD Radeon HD 5870 [58]	VLIW	320	20	16	0.85	188	40

Table 3.8: Graphic Processing Units (GPUs)

Most GPUs architectures include hardware modules for performing decoding of the most common video CODECs such as MPEG-1, MPEG-2, MPEG-4 ASP, H.264/AVC and VC-1. When using this approach the main application is executed on the CPU and it offloads parts of the video decoding process to the hardware modules on the GPU. In order to use this model the video decoder has to use a platform specific API that allows the main program to communicate with the GPU. Some examples of these APIs are NVIDIA's Video Decode and Presentation API for Unix (VDPAU) [61], AMD's X-Video Bitstream Acceleration (XvBA) [59] and Intel's Video Acceleration API (VA API) [82]. These APIs work at different granularity. One of them is at the kernel-level which allows the CPU to offload specific kernels like entropy decoding, motion compensation, inverse transform or deblocking filter. The other one is at the slice level, meaning that the CPU is just responsible for parsing and demultiplexing the container format and parsing the bitstream headers and then the GPU is responsible for the complete decoding of the slice. The main limitation of this approach is that it requires hardware modules for each video CODEC. Support for new formats require a new GPU design.

The second approach is the use of the GPU as a programmable parallel coprocessor. This requires the use of an API that allow the main program running on the host CPU to access the computational resources on the GPU. The most widely used APIs for parallel programming on GPUs are Compute Unified Device Architecture (CUDA) [60] and OpenCL [87]. Both of them share the same principles of heterogeneous comput-

Architecture	ISA	Total cores	Freq. [GHz]	Power [W]	Tech. [nm]	Cache	
						L1 [KB]	L2 [KB]
Tilera TILE-64 [28]	VLIW	64	0.75	10.8	90	8i - 8d	64
IBM Cell B.E. [183]	PowerPC, SPU	9	3.2	100	90	256 (local store)	-

Table 3.9: Specialized data intensive multicores

ing on platforms with one (or more) CPU(s) running a control program and one (or more) GPU(s) executing parallel kernels. These parallel computing environments allow to program the kernels that will be executed on the GPU hardware and also provide mechanisms for controlling the sequence of execution of those kernels and for executing the memory transfer between the CPU and the GPU.

The main limitation with parallel implementation of H.264 decoding using CUDA or OpenCL is that those programming frameworks are oriented to regular data parallel applications. H.264 is highly irregular because of the use of variable block size motion compensation, multiple interpolation filter for fractional motion compensation, adaptive deblocking filter, etc. Some attempts to use GPUs for video decoding exhibit limited performance gains because of that [184]. With the evolution of GPU architectures and their corresponding programming models these limitations may disappear [121]. Apart from the decoder, GPUs has been be used to speedup the motion estimation stage of the video encoder [47].

3.4.4 Specialized Data-intensive Multicores

Some commercially available multicore architectures have been designed for data-intensive parallel applications. The main characteristics of these architectures is to have multiple simple processors with support for SIMD operations connected with a high bandwidth on-chip interconnection network. Two examples of such architectures are IBM's Cell Broadband Engine (Cell B.E.) and Tilera's TILE architecture.

The Cell Broadband Engine

Cell Broadband Engine is an architecture developed jointly by IBM, Sony and Toshiba. It targets gaming, multimedia and other data-intensive applications. It is a heterogeneous architecture composed of one general purpose core called the Power Processor Element (PPE) and 8 SIMD processors called the Synergistic Processing Elements (SPEs). The PPE executes 64-bit PowerPC instructions and is dedicated to task management for the SPEs and to run a general purpose operating system. SPEs are simple in-order machines that executes a SIMD instruction set with 16-byte vectors. SPEs do not have caches, instead they feature a local store that is not coherent with other local stores. Transfers between main memory and local stores are made via DMA operations declared explicitly by the programmer. The interconnection between SPEs, the PPE, and other on-chip elements is done with a circular ring called the Element Interconnection Bus (EIB) which has 4 16-byte channels. Each channel in the EIB support a maximum of 25.6 GB/s for a total bandwidth of 102.4 GB/s. The interconnection with the external DRAM memory is done with a dual-channel on-chip memory interface controller that

supports 25.6 GB/s [116, 122].

There are several works on implementations of H.264 decoding on the Cell processor. Different parallelization strategies have been evaluated such as function level parallelism [25] and data-level parallelism [26, 48]. Reported results show that it is possible to process up to 140 FHD frames per second using one PPE and 16 SPEs on a chip running at 3.0 GHz [48]. More information about parallelization strategies will be provided in Chapter 8.

Tilera TILE

Tilera TILE is a multicore SoC targeting high performance embedded applications in the networking and multimedia domains. The architecture consists of a 2D array of simple processors (tiles) connected by a coherent NoC interconnect. Each tile consists of a Processing Element (PE) with L1 and L2 caches and a non-blocking switch that connects the tiles to the mesh network. The PE is a simple low-power a 3-way VLIW processor with a 64-bit instruction word. The processor has access to private and fast L1 caches for data and instructions and a L2 cache; the combined L2 caches of all tiles act as distributed and coherent cache. Tiles are connected through a 2D mesh network that, unlike buses, can scale to a large number of cores [255]. Each tile includes a switch that connect to five different networks each one with five connection ports that supports 120 GB/s interconnection bandwidth per tile [28]. Tilera architecture supports configurations from 16 up to 100 cores. Table 3.9 shows some details of a Tilera based processor with 64 tiles.

A report on parallel H.264 deblocking filter for the Tilera architecture exhibits up to 12X speedup for FHD videos only in the deblocking filter compared to a sequential code and 1.40X speedup for the complete application [265]. Another work shows the performance improvement on the H.264 decoder for low resolution videos (CIF) using up to 8 cores. A maximum speedup of 6X was obtained [100].

3.5 Summary

In this chapter we have presented different architectures that are used for video decoding. We started with ASICs that are the most efficient in terms of area and power consumption but, at the same time, have the smaller flexibility and scalability. Next, we presented multimedia processors, which are programmable processors oriented to multimedia applications specially video processing. They offer better flexibility than ASICs while, at the same time, offer low power consumption. As opposite to ASICs, we presented GPPs which offer the maximum flexibility of software solutions and the highest performance in terms of high frequency and ILP support, but also have the higher area and power consumption. In GPPs the most common solution for dealing with the requirement of multimedia applications has been the use of μSIMD instructions; they were described along with their use for video decoding applications. Finally, we have presented different CMPs architectures such as homogeneous general purpose multicores, heterogeneous media processors, GPUs and data-intensive multicores.

4 Scalability of Vector ISAs for Video Coding and Decoding

As multimedia standards become more complex, processors need to scale their SIMD multimedia extensions in order to provide the performance required by new applications. Scaling these extensions not only need to address performance issues, but also power consumption, design complexity and cost, especially for embedded processors.

At the time this work was done most multimedia extensions used 64-bit length μSIMD registers. A question that emerged was: can applications benefit from going from 64-bit to 128-bit registers or from adding more functional units to the pipeline? In this chapter we answer these questions performing a scalability analysis of a 1-dimensional SIMD extension, like Intel MMX, and a 2-dimensional extension, like MOM (More details about MOM were presented in section 3.3.2). For the two kinds of SIMD extensions a scaling in the width of registers and the number of execution units was performed.

In this study, we use MPEG-2 coding and decoding because MPEG-2-video was (and, to some extend, continue to do so) the most popular video coding format at the time of this initial study. This study was done when most of the SIMD extensions used 64-bit registers but the last generation of SIMD extensions uses 256-bit registers [63]. At the end of this chapter we include a new section that reflects on recent changes in SIMD extensions.[1]

4.1 Scaling SIMD Extensions

The amount of parallelism that can be exploited using SIMD extensions is a function of three conditions. The first one is the number of SIMD instructions that can be executed in parallel, which is related with the number of SIMD functional units and the hardware resources necessary for continuous processing multiple SIMD instructions. The second one is the number of subwords that can be packed into a word, which depends on the size of registers. Packing more data into a single register allows to perform more operations in parallel for each instruction. The last one is the presence of combined instructions that allow the execution of different types of instructions (integer, floating-point, SIMD) concurrently; this condition depends on the application and the code generated by the compiler [141].

The first two techniques are related with microarchitecture and architecture features that can be modified to scale SIMD extensions. Next we are going to analyze the requirements and possibilities of implementing these techniques for scaling 1- and 2-dimensional SIMD extensions.

[1] Part of the text and results are taken from a work with a more general study on the scalability of 1 and 2-dimensional SIMD extensions, which was made in conjunction with Friman Sánchez and others. We present here only the results that are relevant for the video domain, and that were developed by the author

4.1.1 Scaling 1-Dimensional SIMD Extensions

The first approach for scaling SIMD extensions consist of adding execution units to the SIMD pipeline. The advantage of this approach is that it could improve the performance at no programming cost. But, adding more functional units to the pipeline implies an increase in register file ports, execution hardware and scheduling complexity [246]. Such modifications could have a big impact in area, timing, power and complexity of the processor.

But even if an aggressive processor with many SIMD functional units could be developed, performance gains could not be as good as expected. Some studies [239], [46] have shown that there are some bottlenecks in the microarchitecture that do not allow to obtain better performance by scaling the SIMD resources. These bottlenecks are related with overhead instructions necessary for address arithmetic, data transformation (packing, unpacking, transposing), access overhead and limitations in the issue width.

The other way of scaling is to increase the width of SIMD registers, i.e. from 64-bit registers in MMX to 128-bit (like in SSE2 and Altivec) to 256-bit, 512-bit or more. However, this option has two main disadvantages. The first one is the hardware implementation cost, that can be significant, taken into account the required increase in the width of interconnect buses, the doubling of execution units, and, more important, the increase in the memory bandwidth necessary to provide enough data to larger SIMD registers [246]. Even if such a processor could be implemented, having 256-bit or 512-bit registers could only be useful if applications have memory data patterns that match the hardware organization; that is, have enough operands arranged in memory to fill the SIMD registers. This can be true for some applications, but the majority of image, video an audio applications have small arrays or matrices sometimes non-contiguous in memory. For example the JPEG and MPEG standards define 8×8 or 16×16 pixel blocks. For this kind of applications, making registers bigger than the basic data structures may incur a big overhead for taking data from memory and/or storing back the results [134].

From the previous discussion we can conclude that a scalable SIMD extension need to provide some features in the ISA and in the microarchitecture that allow the exploitation of DLP taken into account the data patterns present in multimedia applications without increasing the complexity of critical structures in the processor.

4.1.2 Scaling 2-Dimensional Extensions

As with 1-dimensional SIMD extensions, a 2-dimensional architecture, like MOM, can be scaled in the width of registers and the number of execution units. Additionally, a vector architecture can be scaled in the number of parallel lanes and the maximum vector length.

The original MOM architecture provides the programmer with 16 matrix registers, each one holding 16 64-bit words [57]. In this chapter, we study how the vector register file in MOM architecture can scale from 64-bit width to 128-bit, adding to the original proposal more capacity and instructions. A 128-bit matrix register can hold an 8×8 16-bit matrix or a 16×16 8-bit matrix. Like other vector architectures, MOM vector load and store operations supports two basic access methods: unit stride and strided [20]. By using a strided access to memory, a matrix register can be filled with data that it is not

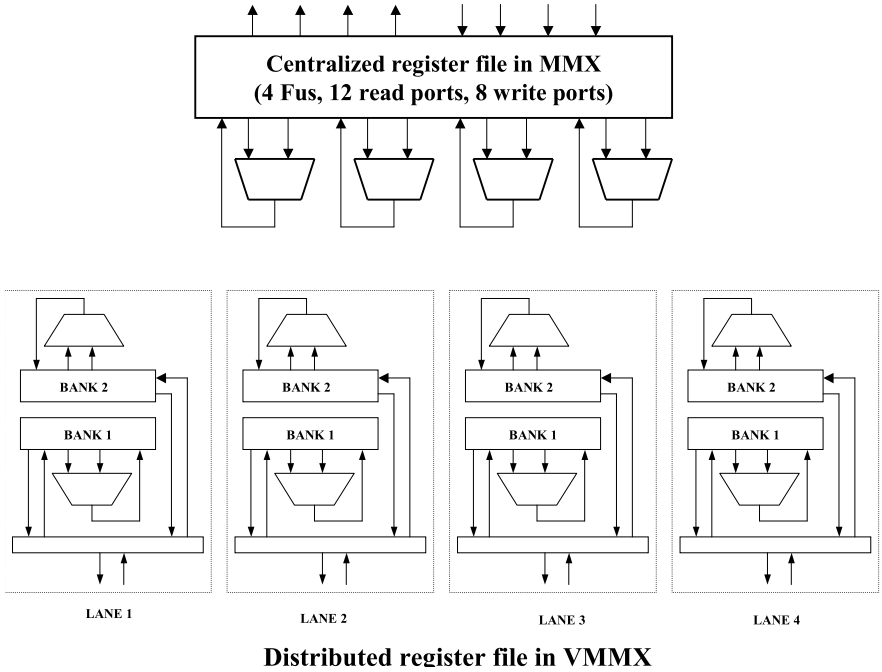


Figure 4.1: Register File and Datapath Comparison Between μSIMD (MMX) and 2D-vector (VMMX) Architectures.

adjacent in memory and with almost zero overhead for looping and address calculation. In this way, with the strided access using vector registers is possible to overcome part of the overhead associated with reorganization instructions of SIMD extensions. These instructions represent a significant part of the SIMD version of common image and video applications, in which full images or frames are divided into small blocks that are non contiguous in memory.

Multimedia applications on vector architectures are characterized by having small vector lengths [129], [206], for that reason the maximum vector length of MOM architecture is not going to be increased. As we will show in section 4.3, matrix registers of 128-bit with a maximum vector length of sixteen adapts well for most common multimedia applications.

Scaling the number of processing units is different in vector architectures than in SIMD extensions like MMX. MOM register file and datapath are organized in a distributed way in which the total register file is partitioned into banks across lanes, and these banks are connected only to certain functional units in their corresponding vector lane. Figure 4.1 shows an specific organization of two functional units divided into four vector lanes in which the register file is divided into two banks per lane. With the figure it is possible to compare the distributed vector register file to a centralized SIMD register file used in MMX. The distributed organization of the datapath in MOM provides an effective mechanism to scale performance. By adding more parallel lanes MOM can execute more operations of a vector instruction each cycle without increasing the complexity of the register file. This can be obtained by dividing the register file inside a lane into sub-banks. The limit for including more lanes is the vector length that can be achieved in the vectorization of multimedia applications [56].

Configuration	4WAY				8WAY			
	mmx64	mmx128	vmmx64	vmmx128	mmx64	mmx128	vmmx64	vmmx128
Logical regs.	32	32	16	16	32	32	16	16
Physical regs.	64	64	36	36	96	96	64	64
Lanes	1	1	4	4	1	1	4	4
Banks / Lane	1	1	2	2	1	1	4	4
Read ports / Bank	3	12	12	3	3	24	24	3
Write ports / Bank	8	8	2	2	16	16	2	2
RF storage KB	0.5	1.0	4.6	9.12	0.77	1.54	8.19	16.3
RF Area	1	2.00X	1.41X	2.63X	5.14X	10.29X	2.10X	4.20X

Table 4.1: Scaling register files for SIMD extensions. (RF area is normalized to mmx64 versions).

Additional to a bigger register file, the scaled MOM architecture includes new instructions to support partial data movement between registers and memory. These instructions are necessary for applications with data patterns that do not fill well in 128-bit matrix-registers and they are similar to those ones that were included by Intel in the SSE2 and SSE3 extensions [33].

For simplification purposes, we are going to refer to MOM architecture with 64-bit registers as VMMX64 (64-bit VectorMMX) and to MOM architecture with 128-bit registers as VMMX128 (128-bit VectorMMX).

4.1.3 Hardware Cost of Scaling

When scaling SIMD extensions, the register file storage and communication between arithmetic units become critical factors, dominating in area, cycle time and power dissipation of the processor [198]. Table 4.1 resumes the parameters of capacity, complexity and area of the registers files for a 4-way and 8-way superscalar processors with 4 different SIMD extensions. Area estimations are relative to the MMX64 configuration.

Register file area has been estimated assuming a 0.18 μ CMOS process technology based on the models described in [198]. It is very important to note that these models are just approximate and useful to give upper bounds and determine trends, but cannot be translated directly to reality, because several full custom VLSI optimizations could be done.

As Table 4.1 shows, the VMMX configurations have more capacity in the register file and can support more functional units that the MMX ones. This resource capability is reflected in terms of the necessary area for implementing VMMX extensions. The approach followed here is similar to the Altivec extension to the PowerPC architecture [71], in which a considerable silicon area is invested in the implementation of the SIMD extension in such a way that processor cycle and complexity are not affected. By using a vector organization in parallel lanes, these objectives can be achieved when scaling the VMMX extension [129].

On the other hand, when MMX64 extension is scaled to MMX128, the register file complexity becomes a predominant factor in terms of area and cycle time. Table 4.1 shows that the ratio of area increase are lower in VMMX than in MMX, so that for a 8-way processor configuration the VMMX128 register file has less area cost with less

complexity than the MMX128.

The VMMX and MMX pipelines are not balanced in terms of functional units and register file capacity, but what we want to argue is that the VMMX processors have these bigger resources because they can map effectively the hardware structure to the DLP available in video applications without a significant increment in complexity. In the VMMX configuration the number of lanes and functional units can be adjusted in order to fulfill different design constraints in terms of power and area without compromising the binary compatibility or the complexity of the register file.

4.1.4 A Case of Study: Motion Estimation

Figure 4.2 shows different versions of a fragment of code taken from the motion estimation routine that is part of the MPEG-2 encoder application. Code is taken from function *dist1* that computes the Sum of Absolute Differences (SAD) between two blocks of hx16 pixels (h is typically 8 or 16) pointed by *p1* and *p2*. There is a stride *lx* between rows.

Figure 4.2a shows the scalar version: there are two nested loops, one for intra row elements (i) and the other one for the different rows (j). In the MMX versions the inner loop is eliminated. The MMX64 version, illustrated in Figure 4.2b, operates over arrays of 1×8 pixels, being necessary to divide the data array in two regions. For each of these regions, it is necessary to use and update pointers and to accumulate the partial results of each sub-block in one register. The MMX128 version, shown in Figure 4.2d, operates over 1×16 pixels arrays that are contiguous in memory, allowing a single load to be performed for each row, and requiring less pointer overhead than the 64-bit version.

In the VMMX versions, both loops can be eliminated because it is possible to pack the two dimensional array into vector registers. For loading the data into vector registers, it is necessary to define the vector length (VL=h), and for each load, it is necessary to specify, as a part of load instruction, the vector stride *lx*. In the VMMX64 version, shown in Figure 4.2c, it is necessary to divide the array into 2 blocks of hx8 pixels, thus being necessary two vector registers to store the data array. Finally, in VMMX128, as shown in Figure 4.2e, it is possible to pack all the pixel array in a single vector register, reducing drastically the number of instructions used. Specially a lot of overhead instructions used for looping and address calculation are eliminated, and SAD is implemented using a packed accumulator that allows a parallel execution of the operation over the vector registers [55].

4.2 Experimental Methodology

4.2.1 Workload

In order to evaluate the different architectures under study we have selected two video applications from the Mediabench suite [136] which are: MPEG-2 video coding and decoding. For each application we have selected the most computational intensive kernels with potential DLP and evaluated them in isolation. Table 4.2 describe the kernels and benchmarks and their characteristics.

```
for (j=0; j<h; j++){
  for (i=0; i<16; i++){
    if ((v = p1[i] - p2[i]) < 0)
      v= -v;
    s+= v;
  }
  p1+= lx;
  p2+= lx;
}
```

(a) Scalar

```
for(j=0; j<h; j++){
  VR1 = MEM[p1];
  VR2 = MEM[p2];
  VR1 = VR1 >> 1;
  VR2 = VR2 >> 1;
  VR3 = MEM[p1+8]; p1 += lx;
  VR1 = VR1 - VR2;
  VR4 = MEM[p2+8]; p2 += lx;
  VR1 = Sum(|VR1|);
  VR3 = VR3 >> 1;
  VR4 = VR4 >> 1;
  VR15 += VR1;
  VR3 = VR3 - VR4;
  VR3 = Sum(|VR3|);
  VR15 += VR3;
}
s = VR15;
s = s<<1;
```

(b) mmx64

```
ACC1 = 0;
ACC2 = 0;
VL   = h;
R2   = lx;
VR1  = MEM[p1]    (Vs=R2);
VR2  = MEM[p2]    (Vs=R2);
ACC1 = Sum(|VR1 - VR2|);
VR3  = MEM[p1+8]  (Vs=R2);
VR4  = MEM[p2+8]  (Vs=R2);
ACC2 = Sum(|VR3 - VR4|);
R5   = Sum(ACC1);
R6   = Sum(ACC2);
R5   = R5 + R6;
s    = R5;
```

(c) vmmx64

```
for(j=0; j<h; j++){
  VR1 = MEM[p1];   p1 += lx;
  VR2 = MEM[p2];   p2 += lx;
  VR1 = VR1 >> 1;
  VR2 = VR2 >> 1;
  VR1 = VR1 - VR2;
  VR1 = Sum(|(VR1|);
  VR15 += VR1;
}
s = VR15;
s = s<<1;
```

(d) mmx128

```
ACC1 = 0;
Vl   = h;
VR1  = MEM[p1]  (Vs=lx);
VR2  = MEM[p2]  (Vs=lx);
ACC1 = Sum(|VR1 - VR2|);
R5   = Sum(ACC1);
s    = R5;
```

(e) vmmx128

Figure 4.2: Motion estimation code example

4.2.2 Simulation Framework

Emulation libraries containing the multimedia instructions have been used for the evaluated extensions: MMX64, MMX128, VMMX64 and VMMX128. Most of the function-

Application	Description	Kernel	Description	Data size
mpeg2enc	MPEG2 video encoder	*motion1*	Sum of Absolute Differences	16 × 16 8-bit
		motion2	Sum of Quadratic Differences	16 × 16 8-bit
		idct	Inverse Discrete Cosine Transform	8 × 8 16-bit
		fdct	Forward Discrete Cosine Transform	8 × 8 16-bit
mpeg2dec	MPEG2 video decoder	*comp*	Motion compensation	8 × 4 8-bit
		addblock	Picture decoding	8 × 8 8-bit
		idct	Inverse Discrete Cosine Transform	8 × 8 16-bit
		fdct	Forward Discrete Cosine Transform	8 × 8 16-bit

Table 4.2: Benchmark set description

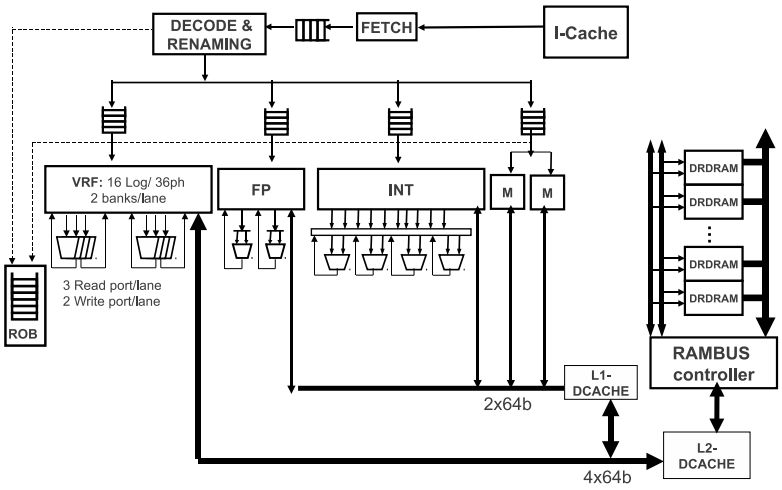


Figure 4.3: General diagram of the simulated processor microarchitecture

ality of MMX and SSE ISAs have been implemented into the MMX64 and MMX128 emulation libraries respectively, although it is important to note that the modeled extensions use more logical registers and they are based on the Alpha ISA, not on the IA32. Kernels were developed using these emulation libraries. To maximize performance, optimization techniques like loop-unrolling and software pipelining were applied. All codes have been compiled using GCC 2.95.2 with the -O2 flag.

The simulation tool used in this work was an improved version of Jinks Simulator [74], that is a parametrizable simulator targeted at evaluating out-of-order superscalar architectures with vector extensions. A combination of trace-driven and execution-driven approaches based on ATOM [230] were used for generating input traces for the simulator.

4.2.3 Processor Models

The baseline processor is a 2-way out-of-order superscalar core similar to MIPS R10000 [266] with the addition of a MMX64 SIMD extension. A general diagram of the processor microarchitecture is shown in Figure 4.3

We have evaluated four different configurations that include MMX and VMMX approaches for 64 and 128-bit registers:

- 2/4/8-way superscalar processor + MMX64

Parameter	MMX 2/4/8 way	VMMX 2/4/8 way
Physical SIMD registers	40/64/96	20/36/64
Fetch, Decode, Grad.	2/4/8	2/4/8
Integer FUs	2/4/8	2/4/8
FP FUs	1/2/4	1/2/4
SIMD issue	2/4/8	1/2/3
SIMD FUs	2/4/8	1×4/2×4/3×4
Mem FUs (L1 ports)	1/2/4 (x64b)	1/1/2 (x64b)
L2 ports	-	1x(64b/128b/256b)

Table 4.3: Modeled processors

- 2/4/8-way superscalar processor + MMX128
- 2/4/8-way superscalar processor + VMMX64
- 2/4/8-way superscalar processor + VMMX128

Table 4.3 shows the processor configurations used for the simulations. The 8-way superscalar processors are too aggressive configuration that are obviously not suitable for embedded systems and are nowadays unfeasible in a high performance general purpose processor at current clock frequencies, but they can be used as a guide of the potential performance that could be obtained (and complexity problems that could be found) when scaling processor resources.

4.2.4 Memory Hierarchy Model

A detailed memory hierarchy model with two levels of on-chip cache and a Direct RAMBUS main memory system have been included in the simulator. Table 4.4 shows the configuration parameters for caches and main memory. Parameters are similar to those found in some recent microprocessors with multimedia extensions like PowerPC970. For VMMX versions a *vector cache* was used [190]. The *vector cache* is a two-bank interleaved cache targeted at accessing stride-one vector requests by loading two whole cache lines (one per bank) instead of individually loading the vector elements. Then, an interchange switch, a shifter, and a mask logic correctly align the data. Scalar accesses are made to the L1 conventional data cache, while vector accesses bypass the L1 to access directly the L2 vector cache. This bypass is somewhat similar to the bypass implemented in Itanium2 processor for the floating point register file [154]. If the L2 port is B×64-bit wide, these accesses are performed at a maximum rate of B elements per cycle when the stride is one, and at 1 element per cycle for any other stride. A coherency protocol based on an exclusive-bit policy plus inclusion is used to guarantee coherency.

As shown in Table 4.4 the latency value for the 2 cache levels and the main memory are relative high, this is done because we want to determine the ability of the proposed extensions to tolerate high latencies in the memory subsystem.

Parameter	L1	L2
size	32KB	512KB
number of ports	1/2/4	1
port width (bytes)	8	16/32/64
number of banks	8	2
sets per bank	32	2048
associativity	4	2
line size (bytes)	32	128
latency	3	12
Main Memory Latency (cycles)	500	

Table 4.4: Memory hierarchy configuration

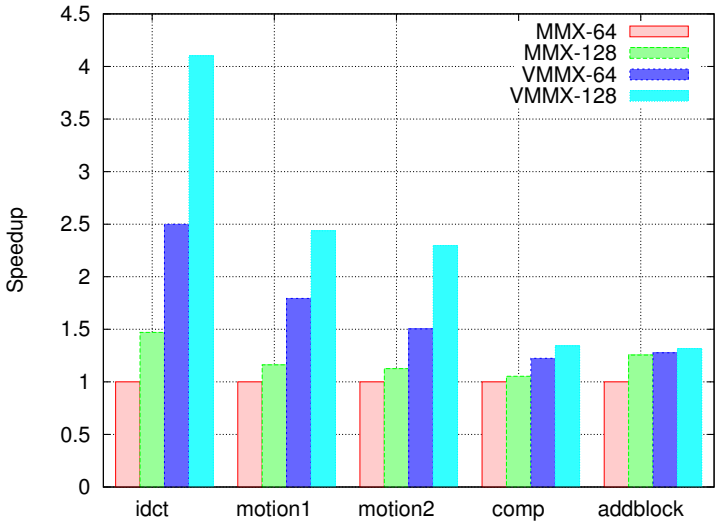


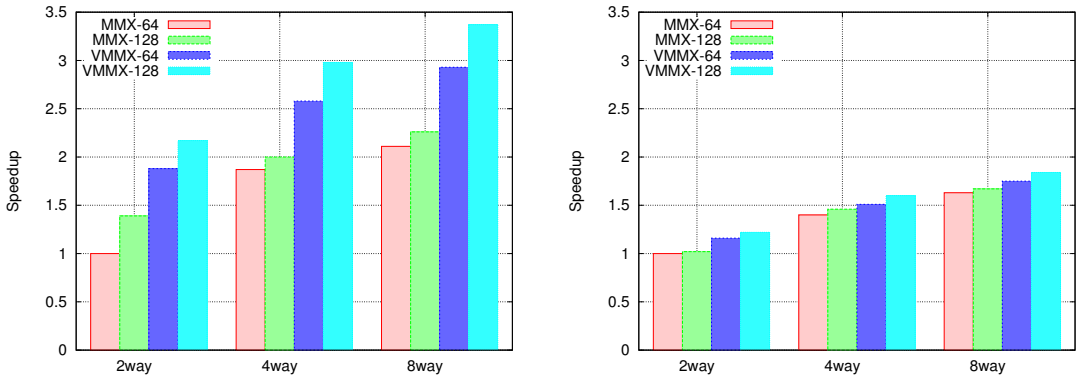
Figure 4.4: Kernels speedup (2-way)

4.3 Simulation results

4.3.1 Kernels Speedup

Figure 4.4 shows the kernels speedup for the different multimedia ISAs under study. The baseline is the 2-way superscalar processor with a MMX64 extension. Scaling from MMX64 to MMX128 does not result in great performance increment taking into account that register and functional units are twice the size of the MMX64 ones. The speedup goes up to 1.47X for *idct* and 1.25X for *addblock*. These kernels have a regular data pattern and they adapt well to 128-bit wide registers.

VMMX versions of kernels exhibit bigger speedups than the MMX ones in all the cases and produce significant speedups when going from VMMX64 to VMMX128 versions, except for *addblock* kernel. The bigger speedups (4.10X for *idct*, 2.43X for *motion2* and 2.29X for *motion1*) are due to the better matching between the data organization and the matrix registers structure. The small speedup obtained by *comp* and *addblock* in all versions is related with the parallel data available (8x4 pixels in *comp* with a stride of 800), that represents a small fraction of the matrix registers in VMMX64 and incurs in some arithmetic overhead in VMMX128.



(a) Mpeg2enc (b) Mpeg2dec

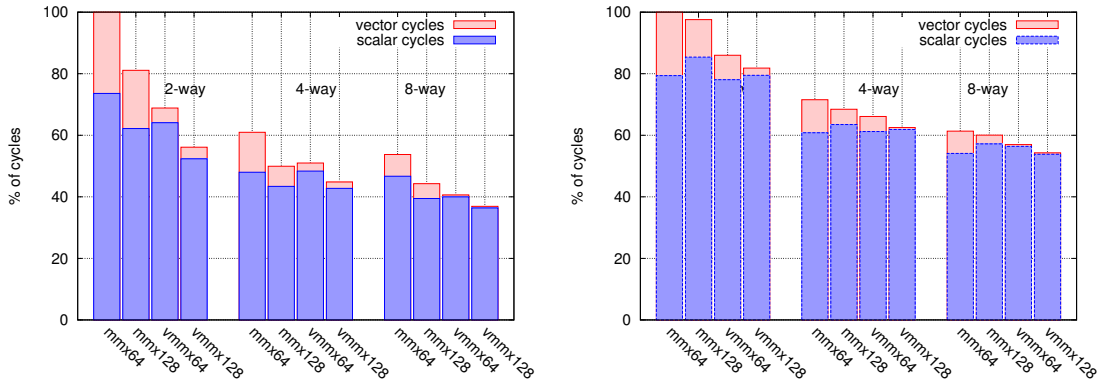
Figure 4.5: Full applications speedup

In the *idct* VMMX128 version, it is possible to pack the 8x8 16-bit input data set and coefficients in matrix registers and then performing a multiply-accumulate operation between them. *Idct* exhibits the biggest speedup due to the use of vector registers as a cache. In VMMX versions we use a 2D-matrix algorithm that need to multiply the input matrix and its transpose with the coefficients matrix. In VMMX128, due to the fact that we can store the whole matrices in vector registers, we can maintain the matrix coefficients in a vector register during the two matrix products and perform all the operations inside a vector register and only go to memory to store the final result. This saves a lot of redundant load operations and allows to apply software pipelining over the packed accumulator that would be extremely difficult to implement in a scalar or MMX versions.

4.3.2 Complete Applications Speedup

Mpeg2enc is the application which takes more benefit from the use of matrix registers. Figure 4.5a shows that VMMX versions of the application scales better than MMX ones. VMMX128 version has the biggest speedup due to the good matching of data in the *motion* and *idct* kernels to the 128-bit matrix registers. These kernels account for more than 25% of the total execution time of the MMX-64 version of the application running on the 2-way processor. *Mpeg2dec*, instead, shows a significant speedup but the difference between MMX and VMMX versions is smaller than in *mpeg2enc*. In this application motion compensation routines are not so much significant of the total execution time and their data parallelism is not so big. Furthermore *mpeg2dec* presents a lot of scalar code in picture decoding that can not be vectorized.

As shown in figure 4.5a, for *mpeg2enc*, it is possible to obtain a similar performance with a 2-way VMMX128 processor instead of a 8-way MMX128 one. In this case, scaling the 2-dimensional register file of a simpler processor is much more effective than scaling the complete resources of a processor with 1-dimensional registers.



(a) mpeg2enc (b) mpeg2dec

Figure 4.6: Cycle count distribution

4.3.3 Cycle Breakdown

Figure 4.6 shows the dynamic cycle distribution for the two applications. The lower part of each column represents the dynamic cycles used in vector operations, while the upper part comes from the scalar ones. Results are normalized by the dynamic cycle count of the reference 2-way MMX64 superscalar processor.

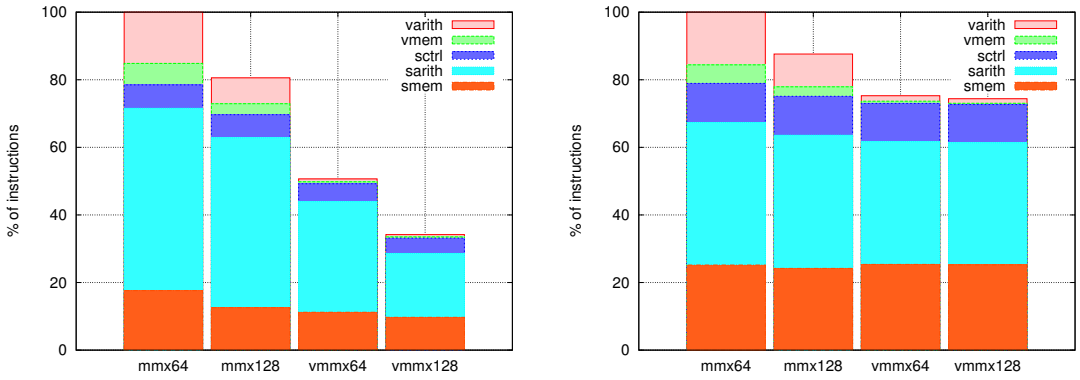
As it was expected, scaling the MMX64 extension to MMX128 provides a significant drop in the number of cycles to execute the vector code section. For the 2-way architecture MMX128 achieves a 40% and 69% reduction in vector cycles over MMX64 for the *Mpeg2enc* and *Mpeg2dec* applications respectively.

Scaling the 2D vector extension in both dimensions (width and length) achieves the maximum reduction: for the 2-way architecture, the VMMX128 extension reduces the execution time of the vector code over the MMX64 extension by 7 and 8.75 times for the *Mpeg2enc* and *Mpeg2dec* applications respectively. It is important to note that VMMX128 not only reduces the execution time of the vector section but also the scalar one due to overhead elimination.

However, it can be observed that, when most of the available DLP parallelism is exploited via multimedia extensions, the remaining scalar part of the code becomes the main bottleneck. For the 8-way VMMX128 architecture, the vector cycles represent only the 1.43% and 0.82% of the overall execution time for the *Mpeg2enc* and *Mpeg2dec* applications respectively. By the Amhdal Law, further improvements in the execution of the vector region would be imperceptible in the full application.

4.3.4 Dynamic Instruction Count

Figure 4.7 shows the dynamic instruction count for the applications under study. Again, results are normalized by the dynamic instruction count of the MMX64 architecture. The operations have been classified into five categories: scalar memory, scalar arithmetic, control, vector memory and vector arithmetic. We observe that the VMMX architectures execute about 30% fewer instructions than the MMX64, and the MMX128 an average of 15% fewer instructions. This is obviously due to the capability of these extensions to pack more operations into a single instruction.



(a) mpeg2enc (b) mpeg2dec

Figure 4.7: Dynamic instruction count

As seen in figure 4.7a, the biggest instruction reduction is achieved by the *mpeg2enc* application. This reduction comes from the commented elimination of scalar instructions used for address computation and loop manipulation. In any way, note that the limit of packing data seems to be reached, and scaling further over, either in width or in length, would not provide any noticeable benefit.

4.4 Analysis of New SIMD Extensions

As it was mentioned in the introduction this study was performed when most of the SIMD extensions had 64-bit or 128-bit wide registers. Currently most SIMD ISAs use 128-bit registers and recent processors have included wider registers like Intel AVX extension which uses 256-bit registers [63] and the Larrabee processor that uses 512-bit registers [211]. But matrix registers has not been included in any commercial processor.

It is worth to mention that the newer extensions have been developed for scientific floating point code. AVX, for example, does not include integer 256-bit μSIMD instructions. Even if such wider SIMD extensions include integer support their usability for video coding applications is still limited. First, because newer video codecs like H.264 include features like variable block size, that results in low efficiency when processing smaller block sizes (e.g. 4×4 8-bit) in wide registers. The second limitation is the memory architecture: even with larger blocks (e.g. 16×16 8-bit) that can use a whole 256-bit register, loading and storing from memory requires strided accesses with a stride equal to frame width.

We consider that the scalability analysis presented in this article is still valid even for the new and wider SIMD extensions. A 2-D SIMD extension can exploit the same (or more) DLP while at the same time it can reduce the complexity of key units in the microarchitecture. A fully matrix architecture has several advantages to conventional SIMD extensions for video coding applications: flexibility by adapting the vector length to the data structures, support for gather and scatter for non-unit strided accesses and the ability to scale the hardware resources without compromising code compatibility.

4.5 Summary

In this chapter we have presented an scalability analysis of SIMD extensions for video coding and decoding applications. Scaling current 1-dimensional SIMD extensions was compared to scaling a 2-dimensional architecture. The comparison was made using both kernels and complete applications. Scaling was made in the width of SIMD registers and in processor resources. The matrix architecture with 128-bit registers has shown the best performance improvements compared to a 64-bit matrix architecture and to 1-dimensional (64-bit and 128-bit) SIMD extensions.

It was demonstrated that, for the analyzed video applications, a simple processor with a VMMX128 extension can delivery more performance than a processor with a MMX extension and more resources. This feature and the reduced complexity in some critical structures of the matrix pipeline, like the register file, makes the matrix enhanced processor a suitable choice for embedded applications.

By using the scaled Matrix architecture the analyzed video applications are reaching the limits of available DLP in the inner loops. Further scaling on the width or length of matrix registers can no deliver significant performance improvements because the execution time is now dominated by the scalar portion of the code. Extracting more parallelism requires to go beyond the inner loops and exploit a coarse-grain DLP.

5 Workload Characterization of H.264 Decoding

The chapter presents an analysis of the performance of H.264/AVC decoding and a comparison with other video codecs. The main objective is to measure the computing demands of H.264/AVC decoding at high definition and to analyze the potential for optimization and parallelization.

After the publication of the standard the only publicly available software implementation of the decoder was the reference code (called "Joint Model" or JM). We performed the first experiments using that code. Although these experiments allowed us to estimate the computational complexity of HD H.264/AVC decoding, the performance of the reference code resulted to be extremely low. This is not the result of the complexity of H.264/AVC decoding algorithm but the consequence of an unoptimized code. Our own evidence shown that this code is not suitable for complexity and performance analysis. Because of that, we switched to a different implementation of the H.264/AVC decoder that comes from the FFmpeg project. This alternative code is, at least, one order of magnitude faster than the reference implementation. Due to that, all the following experiments were based on the FFmpeg code. Even that, the reference code was (and still it is) used for many performance studies and it was included in the SPEC-2006 CPU benchmark [93].

The workload characterization is performed by running the benchmarks on a real machine and collecting run time information using hardware performance counters. The performance analysis of H.264/AVC decoding is complemented with results of MPEG-2 and MPEG-4 video decoding.

5.1 Related Work

One of the most relevant in workload analysis was the design and characterization of the *Mediabench* benchmark which includes a MPEG-2 encoder and decoder with low resolution input sequences [136]. At that time, the MPEG-4 and H.264/AVC video codecs were not available. The *Berkeley Multimedia Workload* benchmark is based on *Mediabench* and use the same MPEG-2 video codecs but with input sequences at SD, HD and FHD resolutions [229]. This benchmark does not include the most recent video codecs either. This issue was partially solved by an extension to *Mediabench* in which the MPEG-4 and H.263 codecs were included [84]. But they do not address the issue of the increase in frame resolution either. Although these studies perform an analysis of the potential for optimization they do not include an analysis of the performance effect of using SIMD instructions.

On the other hand, in a study about the available parallelism in video applications the authors analyze the performance of the MPEG-1, MPEG-2, H.263 and MPEG-4 video codecs by simulating an idealistic machine with infinite resources [148]. They show that

with such a machine it is possible to obtain speed-ups from 30X to 1000X compared to a reference implementation. They do not address the problem of the increase in image resolution, and do not use SIMD instructions. In a similar study, the authors analyzed several multimedia applications including MPEG-2 and H.263 video codecs and conclude that the variability observed in the execution of these applications comes from application properties, like I-P-B type of frames, not from the unpredictability introduced by cache memories and other architecture features of superscalar processors [99]. In a comparison of different multimedia instruction sets, the authors use the MPEG-2 codec and analyze in detail the motion compensation kernels with different image resolutions, the focus of this analysis is the comparison of media ISAs [226]. In another study, the authors present an evaluation of some multimedia applications including the MPEG-2 codec but using low resolution sequences [193]. They analyze the impact in performance by using the VIS extension of the SPARC architecture.

There are some works that deal with the performance of the H.264 codec. Some of them perform a complexity analysis of H.264 with special attention on the video quality for low bitrate applications [96, 133]. They study the complexity of the H.264 decoder and conclude that H.264 is approximately 2.5 times more complex than H.263. In an another study, the authors have developed a SIMD optimized version of the H.264 decoder using Intel SSE instructions and analyze the performance of the decoder for CIF and SD resolutions [272]. In a different micro-architectural study of the H.264 reference decoder, the authors use low and standard video resolutions in order to analyze the availability of ILP by simulating a machine with infinite resources [219]. They suggest that the main bottleneck of the application is unpredictable branch behavior. In an another performance characterization of MPEG-2 and H.264 decoders, the authors develop a performance analysis of these codecs on the Pentium architecture in which they compare the differences between the kernels of H.264 and MPEG-2 for video at SD resolution[94]. They conclude also that branch misprediction is a limiting factor for H.264 decoding due to the data dependent branches of some kernels.

The main difference of our study with respect to all the previously mentioned works is the performance characterization of H.264 at high resolutions. Most of the published results are focused on low bitrate applications like mobile video or video conferencing but the performance requirements of HD H.264/AVC decoding has been not analyzed previously. Additionally, our study performs a comparison of H.264/AVC with other video codecs like MPEG-2 and MPEG-4.

5.2 Methodology

The workload characterization is performed by executing a video decoder on a real machine using several input videos coded with different video codecs. Below, we describe the details of the platform, the video codecs and input videos.

5.2.1 Processor and Tools

The experiments have been done on a PowerPC970 [102, 200] machine. PPC-970 is a 64-bit PowerPC processor which is a single core derivative of the dual core Power4 processor[243]. PPC-970 has the addition of Altivec μSIMD instructions [71] and a dedicated 512KB L2 cache. It includes 10 functional units: 2 load/store units, 2 fixed

Processor	IBM PowerPC 970
Clock frequency	1.6 GHz
Level 1 I-cache	64 KB
Level 1 D-cache	32 KB
Level 2 cache	512 KB
Main Memory	512 MB
System Bus	800 MHz
Operating System	Mac OS-X
Compiler	GCC 3.3.3
Compiler Optimizations	-O3, mtune=G5

Table 5.1: Experimentation platform

point units, 2 floating point units, 2 Altivec SIMD units, one branch unit and one control register unit. One of the Altivec units performs simple fixed, complex fixed and floating point operations; and other one performs permute operations. The processor can fetch and decode up to 8 instructions per cycle, dispatch up to 5 instructions per cycle (in a single instruction group), issue up to 1 instruction per cycle to each of the functional units and retire up to 5 instructions per cycle. In total, it can maintain up to 200 instructions in flight. At 2.0 GHz the processor consumes 50W.

Performance data has been obtained using the performance monitoring counters available in the processor and collected with the Apple Computer Hardware Understanding Developer (CHUD) tools [108]. Hardware monitor counters have been used to collect profiling information, completed instructions, CPU cycles, cache accesses and misses, and branch prediction information. Additionally, a time interval sampling analysis was conducted for analyzing the phase behavior of the program execution [146].

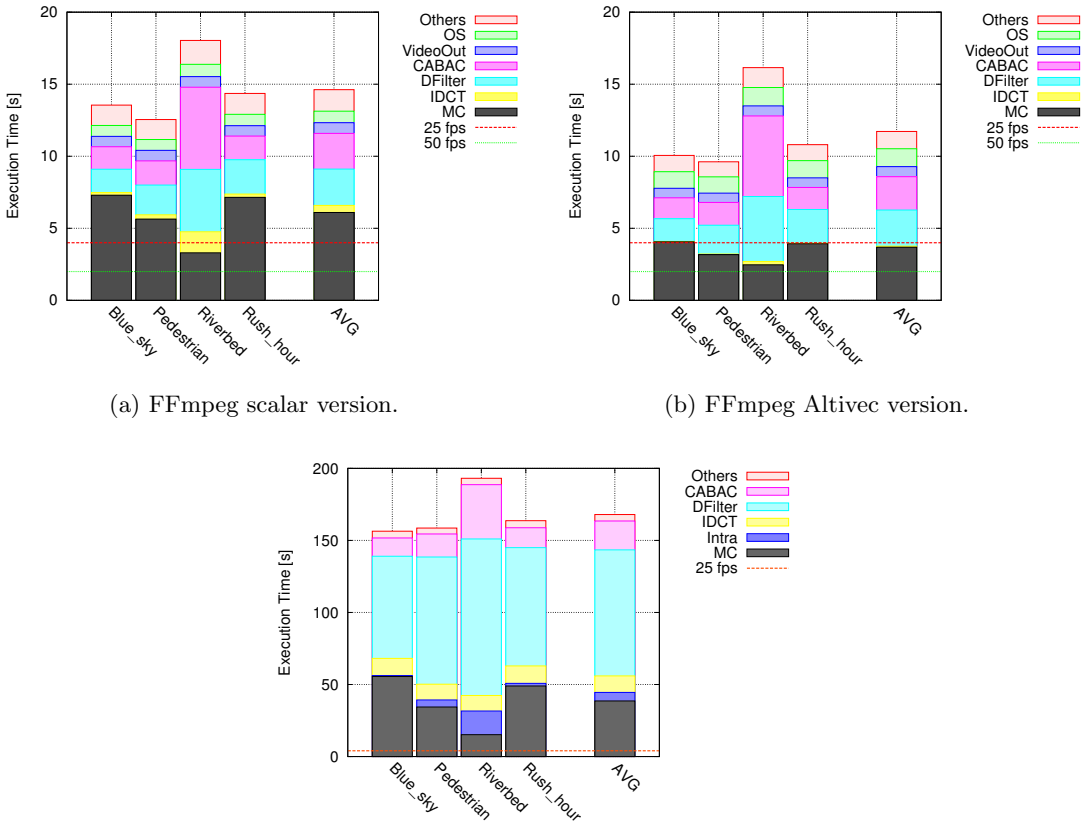
5.2.2 Codec Configuration

Two H.264/AVC decoders are used: the standard reference code (H.264-REF) version 9.5 (JM-9.5 [114]) and the FFmpeg highly optimized (H.264-FF) decoder [76]. These two H.264/AVC decoders are compared with the *XviD* MPEG-4 decoder [262] and the *libmpeg2* MPEG-2 decoder [149]. H.264/AVC videos were coded with the JM-9.5 reference encoder using the H.264/AVC main profile, applying a constant quantization parameter and using a I-P-B-B-P-B-B sequence of pictures. The configuration parameters of all codecs were equally balanced in order to maintain a similar quality in the resulting videos.

In order to improve the performance of the H.264 decoders some kernels were implemented using Altivec SIMD instructions, both in the reference code and in the FFmpeg code. Optimized kernels include: luma and chroma interpolation in motion compensation and the inverse cosine transform.

5.2.3 Test Sequences

We use a set of four input videos called: *blue_sky*, *rush_hour*, *pedestrian_area* and *riverbed* from the "MPEG-Test Sequences" [244]. They have different motion characteristics and spatial details. All of them have 100 frames with progressive scanning at 25 frames per second and use a 4:2:0 chroma sub-sampling format. We coded all the input videos at three different resolutions: SD (720×576), HD (1280×7120) and FHD (1920×1088).



(a) FFmpeg scalar version.

(b) FFmpeg Altivec version.

(c) Reference code Altivec version.

Figure 5.1: Profiling of H.264 decoder

5.3 Analysis

By running the workloads on a real machine we have collected different kind of measures. First, we perform a profiling for determining the most executed parts of the code of each codec, next we obtain different performance metrics per frame, and finally we have perform a sampling analysis that allows us to detect program execution phases.

5.3.1 Profiling of the H.264/AVC Decoders

For profiling purposes we divide the execution of the H.264/AVC decoder in 8 stages: motion compensation (MC) that includes the luma and chroma interpolation kernels, Intra Prediction (Intra), Inverse Discrete Cosine Transform (IDCT), Deblocking Filter (DFilter), Entropy decoding (CABAC), bitstream parsing and other (Others), video output (VO) and Operating System (OS).

Figure 5.1 shows the profiling of the scalar (Figure 5.1a) and SIMD (Figure 5.1b) versions of the FFmpeg and the SIMD (Figure 5.1c) version of the reference H.264/AVC decoders. For space reasons we only show the results for DHS resolution.

An important result is that the JM-9.5 reference decoder (with Altivec optimizations) is 17.4X times slower than FFmpeg one. This result confirms the fact that JM reference

software is not well suited for complexity and architecture studies.

With the Altivec optimizations in the FFmpeg code, the total execution time has been reduced in 1.27X in average. The speedup for Luma interpolation and IDCT kernels are 3.27X and 6.9 respectively. After Altivec optimization the execution time is dominated by the MC (30.5%), DFilter (20.8%), and the entropy decoding (19%). It is important to note that there is more room for SIMD optimization, specially in the interpolation of chroma signals and in the deblocking filter [272]. We will consider these optimizations later.

When comparing the profiling for different videos, the time distribution for *blue_sky*, *rush_hour* and *pedestrian_area* remains approximately equal. *riverbed* sequence exhibits a significant low performance compared to the other ones. In this video there is a random motion of fluids that generates a lot of macroblocks with intra-prediction, and a lot of blocks with transform coefficients. This results in a bigger execution time of the entropy decoding stage which can not be optimized with SIMD instructions.

In Figure 5.1 we have included the lines for 25 and 50 frames per second as a reference of the performance required for real-time operation. It can be noted that it is not possible to achieve real-time even with the FFmpeg code with Altivec optimizations. Speedups of 2.9 and 5.8 are required to reach an average of 25 and 50 fps at FHD respectively. Even reducing to zero the MC, IDCT and DFilter stages (e.g. exploiting DLP), the optimized H.264/AVC decoder will not reach real-time operation. That means that multiple levels of parallelization are required to provide the required performance for real-time.

5.3.2 Instructions and Cycles

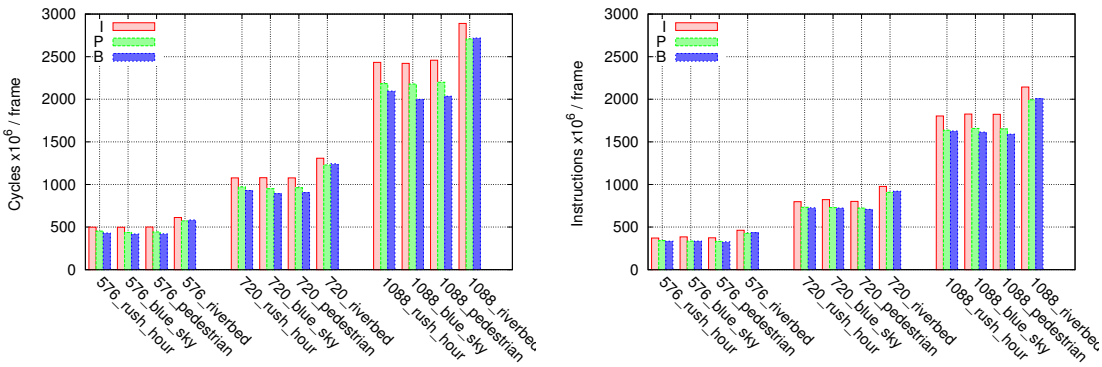
Table 5.2 shows the average CPU cycles and instructions per frame for the different input sequences and different codecs under study. The H.264-REF decoder executes 10.5X and 66X more instructions in average than MPEG-4 and MPEG-2 respectively, and the H.264-FF decoder executes 1.36X and 8.61X more instructions respectively.

Also can be noted that the IPC changes with different input sequences but not with frame resolution. Taken that $IPC = (InstCount)/(Freq \times ExecTime)$ and with the information of instructions per frame and assuming a 3.0 GHz clock frequency it is possible to estimate the necessary IPC for decoding H.264/AVC in real time. The results are 2.99, 6.43 and 14.33 for the H264-REF decoder at 720x576, 1280x720 and 1920x1088 resolutions respectively. For the H.264-FF decoder it is possible to reach the real time performance at 720x576, 1280x720 resolutions at 3.0 GHz, but for the 1920x1088 resolution it is necessary an IPC of 1.83. Although this value could be achieved with more SIMD and scalar optimizations it becomes clear that there is not enough scalability for the growing quality demands of video applications.

The impact of the different types of frames in performance is shown in Figure 5.2 with the distribution of cycles and instructions in I, P and B frames for the H.264-REF code. The increase in cycles and instructions is proportional to the increase on the image area. The HD and FHD frame resolutions have an area that is 2.2 and 5 times bigger that the SD resolution respectively. Also it can be noted that intra-prediction takes more cycles and instructions than inter-prediction (P and B). Consequently for the sequences with a lot of I-macroblocks, like riverbed, the decoding time is bigger.

Sequence	H.264-REF			H.264-FF			MPEG-4			MPEG-2		
	Cyc. $\times 10^6$	Inst. $\times 10^6$	IPC	Cyc. $\times 10^6$	Inst. $\times 10^6$	IPC	Cyc. $\times 10^6$	Inst. $\times 10^6$	IPC	Cyc. $\times 10^6$	Inst. $\times 10^6$	IPC
720×576												
rush_hour	438	337	0.77	42	43	1.03	32	34	1.05	5.6	4.0	0.71
blue_sky	426	336	0.79	40	42	1.05	34	37	1.07	6.6	4.9	0.74
pedestrian	429	330	0.77	40	40	1.02	28	27	0.97	5.8	4.2	0.73
riverbed	580	432	0.74	72	68	0.94	44	41	0.93	11.0	9.9	0.90
Average	468	359	0.77	48	48	1.01	34	34	1.00	7.3	5.8	0.77
1280×720												
rush_hour	945	729	0.77	88	90	1.03	70	74	1.05	11.0	8.0	0.73
blue_sky	913	726	0.80	82	86	1.05	71	77	1.07	11.9	9.0	0.76
pedestrian	925	714	0.77	83	86	1.04	60	59	0.98	11.5	8.6	0.75
riverbed	1239	917	0.74	144	139	0.96	91	85	0.93	21.6	19.8	0.92
Average	1005	771	0.77	99	100	1.02	73	73	1.01	14.0	11.4	0.79
1920×1088												
rush_hour	2128	1632	0.77	192	198	1.03	160	165	1.03	26.5	19.2	0.72
blue_sky	2060	1631	0.79	181	191	1.05	160	168	1.05	25.9	19.1	0.74
pedestrian	2094	1613	0.77	185	196	1.06	146	144	0.99	27.5	20.7	0.75
riverbed	2717	2004	0.74	295	295	1.00	194	182	0.94	47.4	41.1	0.87
Average	2250	1720	0.77	213	220	1.04	165	165	1.00	31.8	25.0	0.77

Table 5.2: Cycles, instructions and IPC per frame



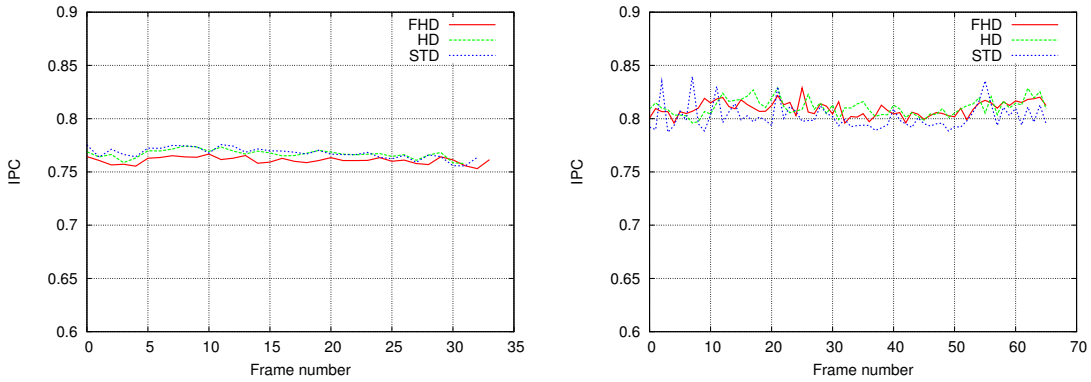
(a) Cycles per frame

(b) Instructions per frame

Figure 5.2: Performance impact of the different types of frames

IPC Variability

A previous study [99] has demonstrated that most of the variability in the performance of frame based multimedia applications comes from the different kind of frames that exist in the application (ie I,P,B). Figure 5.3 shows the IPC per frame classified by P- and B-frames for the blue_sky sequence. Instructions and cycles exhibit a small variation between frames of the same type, and because of that the IPC remains almost constant. Based on that, we can conclude that the amount of computational work necessary to decode each frame depends mostly on the frame type (P and B) and on the density of I-macroblocks in each frame. There was no significant variation in average IPC per frame for the three resolutions under study.



(a) P-frames (b) B-frames

Figure 5.3: IPC in P- and B-frames for 1008_blue_sky

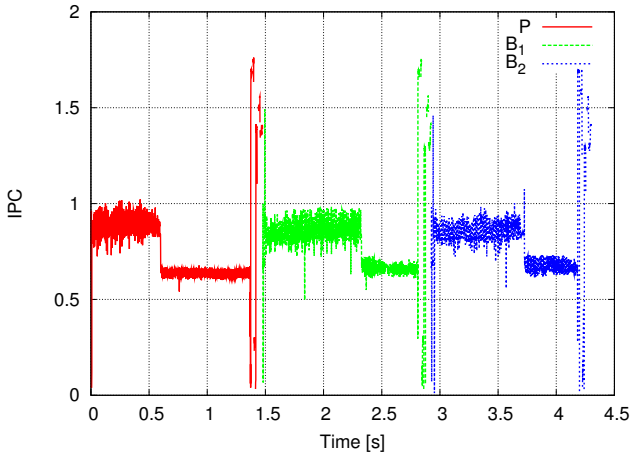


Figure 5.4: Sampling of IPC for 1088_blue_sky

IPC Sampling

In addition, we have made an analysis of the decoding of the whole sequence and we found that the application exhibits a phase behavior at the granularity of a P-B-B sequence. Figure 5.4 shows the time behavior for the decoding of a P-B-B group of frames for the 1088_blue_sky sequence. The three frames are separated by big peaks and fluctuations of IPC which corresponds to the copy of the decoded frame out of the frame buffer. In turn, the processing of each frame has three clearly differentiated stages: the first, which includes the entropy decoding, inverse transform and motion compensation, has an IPC close to one; the second, which consist of the deblocking filter, has a lower IPC near 0.6; and the third phase the IPC exhibits big fluctuations due to the "memcopy" kind of operations. The behavior is similar for both P-and-B frames (I frames are not analyzed) and in B frames the motion compensation stage is longer. For space reasons we only show the results for FHD resolution but for the other resolutions the figure is very similar with the only difference of time scale.

Sequence	H.264-REF		H.264-FF		MPEG-4		MPEG-2	
	Acc. $\times 10^6$	Miss rate	Acc. $\times 10^6$	Miss rate	Acc. $\times 10^6$	Miss rate	Acc. $\times 10^6$	Miss rate
720×576								
rush_hour	204	2.4	18	2.9	15	3.6	2.2	6.8
blue_sky	192	2.7	18	2.9	16	2.9	2.3	6.8
pedestrian	205	2.4	17	2.6	13	4.2	2.3	6.9
riverbed	299	1.7	35	1	15	5.5	3.6	6.1
Average	225	2.3	22	2.3	15	4.1	2.6	6.6
1280×720								
rush_hour	442	2.4	37	2.9	34	4.3	5.2	5.8
blue_sky	415	2.6	36	3.3	35	4	5.4	6
pedestrian	446	2.3	35	2.6	29	5.1	5.5	6
riverbed	646	1.6	69	1.8	31	5.7	7.8	6.1
Average	487	2.2	44	2.6	32	4.8	6	6
1920×1088								
rush_hour	1002	2.4	79	2.8	77	6.4	12.6	6.2
blue_sky	948	2.6	79	4.1	78	6.5	12.4	6.4
pedestrian	1015	2.3	78	2.7	68	6.7	13.1	6.2
riverbed	1431	1.6	139	1.7	67	6.1	17.6	6.3
Average	1099	2.2	93	2.8	72	6.4	13.9	6.3

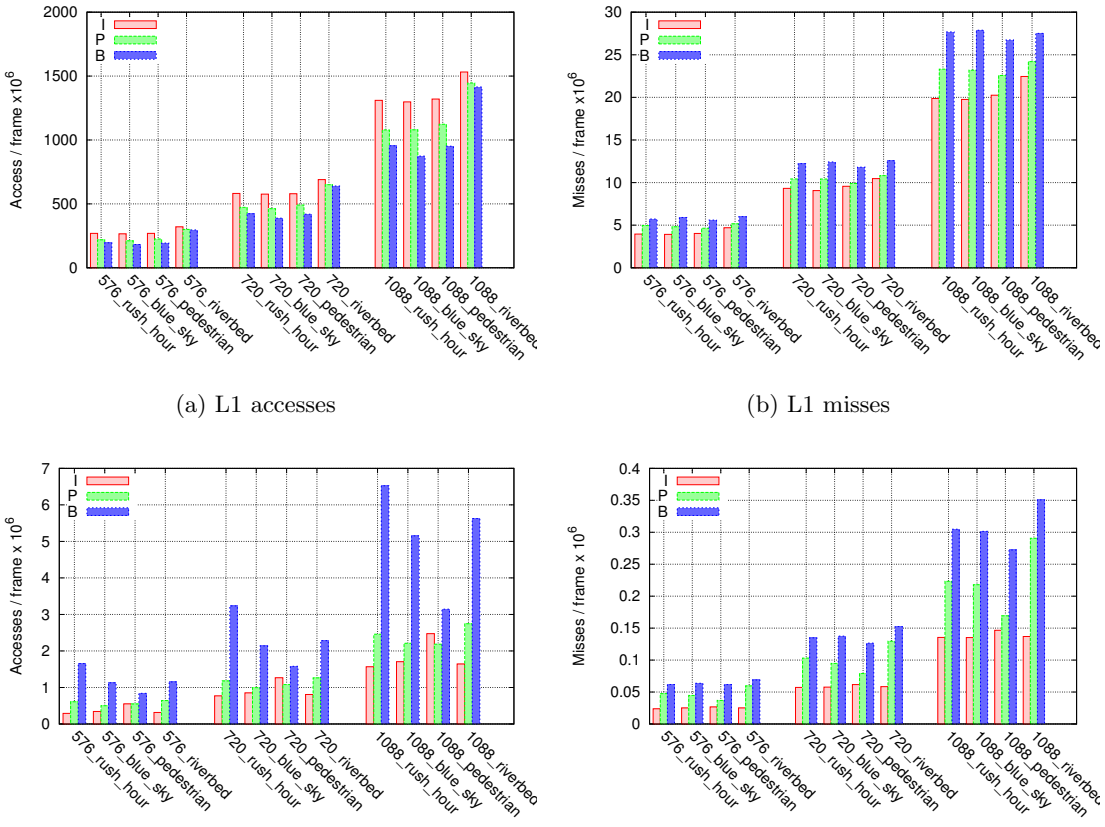
Table 5.3: d-L1 accesses and miss rate comparison

5.3.3 Cache Analysis

In order to analyze the cache behavior of the codecs under study we have collected performance events for the L1 and L2 data cache. Table 5.3 shows the average number of accesses and misses per frame for the L1 data cache. H.264/AVC decoder has many more L1 data cache accesses and misses than the other two codecs; for example, for FHD resolution, the H.264-REF decoder performs 15.2X and 79.06X more memory accesses than MPEG-4 and MPEG-2 respectively and the H.264-FF decoder performs 1.29X and 6.6X more memory accesses respectively.

Although H.264/AVC performs more memory accesses per frame it has a smaller miss rate that the other two video codecs. This is due to the fact that H.264/AVC perform more operations per frame than the other codecs but those operations are performed at the macroblock level. Macroblocks fits well into the data cache, even where the whole frame will not. The miss rate changes more between different input videos than with frame resolution, only in MPEG-4 there is an increment in miss rate with resolution. These results are in consonance with some previous studies on memory behavior for multimedia applications [228], [261] that claim that the use of cache memories benefits the performance of video coding applications.

Figure 5.5 shows the distribution of L1 and L2 data cache accesses and misses for the different type of frames in the H.264/AVC sequences. In I-frames the decoder performs more accesses to the L1 data cache than the other type of frames, mainly because intra-prediction uses several spatial prediction modes with different memory access patterns. On the contrary B frames have more data cache misses, and that is because these kind of frames have to access multiple reference frames, that are stored in a picture buffer, which does not fit in the L1 level cache. For the L2 cache, B-frames exhibits more accesses than in I- or P-frames which generate more L2 cache misses that, in turn, translates into long latency main memory accesses



(a) L1 accesses

(b) L1 misses

(c) L2 accesses

(d) L2 misses

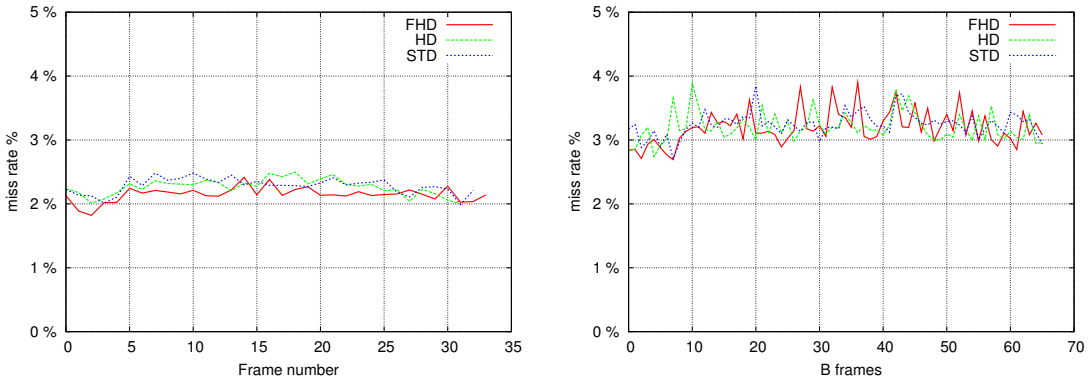
Figure 5.5: Average L1 accesses and misses per frame in H.264-REF decoder

Miss-rate Variability

Figure 5.6 shows the variation of miss rate in P- and B-frames. In the P-frames (Figure 5.6a) L1 data cache miss rate remains almost constant around 2.3%, and the miss rate is not affected by the resolution and input content. But in B-frames (Figure 5.6b), miss rate is bigger (between 3 and 4%) and exhibits variations, specially, for the FHD resolution.

Miss-rate Sampling

Figure 5.7 shows the L1 data cache time behavior for the decoding of a P-B-B sequence of the 1088_blue_sky sequence. As with the time behavior for IPC shown in Figure 5.4, there are three different phases of the execution. The first phase, in which motion compensation is performed, has a bigger miss rate than the second phase in which deblocking filtering is applied. Motion compensation exhibits a bigger miss rate that is related to the inter-prediction decoding process in which a reference frame is used to predict the current frame. The time of this phase is bigger in the B frames in which there are more than one reference frame. The peaks are related again with the third phase in which the decoded frame is sent out of the decoded picture buffer.



(a) P-frames (b) B-frames

Figure 5.6: Average dL1 accesses and misses in P- and B-frames for 1088_blue_sky

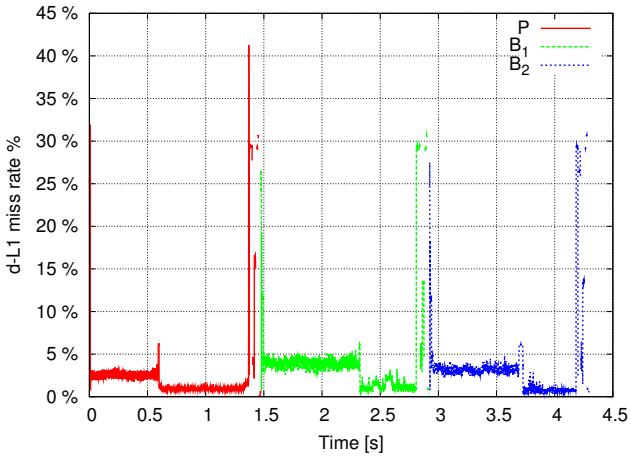


Figure 5.7: Sampling of d-L1 miss rate for 1088_blue_sky

5.3.4 Branch Prediction

H.264/AVC has a lot of different coding options that can change from macroblock to macroblock and has some kernels with a lot of data dependent branches, altogether results in a high density of branches and branch misprediction as can be seen in Table 5.4. The reference decoder executes 4X and 119X more branches compared to MPEG-4 and MPEG-2 respectively and the H.264-FF decoder executes 1.67X and 15X more respectively. In all the codecs branch prediction exhibits variations with input content but not with frame resolution. The H.264-FF decoder has a bigger branch misprediction rate than the H.264-REF decoder (2X for the FHD resolution). This comes from the fact that the reference decoder supports more (exotic) coding options than the H.264-FF decoder, and these options need to be checked in each each frame (sometimes each macroblock) but are easier to predict.

In Figure 5.8 the branch misprediction sampling is shown for a P-B-B sequence of the 1088_blue_sky sequence. The three phases scheme is again evident, and in this case

Sequence	H.264-REF		H.264-FF		MPEG-4		MPEG-2	
	Bran. $\times 10^6$	Misp. rate	Bran. $\times 10^6$	Misp. rate	Bran. $\times 10^6$	Misp. rate	Bran. $\times 10^6$	Misp. rate
720×576								
rush_hour	54	4.7	6.4	9.8	3.4	3.5	0.39	11.2
blue_sky	53	4.7	5.5	10.2	3.5	3.2	0.52	12
pedestrian	54	4.5	6.5	9.4	3.3	4.4	0.43	12.1
riverbed	79	4.6	11.3	13.1	7.2	7.6	1.14	12.4
Average	60	4.63	7.4	10.7	4.3	4.68	0.62	11.9
1280×720								
rush_hour	117	4.6	13.6	8.9	7.3	3.3	0.71	10.3
blue_sky	115	4.7	11.6	9.5	7.4	3.1	0.9	11.7
pedestrian	115	4.5	13.8	8.6	7.1	4.1	0.82	11.5
riverbed	165	4.5	23.3	12.3	14.8	7.4	2.15	12.2
Average	128	4.57	15.6	9.8	9.2	4.47	1.14	11.44
1920×1088								
rush_hour	261	4.6	30.5	8.3	16.9	3.4	1.64	10.3
blue_sky	258	4.6	26.5	8.8	16.2	3.1	1.78	11.5
pedestrian	260	4.5	31.1	8.1	17.4	4.1	1.87	11.4
riverbed	357	4.5	50	11.4	32	7.1	4.25	12
Average	284	4.52	34.5	9.1	20.6	4.44	2.38	11.29

Table 5.4: Branches (Bran.) and branch misprediction (Misp.)

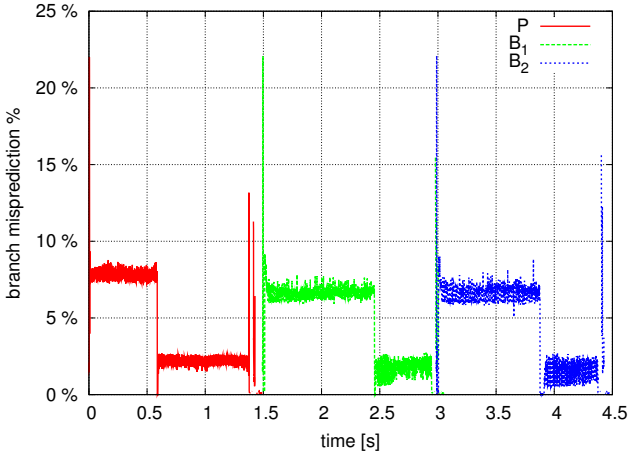


Figure 5.8: Branch misprediction sampling

the entropy decoding and motion compensation stage exhibits a bigger misprediction rate than filtering. The main reason for this behavior is that CABAC entropy decoding has a lot of data dependent branches that are difficult to predict and also motion compensation has a lot of options that can change inside each macroblock (block size, pixel interpolation) and these values are selected by the encoder according to the content of the input sequence also making them difficult to predict.

5.4 Performance on Recent High Performance Processors

The previous analysis showed that high performance uni-core processors from a previous generation (like the PowerPC970) can not decode FHD H.264 video in real-time. This

is not the case for the most recent generation of processors. For example, our own experiments with the Intel Sandy Bridge processor (Intel-core-i5 2500k, GCC-4.4.5 -O2, Linux kernel 2.6.35-25) running at 3.6 GHz shows that is possible to decode FHD H.264 videos at 66 frames per second. This does not invalidate the results presented in this chapter, because the main point that we want to highlight is not the absolute performance of one processor but the scalability issues with newer video decoding applications. If the resolution and/or frame rate is increased, for example, to QHD (3840× 2160), the Intel-core-i5 processor is only able to process 12 fps. Moreover, the performance of uni-core processors is not going to increase at the same rate than in the past years (as discussed in Chapter 1) making clear that additional performance optimizations are needed.

5.5 Summary

In this chapter we presented a performance evaluation of H.264/AVC video decoding for High Definition applications using hardware performance profiling techniques.

The profiling analysis has shown that the H.264/AVC has kernels, like pixel interpolation and deblocking filter, with bigger computational requirements than the kernels of previous video standards. Conversely some kernels that are more important in former MPEG codecs, like the IDCT, have a little impact on the performance of the H.264/AVC decoding. After SIMD optimization, execution time is distributed among three main kernels: CABAC entropy decoding, motion compensation and deblocking filter. Although there is more room for SIMD optimization it proved to be insufficient to provide all the required performance for real-time operation.

In addition, we quantified the complexity of the H.264/AVC decoding process by measuring the instructions and cycles that are necessary for decoding a frame. The comparison with MPEG-4 and MPEG-2 shows that H264/AVC requires 1.36X and 8.62X more operations than MPEG-4 and MPEG-2 respectively. Although it could be possible to provide the processing requirements with some new generation of high frequency high performance superscalar processors, this choice is not scalable in terms of performance or power consumption.

The H.264/AVC reference decoder (H.264-REF) proved to have a very low performance compared to other codecs and the optimized H.264 decoder (H.264-FF). Absolute numbers (in term of frame per second) obtained with the H.264-REF decoder can give misleading results. Our suggestion is to avoid the use of H.264-REF for any purpose different than validating the standard.

The main conclusion is that H.264/AVC decoding of HD (and beyond) video is a big challenge for high performance general purpose processors because it presents a tremendous amount of data that needs to be processed in real-time but not as regular as it has been with other video codecs. H.264/AVC has new kernels, some of them computationally intensive, some with demanding memory access patterns and some with high branch misprediction rates. H.264/AVC decoding of HD video requires a combination of multiple optimizations such as more performance for the multimedia instruction sets of one processor and the use of multiple processors.

6 HD-VideoBench: A Benchmark for HD Video Applications

In this chapter, we present `HD-VideoBench`, a benchmark devoted to HD video processing. In a previous chapter, we presented the limitations of the reference H.264/AVC decoder for performance and complexity studies. This lead us to the definition of our own benchmark for video codec applications for high definition scenarios. Although there are several multimedia benchmarks, such as Mediabench [136], Berkeley Multimedia Workload [229] or EEMBC [147], none of them fulfills all the requirements for a complete HD video benchmark. Some of them use the reference versions of the applications that were written with the purpose of validating the standards but not for high performance. Additionally, most of them focus on the MPEG-2 (or MPEG-4 at the most), but only a few of them include recent video codecs like H.264/AVC. Even in the case of including H.264/AVC, none of them addresses HD resolutions, which requires a particular and careful selection of the coding options and input sequences that, in turn, results in different computational and memory requirements. Apart, some of them include just kernel or not actual applications which can lead to unrealistic results. Finally, some existing benchmarks do not provide source code with a free license limiting the reproducibility of experiments and the implementation of custom optimizations.

`HD-VideoBench` solves of the above mentioned problems and gives researchers in multimedia applications a representative benchmark with a well defined operation environment.

6.1 Benchmarking Video Codecs

The performance of a video codec is a function of the available video coding tools (the coding algorithm itself), the actual implementation of these algorithms, the characteristics of the input sequences, and the architecture in which the codec is implemented. Based on that, in order to make a comprehensive analysis of video applications, a video codec benchmark should meet the following conditions: First, the benchmark should include complete applications (not only kernels) that implement the main features defined in the standard. Second, the benchmarks have to be optimized for high performance. The reference codes are designed for verification purposes and could produce misleading results in complexity or architecture studies. Optimizations can be platform independent (like fast algorithms for motion estimation) and platform dependent (like SIMD optimizations). Third, a complete set of inputs with different resolution, motion characteristics and spatial details have to be provided. Having only one sequence can lead to confusing results in performance evaluations. Fourth, a detailed list of the coding parameters have to be provided. Those parameters have to be tuned for the resolutions under study because the performance of the codecs could change dramatically depending on the selected coding options. Fifth, the programs should be free (as in freedom)

Benchmark	Release Date	Video Applications	Input Sequences
Mediabench I	1997	MPEG-2 dec. (MSSG) MPEG-2 enc. (MSSG)	mei16v2: 352x240 pixels, 30 fps 4 frames YUV sequence: 352x240 pixels
Mediabench+	1999	MPEG-2 dec. (MSSG) MPEG-2 enc. (MSSG) H.263 enc. (Telenor) H.263 enc. (Telenor)	n.a.
Mediabench II	2006	MPEG-2 dec. (MSSG) MPEG-2 enc. (MSSG) MPEG-4 dec. (FFmpeg) MPEG-4 enc. (FFmpeg) H.263 dec. (Telenor) H.263 enc. (Telenor) H.264 dec. (JM 10.2) H.264 enc. (JM 10.2)	704x576, 10 frames, 25fps
Berkeley Multimedia Workload	2000	MPEG-2 enc. (MSSG) MPEG-2 dec. (MSSG)	720x576p, 1280x720p, 1920x1080p (16 frames)
EEMBC Digital Entertainment	2005	MPEG-2 dec. (MSSG) MPEG-2 enc. (MSSG) MPEG-4 dec. (Xvid) MPEG-4 enc. (Xvid)	Graphic: 720x480p30, 50 frames Ralgrind: 320x240p25, 30 frames Sign: 352x240p25, 30 frames Zoom: 320x240p30, 30 frames Marsface: 192x192p25, 49 frames
BDTI Video Benchmarks		H.264 like dec. H.264 like enc.	n.a.

Table 6.1: Description of existing multimedia benchmarks

in order to be able to access the source code, analyze it, perform changes, and be able to distribute them. The same apply for the input sequences. Sixth, the code has to be easy to port between different processor architectures, compilers and operating systems. Finally, the programs must be representative enough of real life multimedia applications, for example as part of multimedia players used in desktop operating systems. The desired characteristics for a video benchmark can be summarized as follows:

- The benchmarks should be complete applications and implement all the features defined in the standards.
- The codecs should be optimized for high performance.
- A complete set of input sequences must be provided.
- A detailed description of the coding parameters must be provided.
- Programs and input sequences should be free.
- The code must be portable.
- Programs must be representative of the multimedia application domain.

6.2 Related Work

Table 6.1 provides a summary of the existing benchmarks for multimedia. Only the applications related to video processing are detailed.

Mediabench [136] is the most popular multimedia benchmark. For the video domain it includes a MPEG-2 encoder and decoder based on the implementation of the MPEG Software Simulation Group (MSSG) with short input videos in low resolution (352x240 pixels). The MSSG codec does not implement SIMD optimizations and, in general, it has low performance. An extension of the Mediabench called Mediabench+ [84] tried to solve the limitations of Mediabench by including MPEG-4 and H.263 video codecs, but they selected the reference implementations (MoMusys and Telenor respectively) and they do not address high definition. Recently, a new version of the Mediabench (called Mediabench II [85]) has been released in which more video codec applications have been added: it includes Codecs for MPEG-2, MPEG-4, H.263 and H.264. The MPEG-2 Codec is the same MSSG implementation, the MPEG-4 is taken from the `FFmpeg` codec library, the H.263 codec is the Telenor implementation, and the H.264 is taken from the reference software (JM-10.5). The main problem with this selection is the combination of reference implementations for some of the codecs (MSSG for MPEG-2 and JM for H.264) with highly optimized version for others (`FFmpeg` for MPEG-4). Although they have increased the resolution compared to the original Mediabench, they do not address HD applications and remains on Standard Resolution (STD). Additionally, Mediabench II provides only one short input sequence (10 frames) and the coding options are not tuned for HD applications.

The Berkeley Multimedia Workload [229] solved the problem of the low resolution of the input sequences by including inputs with higher resolutions, but they have selected only the MSSG implementation of the MPEG-2 Codec. The EEMBC Digital Entertainment [147] benchmark includes codecs for MPEG-2 and MPEG-4 video standards using the MSSG and `Xvid` implementations respectively, they address low and standard resolutions and provide a different set of input sequences. Nevertheless, they do not have recent codecs like H.264/AVC and the source code, coding options and input sequences are not publicly available. Finally, the BDTI Video Encoder and Decoder Benchmark [27] is a set of applications representative of modern video codecs, but they are not complete video codec applications. The codecs seems to be similar to H.264/AVC but the codec details, its sources, coding parameters, and input sequences are not publicly available.

Thus, none of the available benchmarks for multimedia includes all the desired characteristics for a complete benchmark for video codec applications and for HD environments. `HD-VideoBench` try to solve all the before mentioned limitations by providing different a set of different video codec applications optimized for high performance, and providing a complete, and free, set of input sequences and coding options tuned for HD applications.

6.3 The HD-VideoBench Applications

In this section we provide a description of the applications included in HD-VideoBench. A description of the reference implementations of the video standards is included for comparison purposes. Table 6.2 shows a summary of the HD-VideoBench applications.

Application	Description
libmpeg2	MPEG-2 video decoding
ffmpeg-mpeg2	MPEG-2 video encoding
Xvid	MPEG-4 video decoding
Xvid	MPEG-4 video encoding
ffmpeg-h264	H.264 video decoding
x264	H.264 video encoding

Table 6.2: Summary of HD-VideoBench applications

6.3.1 MPEG-2

MSSG: MPEG Software Simulation Group

The MPEG-2 Reference Video Codec [169] is a MPEG-2 codec widely used for benchmarking. Nevertheless, it was designed for the verification of the standard, but not for high performance. Because of that, we have not included it in `HD-VideoBench`.

FFmpeg MPEG-2 Encoder

`FFmpeg` [76] is a free solution to record, convert and stream audio and video. It includes `libavcodec`, a very complete audio/video Codec library that is capable of encoding and decoding streams in many audio and video codecs. It is optimized for high performance with fast algorithms and SIMD extensions for X86, PowerPC and other architectures. It is a widely used library for video and audio encoding and decoding in many free software projects like `MPlayer`, `Xine`, `VideoLAN` and others. As a part of the `FFmpeg`, there is a very fast MPEG-2 encoder which includes SIMD optimizations, parallelization at slice level, and provides very fast algorithms for motion estimation.

Libmpeg2

Although `FFmpeg` includes a MPEG-2 decoder, there is another library called `Libmpeg2` [149] that is faster than the `FFmpeg` implementation. `Libmpeg2` is a free library for decoding MPEG-2 and MPEG-1 video streams. It is highly optimized for high performance and include SIMD optimization of the motion compensation and inverse cosine transform kernels. Due to its high performance, `Libmpeg2` is a very popular decoder used in many free multimedia players, such as `MPlayer`, `Xine` and `VideoLAN`.

6.3.2 MPEG-4

MPEG-4 Reference Code

An ISO reference code of the MPEG-4 video coding standard exists, but it is not convenient for benchmarking due to the same performance reasons mentioned before for other reference implementations.

Xvid

`Xvid` [262] is a free implementation of the MPEG-4 video coding standard that supports the MPEG-4 Advanced Simple Profile (ASP). It has algorithmic optimizations

Test Sequence	Resolutions	Frames/ second	No. frames	Description
Blue_sky	720x576 1280x720 1920x1088	25	100	Top of two trees against blue sky. High contrast, small color differences in the sky. Many details. Camera rotation.
Pedestrian_area	720x576 1280x720 1920x1088	25	100	Shot of a pedestrian area. Low camera position, people pass by very close to the camera. High depth of field. Static camera.
Riverbed	720x576 1280x720 1920x1088	25	100	Riverbed seen through the water. Very hard to code.
Rush_hour	720x576 1280x720 1920x1088	25	100	Rush-hour in Munich city. Many cars moving slowly, high depth of focus. Fixed camera.

Table 6.3: HD-VideoBench input sequences

for motion estimation and SIMD optimizations of the most complex kernels. `FFmpeg` also includes a MPEG-4 encoder that has a similar performance than `Xvid`, but `Xvid` provides a higher coding efficiency. `Xvid` is part of other multimedia benchmarks like *EEMBC* and *Berkeley Multimedia Workload*, and it is widely used in free multimedia players and transcoder applications.

6.3.3 H.264/AVC

H.264/AVC Reference Code

Joint Model (JM) [114] is the reference Codec of the H.264 standardization bodies. It is designed for describing and verifying the standard, and it exhibits very low performance; in fact, it is at least one order of magnitude slower than the FFmpeg implementation [12]. Although being included in *Mediabench II* and *SPEC CPU integer 2006* [93], it is not recommended for performance evaluations.

X264 Encoder

`x264` [259] is a free H.264/AVC encoder. It implements most of the standard features and has a lot of algorithmic optimizations for motion estimation, SIMD optimizations, and allows parallel encoding at slice and frame levels. It is widely used in free encoding applications like `MEncoder`, `GordianKnot` and `VideoLAN`.

FFmpeg H.264 Decoder

`FFmpeg` includes a H.264/AVC decoder that implements most of the features of the standard. The code is highly optimized and include SIMD instructions for the most time consuming kernels. It is also widely used in free multimedia players.

6.4 HD-VideoBench Input Sequences and Coding Options

We have selected three resolutions that are useful for performance analysis in HD video: SD (Standard Definition) (720x576), HD (High Definition) (1280x720) and FHD (Full High Definition) (1920x1088). Four input sequences with different motion (objects and

camera) type and spatial are selected [244]. They were taken with a Sony HDW-F900 digital camera at 1920x1080 pixels resolution, 25 frames per second, progressive scan, and using a 4:2:0 chroma subsampling scheme. Table 6.3 summarizes the main characteristics of the input sequences. Also note that for the 1920x1080 resolution, we have changed the resolution to 1920x1088 in order to avoid cropping and padding operations in some encoders when frame width is not divisible by 16.

The rate control mechanism used by the encoders is based on one-pass constant quality (QP) variable bit rate scheme. We do not use multiple pass or constant bit rate mechanisms because *HD-VideoBench* is for benchmarking the video codecs not the rate control algorithms. The equivalence between the quantization parameter of MPEG-2/-4 and H.264 has derived empirically using Equation 6.1.

$$H264\_QP = 12 + 6 \cdot \log_2 (MPEG\_QP) \tag{6.1}$$

The selected sequence of frames is I-P-B-B. Adaptive placement of B frames is disabled. The only intra frame is the first one. The motion estimation algorithms used are EPZS (Enhanced Predictive Zonal Search) [7] for MPEG-2 and MPEG-4 and Hexagonal Search (HEXS) [273] for H.264/AVC.

6.5 Running HD-VideoBench

At the *HD-VideoBench* web page[1] we provide a complete description of the benchmark, a link for downloading the source code and input sequences, and a script for automating the installation and execution processes. Furthermore, in order to provide a single front end to execute all the video codecs, we have selected the `MPlayer` multimedia application. `MPlayer` is a free media player that includes support for multiple video codecs by using `FFmpeg`, `libmpeg2`, `Xvid` and other multimedia libraries. `MEncoder` is a companion application that can encode audio and video in multiple formats. `MPlayer` (and `MEncoder`) simplifies the process of installing and running multiple video libraries because it selects the appropriate codec and uses it to encode or decode the input video. By default, we have disabled the output of the video to the screen because we are interested in benchmarking the video codecs not the displaying process. Table 6.4 presents a summary of the commands for running the `HD-VideoBench` applications.

6.6 HD-VideoBench Performance

Performance of a video codec can be seen from the point of view of compression and the complexity. From the compression perspective, performance is measured as the ability to compress video efficiently with good quality. From the complexity perspective, performance is measured as the computational resources needed to perform the encoding or decoding processes.

6.6.1 Coding Efficiency

Table 6.5 shows the compression performance of the three video codecs under study. Quality is expressed in terms of the average Peak Signal to Noise Ratio (PSNR) for

[1] http://alvarez.site.ac.upc.edu/hdvideobench/index.html

Codec	Application	Execution Command
MPEG-2 decoder	`libmpeg2`	mplayer mpeg2/576p25_blue_sky.avi -vc mpeg12 -nosound -vo null -benchmark
MPEG-2 encoder	`FFmpeg-mpeg2`	mencoder yuv/576p25_blue_sky.yuv -demuxer rawvideo -rawvideo fps=25:w=720:h=576 -o out/576p25_blue_sky_mpeg2.avi -ofps 25 -ovc lavc -lavcopts vcodec=mpeg2video:vqscale=5:vmax_b_frames=2:subq=8:psnr
MPEG-4 decoder	`Xvid`	mplayer mpeg4/576p25_blue_sky.avi -vc xvid -nosound -vo null -benchmark
MPEG-4 encoder	`Xvid`	mencoder yuv/576p25_blue_sky.yuv -demuxer rawvideo -rawvideo fps=25:format=i420:w=176:h=144 -o out/576p25_blue_sky_mpeg4.avi -ofps 25 -ovc xvid -xvidencopts fixed_quant=5:max_bframes=2:qpel:psnr
H.264 decoder	`FFmpeg-h264`	mplayer h264/576p25_blue_sky.h264 -vc ffh264 -nosound -vo null -benchmark
H.264 encoder	`x264`	x264 –bframes 2 –no-b-adapt –b-bias=0 –ref 16 –qp=26 –analyse all –weightb –me hex –merange 24 –subme 7 –8x8dct -fps 25 –frames 101 –progress -o out/576p25_blue_sky.h264 yuv/576p25_blue_sky.yuv 720x576

Table 6.4: Summary of HD-VideoBench execution commands

all the frames; and compression performance as the bitrate of the resultant compressed video (in Kbit per second). All the videos have almost the same quality because they have been coded with a constant quantization parameter.

Resolution	Input	MPEG-2		MPEG-4		H.264	
		PSNR	bitrate	PSNR	bitrate	PSNR	bitrate
576p25	blue_sky	39.82	3504	38.69	1146	39.248	1095
	pedestrian_area	41.28	2724	40.76	1715	41.141	1382
	riverbed	38.95	10688	39.27	9435	38.456	7783
	rush_hour	42.49	2085	41.41	1217	41.965	1092
720p25	blue_sky	40.97	5541	39.84	2154	40.198	1887
	pedestrian_area	41.89	4783	41.47	3093	41.700	2249
	riverbed	39.70	19729	40.15	17108	39.391	13716
	rush_hour	43.09	3647	42.16	2290	42.649	1872
1088p25	blue_sky	41.81	9462	40.71	4265	40.947	3490
	pedestrian_area	41.93	9360	41.69	6219	41.661	3961
	riverbed	40.07	36475	40.65	31063	39.933	24131
	rush_hour	42.73	7086	42.17	4722	42.496	3357

Table 6.5: HD-VideoBench rate distortion with constant quality

Taken MPEG-2 as the baseline, the MPEG-4 codec achieves, on average for the four input sequences, a 39,4%, 36,7% and 34,1% compression gains at the SD, HD and FHD resolutions respectively. H.264/AVC results in bigger compression ratios 48,2%, 49,5% and 51,8% compared to MPEG-2, and 19,9%, 19,4% and 26,4% compared to MPEG-4 for the three resolutions respectively.

When coding the same video with different values of the quantization parameter we can obtain a rate-distortion curve that allows to compare the coding performance at different data rates. In Figure 6.1 the rate-distortion curve is presented for the FHD version of the blue_sky sequence. At a given quality value, it is clear that H.264/AVC

has a lower bitrate, and at a constant bit-rate the H:264/AVC codec exhibits better quality.

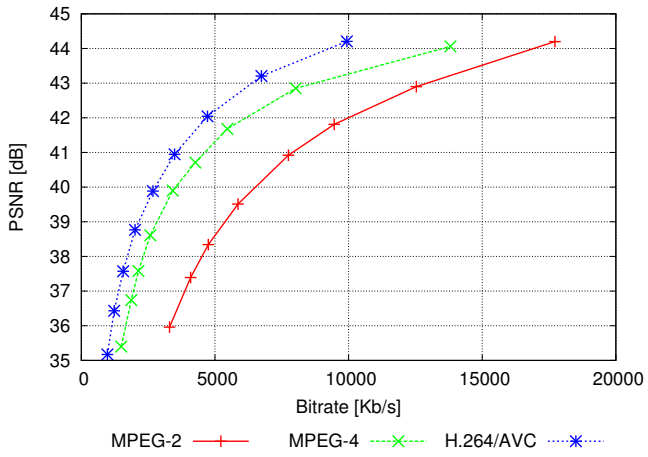


Figure 6.1: HD-VideoBench rate distortion for 1088p25_blue_sky

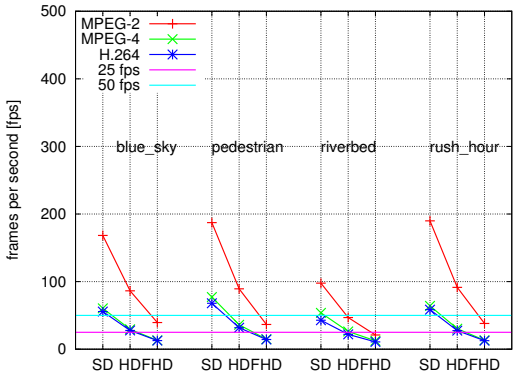
6.6.2 Decoding Performance: Frame Rate

To estimate the average frame rate we have executed the *HD-VideoBench* decoder applications on three different real machines. Two high performance system configurations, one based on the IBM PowerPC-970 processor and the other one on the Intel x86-64 Xeon processor. The third one is a low power system for notebooks and mobile computers based on the Intel x86-64 Atom processor. Table 6.6 shows the main parameters of the three systems used for the experiments.

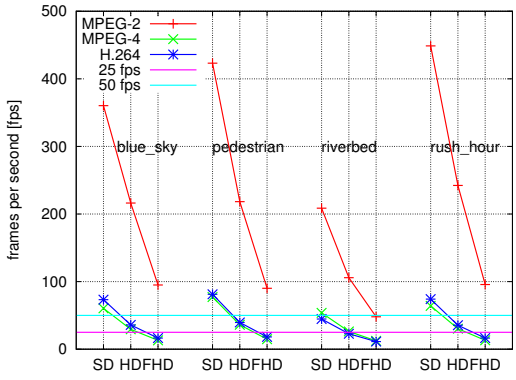
We have evaluated two versions of each benchmark: a scalar version and a version with SIMD optimizations. For each application we compute the average of 5 executions on each machine.

Configuration	X86 High-end	X86 Low-end	PowerPC High-end
ISA	x86-64	x86-64	PowerPC 64-bit
SIMD extensions	MMX, SSE, SSE2, SSE3, SSE4	MMX, SSE, SSE2, SSE3, SSSE3	Altivec
Processor	Intel Xeon L5630	Intel Atom 330	PPC-970 MP
Technology	32nm	45 nm	90 nm
Clock frequency	2.13 GHz	1.6 GHz	2.2 GHz
Power	40 W	8 W	100 W
Level 1 I-cache	64 KB	32KB	
Level 1 D-cache	64 KB	24 KB	
Level 2 cache	1MB	1 MB	1 MB
Level 3 cache	12 MB	n.a.	
Main Memory	24 GB	1.5 GB	4 GB
Operating System	Ubuntu Linux kernel 2.6.15.26	Ubuntu Linux kernel 2.6.32-24	SUSE Linux kernel 2.6.16
Compiler	GCC-4.0.3	GCC-4.4.3	GCC-4.1.2
Compiler Optimizations	-03	-03	-O3

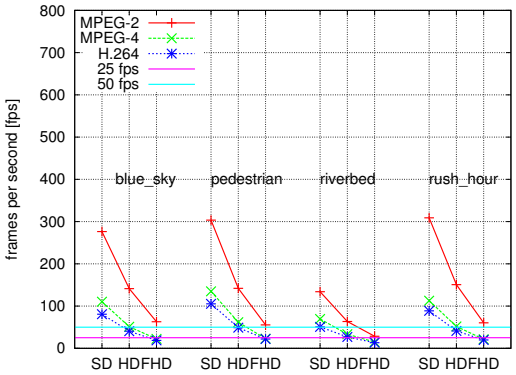
Table 6.6: Experimentation platform



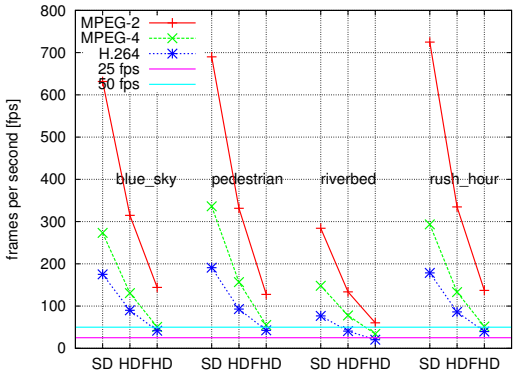
(a) PPC970 scalar version



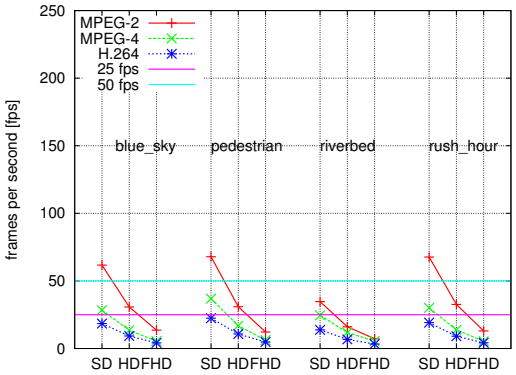
(b) PPC-970 SIMD version.



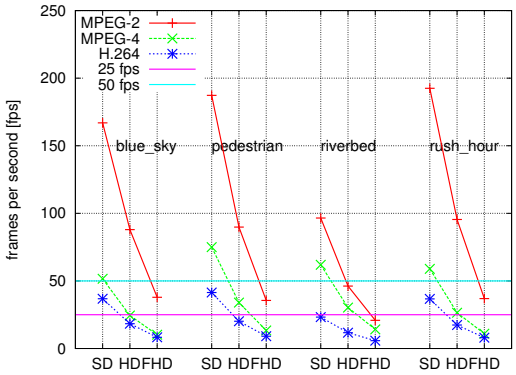
(c) X86-Xeon scalar version



(d) X86-Xeon SIMD version.



(e) X86-Atom scalar version



(f) X86-Atom SIMD version.

Figure 6.2: HD-VideoBench decoding performance

Performance of the Scalar Code

Figure 6.2 shows the performance (in frames per second) of the three codecs under study on three platforms for both scalar and SIMD versions. As a reference we have included the lines of 25 and 50 fps real-time performance.

Figures 6.2a, 6.2c and 6.2e show the scalar decoding performance for the PowerPC-

970, X86-Xeon and X86-Atom systems respectively. In the first two, which are high performance systems, it is possible to process most of the inputs at 25 fps real-time except for MPEG-4 and H.264 at FHD. The X86-Atom platform is an interesting case because it is not able to decode any of the H.264/AVC videos in real-time, even for STD resolution. The same happens for both MPEG-2 and MPEG-4 at FHD.

Impact of SIMD Optimizations

Figures 6.2b, 6.2d and 6.2f show the SIMD decoding performance for the PowerPC-970, X86-Xeon and X86-Atom systems respectively. In the case of H.264 decoding on the PowerPC platform the FFmpeg code was extended with (our) additional SIMD optimizations, like luma and chroma interpolation for small block sizes.

With SIMD optimizations the high-end X86-Xeon platform is able to decode all the streams in real-time, H.264/AVC at FHD which gets an average of 36 fps. The PPC-970 is still not able to decode FHD H.264/AVC in real-time obtaining only 15 fps in average. The x86-Atom architecture with SIMD optimizations can decode H.264/AVC at STD resolution at 35 fps, but HD (17 fps) and FHD (8 fps) are still below the real-time limit.

Architecture	MPEG-2	MPEG-4	H.264/AVC
PPC970	2.37	1	1.22
X86-Xeon	2.22	2.35	1.91
X86-Atom	2.84	2.11	1.88

Table 6.7: Speedup of SIMD optimizations compared to scalar code

The average speedup of SIMD optimizations is shown in Table 6.7. MPEG-2 obtain more benefit from SIMD optimization basically because it has a more regular data layout. In the opposite side, H.264/AVC has the lesser benefits from SIMD optimizations, with less than 2X improve in performance for the whole application. The smaller benefits in the PPC-970 are not due to a limitation of the architecture but a lack of SIMD optimization in the deblocking filter kernel. It is important to note that in this platform the MPEG-4 codec does not get speedup because the Altivec code of the Xvid codec does not run on PowerPC 64-bit platforms.

6.7 Summary

We have presented `HD-VideoBench`, a benchmark devoted to video coding applications and specialized for High Definition. After a careful examination of existing benchmarks for multimedia applications, we have found that none of them have all the required characteristics for a complete benchmark for HD video coding.

`HD-VideoBench` includes codecs for MPEG-2, MPEG-4 and H.264/AVC standards based on open source implementations that have been extensively optimized for high performance. These applications are part of real life programs used in desktop operating systems. By using this kind of applications, we are ensuring the representativeness of the benchmark and, at the same time, we allow the researchers to have full access to the source code. Additionally, we have selected a set of input sequences at HD resolution with different motion and spatial details. We have also analyzed and provided the coding

options that are best suited for HD applications. As a result, `HD-VideoBench` has all the required characteristics for detailed benchmarking of HD digital video applications.

The benchmark has been tested on three different platforms: two of them are high performance systems with superscalar processors and the third one is based on a low power processor for mobile devices.

By using SIMD optimizations it is possible to get some performance improvements. This allow, for example, the system based on the X86-Xeon processor to process H.264/AVC at 36 fps for FHD input videos. Although it is possible to get 25 fps real-time with the latest generation of high performance processors they are not able to scale to higher resolutions or frame rates.

The other two evaluated architectures are still not able to process H.264/AVC at FHD in real-time. The x86-Atom architecture, that uses a processor optimized for low power operation, is the one with the lowest performance, just being able to decode 8 fps for H.264/AVC at FHD.

The speedups that result from SIMD optimization are in the range of 1.22 up to 1.88 for H.264/AVC. These low values suggest that there are some inefficiencies in the data handling of SIMD extensions that do not allow to exploit all the potential DLP efficiently. In the next chapter we will analyze this issue in detail.

7 Support for Unaligned Accesses in SIMD Architectures

As it has been shown in the previous chapters, the speedup that can be obtained by exploiting DLP with SIMD extensions is very low for the H.264/AVC codec. One of the reasons for this is that most SIMD extensions have a limited memory architecture which provides access to only contiguous data in memory, with strong alignment restrictions and a weak support for partial load and stores [53, 226]. These architectures, either do not provide any hardware support for unaligned accesses or provide it but at the expense of a big performance penalty. Therefore, the programmer usually ends up taking care of the alignment in software which, in turn, implies an extra-overhead that reduces or inhibits the performance gains due to vectorization. Moreover, software optimizations such as data reorganization become unsuccessful in video codec applications, where motion estimation (ME) and motion compensation (MC) algorithms and variable block sizes entail unpredictable alignments.

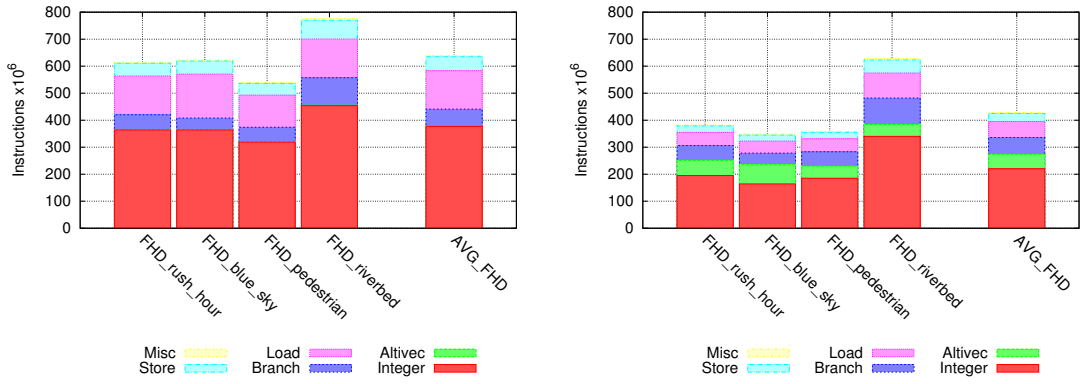
In this chapter we analyze the performance impact of providing hardware support for unaligned access in SIMD extensions for video decoding applications using H.264/AVC. We evaluate an efficient hardware architecture that can deliver high bandwidth and low latency for unaligned accesses. Software support includes new instructions on top of the Altivec SIMD extension of the PowerPC architecture. Our results show that the availability of instructions for unaligned access has an important speed-up in some kernels, and in some cases they allow the vectorization of other kernels that otherwise have to be implemented with scalar instructions.

This chapter is organized as follows: First, we present a justification of the memory alignment issues and we present the problem of alignment in video applications, including an overview of the existing support for unaligned accesses in current SIMD extensions. Next, we describe the process of adding support for unaligned memory access to the Altivec extension both from hardware and software perspectives. After that, we depict the methodology used for the experimental evaluation and next, we present some results in terms of speed-up and reduction in the number of instructions. Finally, we present our main conclusions.

7.1 Motivation: Impact of Overhead Instructions

In order to illustrate the impact of data alignment in video applications, we present here a dynamic distribution of instructions of the H.264/AVC decoder on a PowerPC machine with Altivec SIMD extensions.

Figures 7.1a and 7.1b show the distribution of instructions for the scalar and Altivec versions respectively. The scalar H.264/AVC decoder is dominated by integer, load and branch operations which corresponds to 58.5%, 9.8% and 8.1% of instructions respectively. With the Altivec optimization there is an 1.48X reduction in the instruction



(a) Scalar.

(b) Altivec.

Figure 7.1: Distribution of dynamic instructions for H.264/AVC decoding at FHD

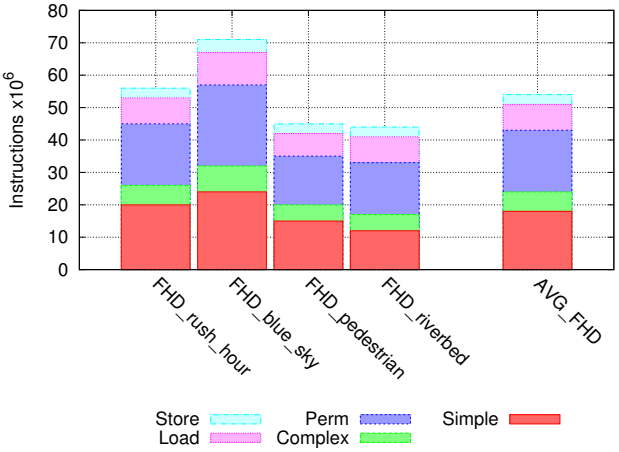


Figure 7.2: Distribution of Altivec instructions

count, mainly in integer and memory operations.

Figure 7.2 shows the distribution of the Altivec instructions. In the Altivec portion of the code, there is a significant amount of permutation instructions (perm: 35.6%) compared to the effective computation ones (simple: 29%, and complex: 11.8%). Permutation instructions are used for re-organizing data to fit properly in SIMD registers. Most of them are used for performing unaligned accesses and constitute an overhead that reduces the efficiency of the SIMD vectorization.

7.2 Current Support for Unaligned Accesses in SIMD Extensions

A memory reference is called misaligned (or unaligned) when it accesses a position that does not match with the memory access granularity of the processor. In most SIMD architectures, it is not possible or it has a big performance penalty to access an

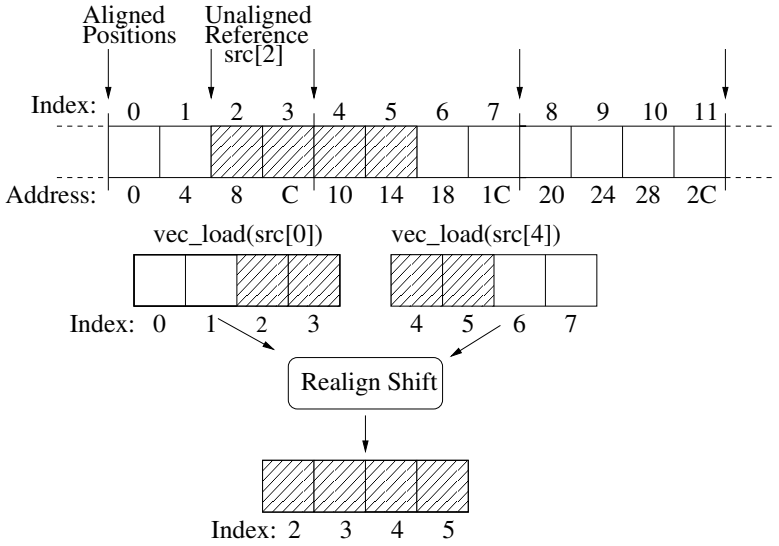


Figure 7.3: Vector load from an unaligned address

unaligned memory position. When there is an attempt to access an unaligned position, it is necessary to perform a realignment process that consist in, first, to read the aligned memory word that is located before the unaligned position and shift out the unnecessary bytes; second, to read the aligned word that is located next to the unaligned position and discard the unnecessary bytes; and finally, to merge the two parts that were extracted previously. The realignment process for an unaligned vector load of four elements is shown in Figure 7.3.

The level of support for unaligned accesses in current SIMD extensions includes variations from hardware mechanisms that transparently perform the memory accesses, system exceptions that generates a call to the operating system, and instructions to do the re-alignment in software. In the domain of high performance general purpose processors (GPPs) the Intel's SSE extension is the only one that includes both hardware support and unaligned exceptions. The initial design of the SSE extension only provides support for aligned accesses, the instruction MOVDQA (Move Aligned Double Quadword) requires that the effective address have to be aligned, and in the opposite case a general protection fault is generated [246]. The SSE2 extension includes support for non-aligned accesses by providing the instruction MOVDQU (Move Unaligned Double Quadword) that allows to load and to store non-aligned 128 bit words. This instruction was implemented using two 64-bit loads (or stores) and was based on microcode; this kind of implementation results in big latencies and big performance penalties for unaligned accesses that cross cache boundaries. In the SSE3 extension another instruction was introduced in order to resolve the above mentioned problems [33]. The LDDQU (Load Unaligned Integer 128 bits) instruction performs a 32-byte load and then performs a shift to extract the corresponding 16 bytes of unaligned data. However this instruction may reduce performance if the load requires store-to-load forwarding and it only applies for loads [65].

In other SIMD extensions like Altivec, MIPS and Alpha, the hardware always returns aligned positions by automatically clearing the lower bits of the effective address. In

```
src_ptr =  InputArray;
LOOP:
  alignmask = vec_lvsl(0,src_ptr);
  aligned_a = vec_ld(0, src_ptr);
  aligned_b = vec_ld(15, src_ptr);
  unaligned = vec_perm(aligned_a, aligned_b, alignmask);
  src_ptr   += srcStride;
END_LOOP
```

(a) Non-unitary stride ((srcStride%16)!=0)

```
src_ptr =  InputArray;
alignmask = vec_lvsl(0,src_ptr);
aligned_a = vec_ld(0, src_ptr);
LOOP:
  aligned_b   = vec_ld(15, src_ptr);
  unaligned   = vec_perm(aligned_a, aligned_b, alignmask);
  aligned_a   = aligned_b;
  src_ptr     += 16;
END_LOOP
```

(b) Stride-one vectors ((srcStride%16)=0)

Figure 7.4: Altivec alignment code for a vector load

these extensions, it is necessary to load the two adjacent aligned positions and to shift them in order to extract the unaligned data elements. SIMD extensions differ in the way they can generate the data necessary for the shift. Nuzman and Henderson call this value the "realignment token" [174]. The realignment token can be an address, a bit mask or any other value that is a function of the unalignment of the original address. The Altivec extension uses an approach in which the realignment token is a vector mask generated with the LVSL (Load Vector for Shift Left) instruction which is used in conjunction with the VPERM (Vector Permute) instruction to merge two aligned vectors and to produce the desired unaligned data [71]. Figure 7.4a shows the necessary code for re-alignment of a vector load, using Altivec C intrinsics.

In the embedded domain also variations exist. Most Digital Signal Processors (DSP) architectures traditionally do not provide support for unaligned accesses. The proliferation of video applications in multimedia devices have prompted the designers to enhance the memory architecture of DSPs with misaligned accesses support. The recent Trimedia TM3270 processor has included support for 32 bit non-aligned loads and stores with no-stall cycles [252]; previous processors in the Trimedia series produce exceptions when trying to access a misaligned position. Due to the fact that the TM3270 has only one load/store unit if the unaligned access crosses a cache line boundary the access may result in two sequential cache misses. In the TMS320C Texas instruments family of DSPs, the recent TMS320C64X set of processors includes support for unaligned loads and stores of 32 and 64 bit values. But when there is an unaligned memory access one of the two memory ports can not be used for memory operations and the memory system does not assure that these memory accesses will be atomic [245]. Other DSP architectures for embedded systems, like the TigerSharc, support accesses to misaligned positions by using specialized hardware units (like the Data Alignment Buffer) which

Architecture & SIMD extension	unaligned load	aligned load	realign operation	realign token
IA32 SSE1,2,3,4	movdqu, lddqu	movdqa		
PowerPC - Altivec		lvx	vperm	lvsl
Cell (PPE) - Altivec	lvlx, lvrx			
MIPS-rev2	ldl, ldr			
MIPS - MDMX		luxc1	alnv.ps	address
ALPHA		ldq_u	extql, extqh, or	address
Trimedia TM3270	ld32r			
TI TMS320C64X	ldnw			

Table 7.1: Support for unaligned loads in different platforms

performs the required aligned loads and shifts [83]. The Cell Broadband Engine has added two instructions for unaligned load and stores into the PowerPC Processor Element (PPE). Using the load instruction an unaligned load requires three instructions (one less than original Altivec) but still two more than a single unaligned load [103]. These instructions belong to the critical path of the loop body representing a significant execution delay. The unaligned store instructions are useful for the leading and trailing edges of misaligned arrays, but not for unaligned 2-dimensional data structures like in video codec applications.

Table 7.1 summarizes the unalignment support provided by different architectures based on the scheme proposed by Nuzman and Henderson and with the addition of some media processors [174].

The first designs of SIMD ISAs did not include support for unaligned accesses because it has been taken as an unnecessary addition of complexity, specially for the load/store pipeline. As a way to overcome the problem of unaligned accesses some architectures, like PowerPC, included powerful permutation instructions and units that help with the problems of data reorganization within a vector register. But still, there are some applications, with video processing being one of the most remarkable one, for those not having an efficient support for unaligned accesses degrade the performance significantly. For this kind of applications the extra cost in hardware complexity is more than justified.

Although currently there are processors that include some extent of support for non-aligned accesses and there are wide consensus about their importance for video applications, most of the current SIMD architectures that support unaligned accesses have restrictions and limitations that do not allow an efficient use of the unaligned instructions in all the cases. These restrictions include: microcode-based operations, short buses in internal datapaths, short bandwidth to the L1 data cache, partial support for unaligned instructions that requires several instructions for each memory access, not-supporting unaligned stores, not being thread safe, causing extra latency for crossing cache boundaries, requiring a sequential handling of more than one cache miss and having restrictions in the use of the load store units. Additionally, there is not in the literature a complete evaluation of the impact of unaligned instructions in SIMD extensions using contemporary multimedia applications. We have addressed this issues by providing a high performance and efficient support for both non-aligned loads and stores and by evaluating their performance impact with the H.264/AVC video codec.

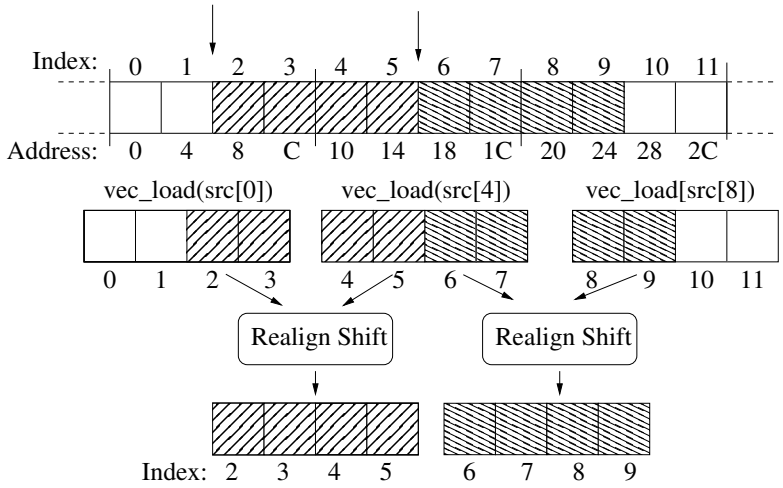


Figure 7.5: Vector load from an unaligned address with stride one

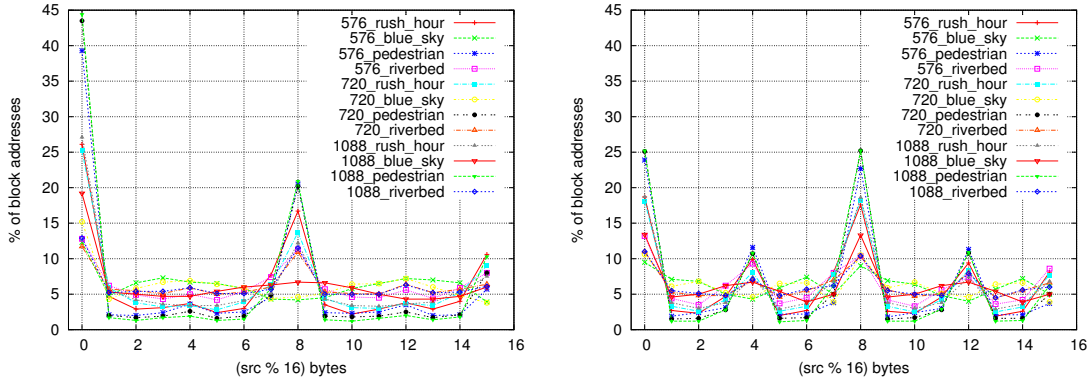
7.2.1 Compiler Optimizations Related to Memory Alignment

As unaligned accesses usually have more latency than aligned ones, the programmer/-compiler tries to avoid them as much as possible. In some algorithms in which unaligned accesses cannot be avoided, compile-time optimizations such as loop peeling and static and dynamic detection of unalignment can still be applied [130, 203]. Additional optimizations exist for stride-one references [72]. Figure 7.4b shows a version of the loop in figure 7.4a optimized for stride-one streams. In this case, the mask for doing the permutation remains constant and has to be calculated only once. Similarly it is possible to reuse one of the aligned loads from one iteration to the next one (see Figure 7.5). As a result the mask generation instructions and one aligned load can be moved out the loop. (Note that in Altivec a stride-one vector means that the stride is equal to 16).

7.2.2 Unaligned Accesses in Video Applications

In video coding and decoding applications there are unpredictable unaligned memory references. This is due to the fact that motion estimation and compensation kernels perform accesses to pixel blocks within a search window. The displacements in the search window can be arbitrary and results in unpredictable unaligned accesses [196]. Additionally, in video codecs like H.264/AVC that supports variable block size, it is necessary to perform unaligned stores in order to save those blocks whose size is not equal to the SIMD register width. In Altivec, 16-bytes of a 16x16 block can be stored simultaneously to an aligned memory address but for other blocks sizes, like 8x8 or 4x4, the data is naturally aligned to 8 or 4 bytes but not to 16-bytes requiring to perform partial stores of unaligned data.

Figures 7.6a and 7.6b show the distribution of unalignment offsets for Altivec loads for two kernels (luma and chroma interpolation) of the MC stage of the H.264/AVC decoder using different input videos at different resolutions. The unalignment offsets are distributed across the full range from 0 (aligned) to 15. These offsets can not be determined at compile time, and the use of optimizations like loop peeling is not well suited. Moreover, these accesses are made on a 2-dimensional pattern preventing the



(a) luma load pointers

(b) chroma load pointers

(c) luma store pointers

(d) chroma store pointers

Figure 7.6: Alignment offsets in H.264/AVC luma and chroma interpolation kernels

use of the compile time optimizations developed for linear streams.

Figures 7.6c and 7.6d show the distribution of unalignment offsets for the Altivec stores for the same kernels and input sets. Here, the unalignment depends only on the block size. The offsets are predictable, and loop peeling can be applied efficiently. The main concern with unaligned stores is that they require a load-store sequence that can take more than 10 assembly instructions for each store and they are not atomic which implies that they are not thread safe. Figure 7.7 shows the Altivec intrinsics for performing an unaligned store. First, there is a sequence of instructions that performs an unaligned load, then the desired data is inserted into the loaded values, and finally, the two modified words are stored back to memory.

7.3 Adding Support for Unaligned Loads and Stores

In order to evaluate the impact of unaligned access support we have added SIMD instructions for unaligned loads and stores. Complete support for unaligned instructions requires modifications both in the load/store pipeline of the processor and in the compiler tool chain.

The new instructions, called LVXU (load vector unaligned indexed) and STVXU

```
dst1    = vec_ld(0, dst);
dst2    = vec_ld(16, dst);
dstperm = vec_lvsr(0, dst);
dstmask = vec_perm(vzero_u8, neg1, dstperm);
rsum    = vec_perm(sum, sum, dstperm);
fdst1   = vec_sel(dst1, rsum, dstmask);
fdst2   = vec_sel(rsum, dst2, dstmask);
vec_st(fdst1, 0, dst);
vec_st(fdst2, 16, dst);
```

Figure 7.7: Altivec alignment code for a vector store

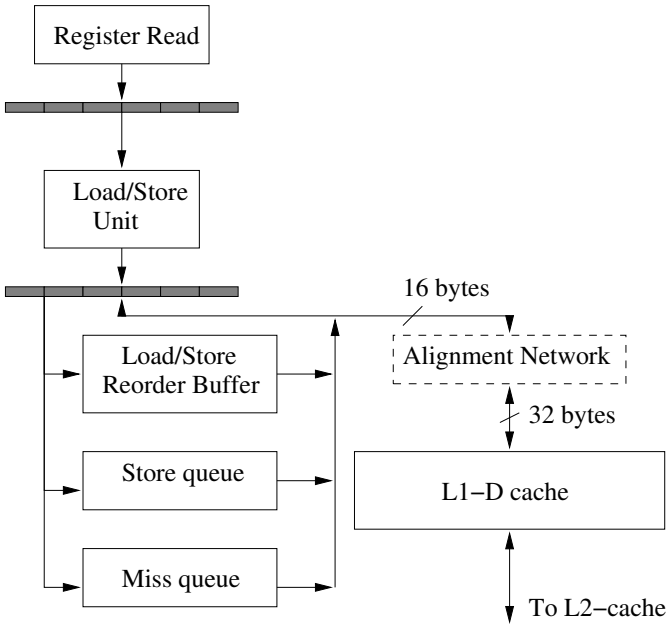


Figure 7.8: Load store pipeline of the modeled superscalar processor

(store vector unaligned indexed), have been added to the Altivec SIMD extension. The instructions are similar to the aligned ones in Altivec; they use indexed addressing but do not impose any alignment restriction on the effective address.

The new instructions can be used directly in assembler programs and, additionally, we have added support for them as intrinsics into the GCC 4.0 compiler allowing to use them in C/C++ programs and leaving the compiler to do the register allocation, instruction scheduling and other optimizations. Direct support for unaligned accesses can be very useful for autovectorizing compilers because it simplifies the alignment detection and correction [174, 72]. Using these instructions, the compiler can find more loops to apply vectorization that in turn can result in bigger speed-ups. Modifications to the GCC auto-vectorizer are not part of this work because the selected kernels were hand optimized for Altivec.

From the processor hardware perspective, it is necessary to adapt the Load Store Unit (LSU) to include a realignment subsystem and to modify the interconnection between

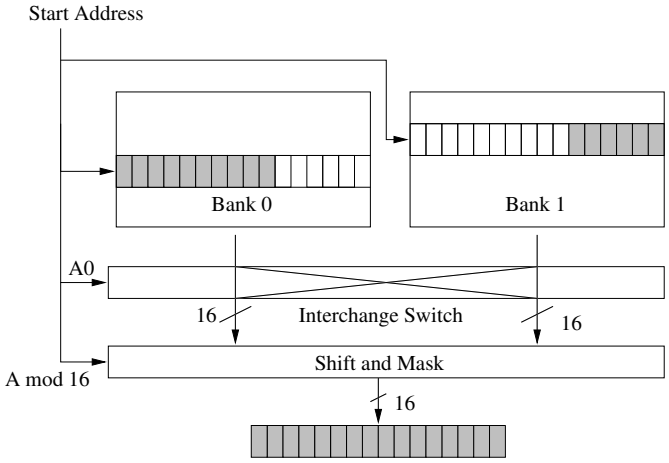


Figure 7.9: Realignment unit using a two-bank interleaved cache

the processor and the L1 data cache (D-L1). Figure 7.8 shows the structure of the LSU with the addition of an alignment network unit. The architecture support for unaligned memory accesses must be added so that it does not severely impact the latency of aligned accesses, and has the minimum possible penalty for unaligned ones. Taken that into account, long latency mechanisms like microcode expansion must be avoided and the bandwidth of the D-L1 must be adapted to the vector accesses. Most of current SIMD implementations use ports to the L1 that have half of the vector width, thus having to perform two or more access to the L1 for each vector reference [102, 191].

The alignment network unit can be designed using a two-bank interleaved cache, so that two consecutive cache lines can be accessed simultaneously, and therefore a whole stride-one vector access, overlapped over two different cache lines, can be performed. This scheme requires three building blocks: an interchange switch, since it may be needed to swap the two cache lines, a shifter to align the lines accessed to the initial address, and a logic to mask the unused data based on the unalignment offset [54] (see figure 7.9). Using this scheme, the unaligned load can be performed in one cycle and the store requires an additional cycle because it first needs to shift and mask the data from the vector register and then to swap the partition for the two cache banks [202, 190]. Using such a scheme does not impose any of the restrictions that most of the current processors that support non-aligned access have. First, there is not a cache line boundary penalty because there are two parallel accesses to the multi-bank cache. Neither are there forwarding restrictions with respect to the other memory operations, and the unaligned accesses are atomic from the processor perspective.

7.4 Methodology

For our experiments we have selected the Altivec/VMX extension of the PowerPC architecture. Altivec is representative of the current SIMD extensions and the results presented here can be extended to other SIMD extensions as well.

To conduct the experiments we are using a trace-driven simulation methodology using the IBM MET tools, that include an instruction emulator and trace generator based on

Configuration	Parameter	2-way In-order	4-way Out-of Order	8-way Out-of Order
Width	Fetch-Rename-Dispatch	2	4	8
	Retire	4	6	12
	Inflight	80	160	255
Units	FX	2	3	6
	FP, LS, BR, VI	1	2	4
	VPERM, VCMPLX	1	1	2
PhysRegs	GPR, FPR, VPR	60	80	128
Queues	BR Issue	5	12	40
	Issue:	10	20	40
	Retire	80	128	160
	Ibuffer	12	24	48
D-cache	Read Ports	1	2	4
	Write Ports	1	1	2
	Mis Max	2	4	8
L1-D	Size		32KB	
	Line Size		128B	
	Associativity		2	
L1-I	Size		32KB	
	Line Size		128B	
	Associativity		1	
L2-(I+D)	Size		1MB	
	Line Size		128B	
	Associativity		8	
	Latency		12 cycles	
Main Memory	Latency		250 cycles	

Table 7.2: Processor configurations used in simulation analysis

the Aria dynamic instrumentation tool, and a cycle-accurate processor simulator based on the Turandot simulator [165].

The applications and kernels are programmed using Altivec intrinsics and compiled with the GCC-4.0.2 compiler [215]. The GCC compiler and GNU assembler have been modified to include intrinsics and opcodes for the new instructions under study. Traces are collected by running the applications on the AIX-5.2 operating system using an Aria based instruction emulator. The execution trace contains PowerPC, Altivec and the new instructions added for unaligned accesses.

We have defined three different processor configurations for the simulations. The first one is a 2-way in-order processor that is somewhat similar to some current embedded media processors like the Cell SPE. The other two configurations are a 4-way and an 8-way out-of-order superscalar processors with a microarchitecture similar to the IBM Power-4 processor with the addition of the Altivec pipeline [243]. The general architecture of the processor is depicted in Figure 7.10. The three configurations have the same number of pipeline stages and the same configuration of the branch predictor and memory hierarchy. The basic parameters of the modeled processors are described in table 7.2.

For our experiments we have used the H.264/AVC decoder and input sequences from the *HD-VideoBench* benchmark that have been described in Chapter 6.

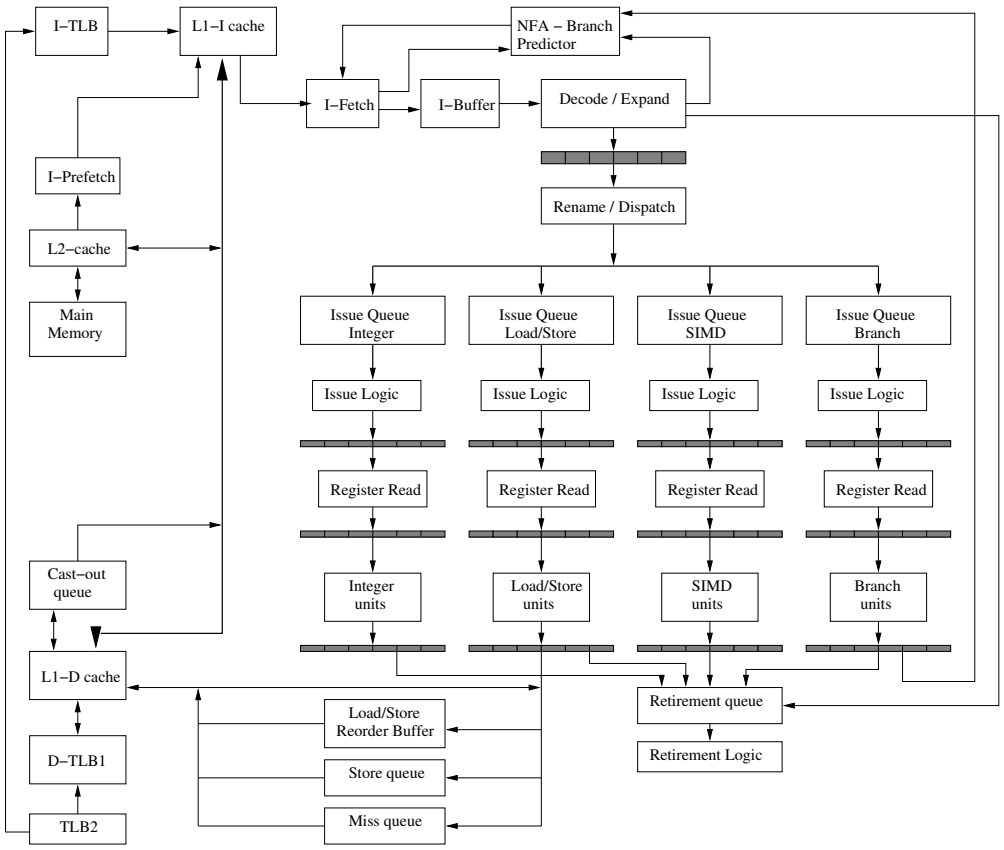


Figure 7.10: General microarchitecture of the simulated processors

Instr. x 1000	Total	Int.	Load	Store	Branches	Altivec Load	Altivec Store	Altivec Simple	Altivec Compl.	Altivec Perm.
LUMA 16×16										
scalar	9,926	6,437	2,693	676	120	0	0	0	0	0
altivec	1,999	269	9	10	85	244	106	564	128	584
unaligned	1,438	155	2	3	51	135	50	564	128	350
CHROMA 8×8										
scalar	2,110	1,439	514	132	25	0	0	0	0	0
altivec	489	108	6	12	34	63	16	64	64	122
unaligned	388	79	6	12	23	40	16	48	64	100
IDCT 4×4										
scalar	4,984	3,074	1,121	608	181	0	0	0	0	0
altivec	2,090	532	49	48	105	112	64	448	0	732
unaligned	1,960	530	49	48	53	112	64	448	0	656
altivec_mat	1,980	486	177	32	105	256	64	128	256	476
mat_unaligned	1,832	450	177	32	53	256	64	128	256	416

Table 7.3: Dynamic instruction count for H.264/AVC kernels (thousands of instructions)

7.5 Performance Evaluation

In order to isolate the effects of unaligned instructions we have extracted some of the most important kernels of the H.264/AVC decoder including: luma interpolation, chroma interpolation, and inverse transform (IDCT). We have implemented all these

kernels for three different block sizes including 16x16, 8x8 and 4x4 pixels. The deblocking filter is an excellent candidate to benefit from unaligned memory access support but at the time of doing these experiments the Altivec version were not yet complete. An analysis of SIMD optimization of the deblocking filtering is presented in [22].

We have evaluated three different implementations of each of these kernels. The first one is a scalar implementation using integer instructions, the second one is the SIMD implementation using Altivec instructions, and the third one is the Altivec implementation extended with unaligned accesses.

7.5.1 Dynamic Instruction Count

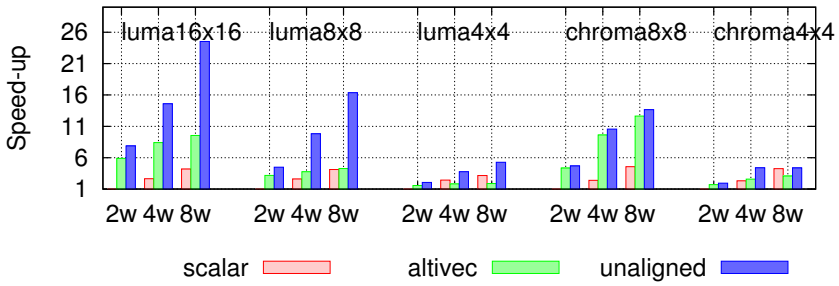
Table 7.3 shows the dynamic instruction count for 1000 executions of each kernel for one block size per kernel (the results for other block sizes are not shown due to space constraints). When comparing the scalar and plain Altivec versions, we appreciate a large reduction in the total number of executed instructions due to vectorization. When comparing the Altivec and the extended Altivec versions, we observe that the use of the new unaligned instructions adds an additional reduction (on average for all the block sizes) of 33.4%, 22.6%, and1.8% for the luma, chroma and IDCT kernels respectively.

The most important instruction reduction comes from the elimination of memory and permutation instructions. But the use of unaligned instructions not only reduces the Altivec memory operations, but also the integer arithmetic and integer load and store instructions, due to the elimination of some pointer arithmetic necessary in the realignment code. Additionally, a number of branches are also eliminated from the Altivec version, because in kernels like chroma interpolation there are branches that depend on the unalignment offset of the address. But, in kernels like IDCT, in which all the input data is properly aligned by rearrangements in the source code, the impact of unaligned instructions only contributes to a small reduction of permutation instructions that are used in the final load-add-store sequence. Additionally, in some kernels there is an additional elimination of branches that were used for peeling the loops in the final store sequence and which are possible to replace with a single unaligned load-store sequence.

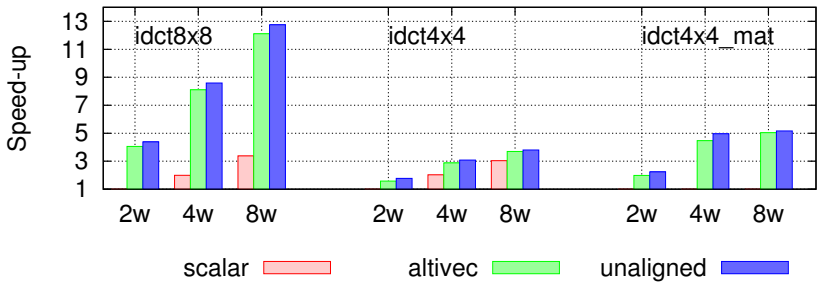
7.5.2 Kernels Speedup

In order to analyze the potential and upper-bound speedups we have made an experiment in which the unaligned accesses have the same latency than the aligned ones. For our architecture that means that they will have a latency of 4 cycles for a D-L1 hit. These results can be taken as an upper-bound of the speed-up achievable with unaligned instructions. The actual values for our proposed realignment network are shown in the next section, but it is important to note that more aggressive implementations are possible in embedded systems (such as the one in the Trimedia-TM3270) in which unaligned accesses are implemented with no stall cycles at all.

Figure 7.11 shows the speed-up in the execution time for all the kernels under study. All the values are normalized to the 2-way scalar version. For the Luma interpolation kernels (fig 7.11a), the Altivec version with unaligned instructions exhibits a 1.9X, 2.6X and 2.1X speed-up for the 16x16, 8x8 and 4x4 block sizes respectively. It is worth to remark that as the block size decreases the overhead of the realignment code in the Altivec



(a) Luma and chroma



(b) IDCT

Figure 7.11: Speedup in kernels with support for unaligned load and stores

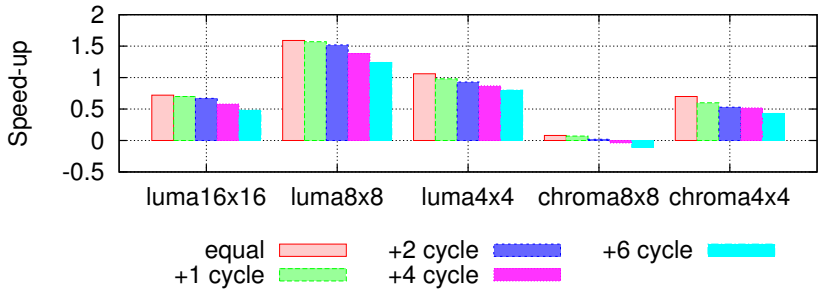
version increases. In the 4x4 case, the scalar implementation has better performance than the Altivec one. In this case, the use of unaligned instructions helps to eliminate a lot of overhead inside the main loop of the interpolation routine, thus allowing to obtain a important speed-up for a kernel that otherwise would not be vectorized.

The chroma interpolation kernel has an average speed-up of 1.1X and 1.25X for the 8x8 and 4x4 sizes respectively. It should be noted that due to the YUV 4:2:0 chroma sub-sampling scheme, used by most current video codecs (included H.264/AVC), the size of the chroma blocks is 8x8, 4x4 and 2x2 pixels. The 2x2 block is not included because the available DLP is very limited.

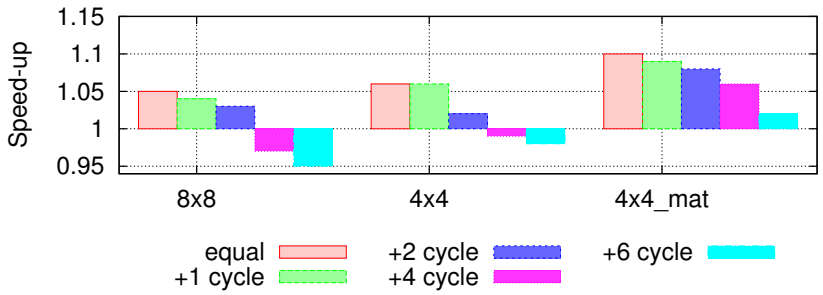
We have evaluated three versions of the IDCT. The first one is the factorized algorithm for the 4x4 block size, in which the speed-up is 1.07X. The second one is based on a matrix product (4x4_altivec_mat version) algorithm and has a speed-up of 1.09X [272]. Finally there is the 8x8 factorized transform in which the speed-up is 1.06X. The impact of unaligned instructions in the IDCT is minimal because, as noted before, the input data structures are properly aligned in software. The unaligned store instruction is useful here for performing partial load and stores required in the output sequence.

7.5.3 Impact of the Latency of Unaligned Load and Stores

The evaluations in the previous section were done assuming that unaligned instructions had the same latency as the aligned ones (i.e 4 cycles in all the processor configurations). In order to analyze the effects of the latency of the hardware realignment network, we have performed an experiment in which the latency of the unaligned loads and stores is



(a) Luma kernel



(b) IDCT kernel

Figure 7.12: Performance impact of latency of unaligned load and stores. Baseline is a system with an equal latency for aligned and unaligned accesses

increased by 1, 2, 4 and 6 extra cycles with respect to the original ones. The resultant speedups compared to the original Altivec implementation are shown in Figure 7.12. Due to space constraints, we only present results for the 4-way processor configuration. Results for 2-wide and 8-wide configurations follow the same pattern.

Luma interpolation is the more insensitive kernel to the latency increase. With one extra cycle, the speed-up decreases only 1.9% on average for the three block sizes. With six extra cycles of latency, the speed-up decreases by 13.2% but still achieves a 1.8X speed-up over the Altivec version.

Chroma interpolation, on the other hand, is more sensitive to the latency of the unaligned load and stores, mainly in the 8x8 block size. The speed-up reduction is very similar to that of the Luma interpolation kernel, ranging from 3.8% for one extra cycle to 17% for 6 extra cycles. But, for the 8x8 block size, when the latency increase in 8 cycles or more, the execution time becomes worse than the original Altivec version. For the 4x4 case, although there is a performance penalty with the latency increase, the use of unaligned instructions results in speed-ups even with high latencies.

In the IDCT kernel the increase of latency affects only a few unaligned memory operations used in the final load-add-store sequence. The latency increase has a bigger impact in the 8x8 version because it has more dependent load and stores in the output sequence. It is important to note that the matrix algorithm not only has a bigger speed-up than the factorized version but additionally tolerates better the latency increase; the speed-up decreases 0.64% (1.09X) and 7.62% (1.02X) with one and 6 more cycles of latency respectively.

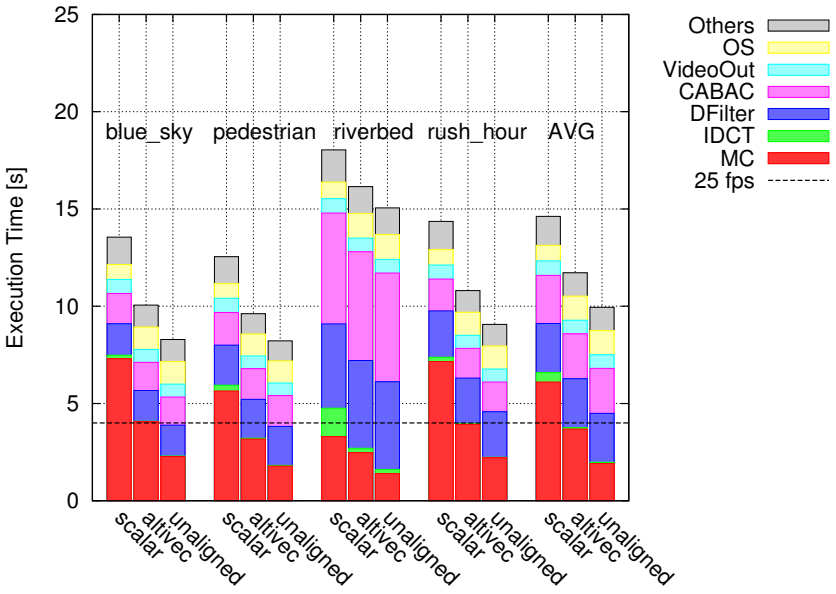


Figure 7.13: Profiling of scalar, altivec and altivec_unaligned H.264/AVC decoder

Summarizing, most of the kernels that use unaligned memory instructions exhibit considerable speed-ups when compared to the original Altivec version until the point where we double the memory latency. Using the proposed hardware design, it is possible to perform a load with just one extra cycle of latency and a store with two cycles (that means 5 and 6 cycles respectively). In such a case, most of the kernels under study achieve a significant speed-up with respect to the original Altivec version.

7.5.4 Complete Application Speedup

Based on a profiling analysis of the H.264/AVC decoder presented in Chapter 5, we estimated the impact of the unaligned memory instructions on the complete application. Figure 7.13 shows a profiling of the H.264 decoder for the scalar, Altivec, and Altivec with unaligned instructions implementations.

The code optimized using unaligned instructions ranges from 1.16X to 1.23X faster compared to the Altivec version. The average speedup is 1.20X compared to the Altivec version, and 1.49X compared to the scalar version. It is important to mention that the deblocking filter has not been optimized with Altivec and accounts for more than 25% of the execution time but it can also benefit from unaligned accesses.

7.6 Summary

We have evaluated the performance impact of extending current SIMD ISAs with instructions for unaligned memory accesses for video codec applications. We have shown that for these kind of applications (but not limited to them) there is a big overhead for the SIMD memory accesses due to presence of unpredictable unaligned memory references. We have shown that this overhead comes from the additional instructions that

are necessary for doing the data realignment in software.

We conclude that instructions for unaligned memory accesses are extremely useful for media SIMD extensions because they allow a significant reduction of the memory access overhead, allow vectorization of codes in which it is necessary to access small amount of data with low latency, and because they also lead to an easier SIMD software development. We also support the approach of having both types of instructions, aligned like in the original Altivec, and unaligned like those evaluated in this study. The original aligned instructions can be used when the alignment is predictable or known at compile time and, in turn, they can be used as a hint to the processor in order to optimize the memory accesses when all the data is aligned.

Other related instructions that can increase the performance of video codec applications are partial load and stores. They are useful for reducing the overhead for loading small data structures like 4x4 blocks.

8 Thread-level Parallelism in Video Decoding

Our previous results have shown that the use of SIMD extensions are insufficient to provide the performance required to process high quality video in real-time. Additionally, the "multi-wall" (problem described Chapter 1) imposes substantial restrictions on the increase of single thread performance. With the trend towards multicore architecture the main way to obtain performance from new processor architectures is with the exploitation of Thread-Level Parallelism (TLP). As a consequence, performance scalability depends on the ability to extract enough TLP from the video decoding applications to use architectures with multi- or many-cores.

In this chapter, we analyze the parallel scalability of video decoding with a special emphasis on H.264/AVC decoding. First, we present the different alternatives that exist for exploiting TLP. We present a detailed analysis of different function- and data-level decomposition approaches. We show that all proposed parallelization strategies such as slice-level, frame-level, and intra-frame macroblock level parallelism, are not sufficiently scalable for future manycore architectures.

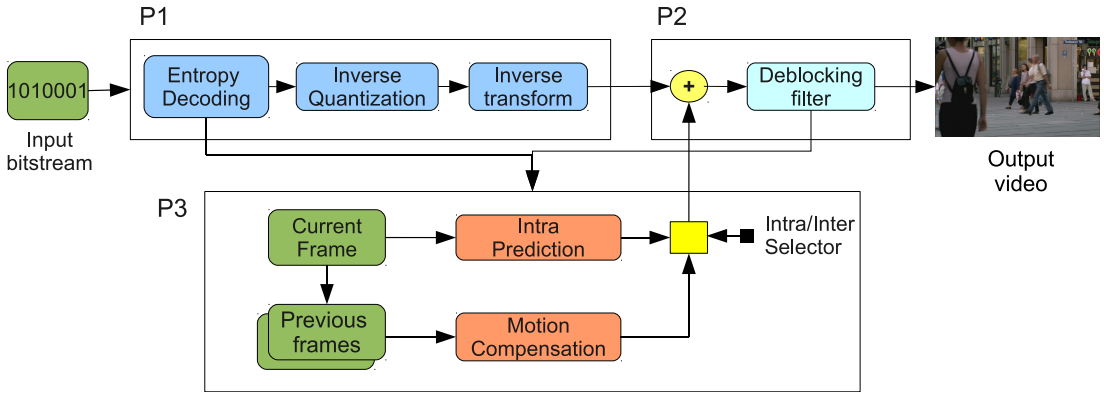
Based on a detailed observation of intra- and inter-frame MB-level data dependencies we propose a new parallelization strategy called Dynamic 3D-Wave. It allows certain MBs of consecutive frames to be decoded in parallel. Using real movie sequences we find a maximum MB parallelism ranging from 4000 to 9000. The results show that H.264 exhibits sufficient parallelism to exploit the capabilities of future manycore CMPs[1].

8.1 Function-level Decomposition

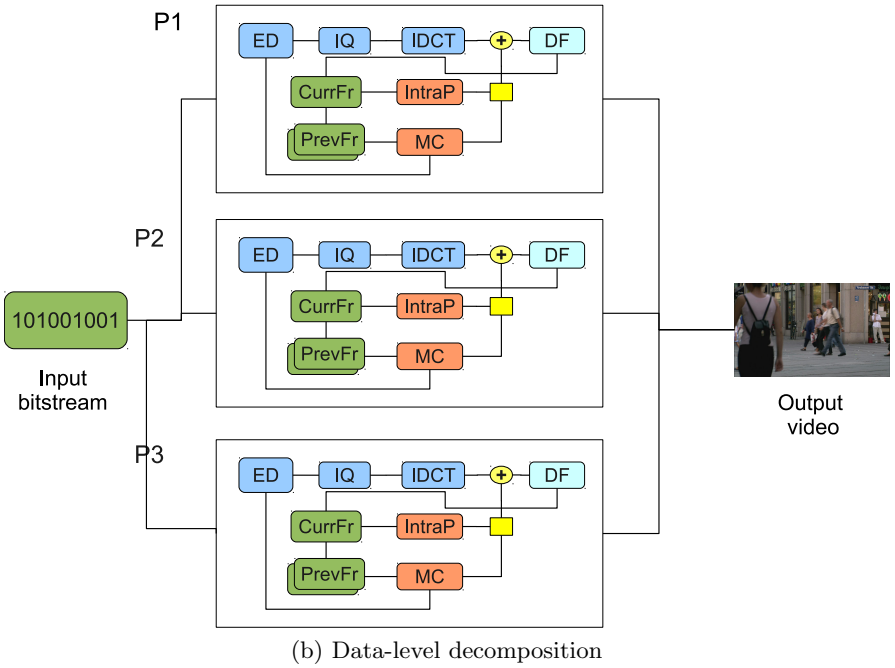
In a function-level decomposition the functional partitions of the algorithm are assigned to different processors. As it was shown in Figure 2.5, the process of H.264/AVC decoding consists of performing a series of operations on data elements that come from the coded input bitstream. Some of these tasks can be done in parallel. For example, Inverse Quantization (IQ) and Inverse Transform (IDCT) can be done in parallel with Motion Compensation (MC). Figure 8.1a shows a possible mapping of the main functional tasks to a 3-processor system. One processor is in charge of Entropy Decoding (ED), IQ and IDCT; another one of the prediction stage (MC or IntraP); and a third one is responsible for the deblocking filter; and additional processor, not shown in figure, is required for general control and synchronization.

Function-level decomposition requires significant communication between the different tasks in order to move the data from one processing stage to the other, and this may become the bottleneck. This overhead can be reduced using double buffering and

[1]This work has been done in cooperation with Cor Meenderink and Arnaldo Azevedo from TU Delft. We include explicit comments when some results are taken from their work



(a) Function-level decomposition



(b) Data-level decomposition

Figure 8.1: Parallelization strategies

blocking to maintain the piece of data that is currently being processed in cache or local memory. Additionally, synchronization is required for activating the different modules at the right time.

The main drawbacks, however, of function-level decomposition are load balancing and scalability. Balancing the load is difficult because the time to execute each task is not known a priori and depends on the data being processed. In a function-level pipeline the execution time for each stage is not constant and some stage can block the processing of the others. Scalability is also difficult to achieve. If the application requires higher performance, for example by going from standard to high definition resolution, it is necessary to re-implement the task partitioning and at some point it could not provide the required performance for high throughput demands.

8.2 Data-level Decomposition

In a data-level decomposition the work (data) is divided into smaller parts and each one is assigned to a different processor. Figure 8.1b shows a possible distribution of the input data into three equal processors. Each processor runs the same program on multiple (different) data elements (SPMD). In H.264/AVC, data decomposition can be applied at different levels of the data structure (see Figure 2.3), which goes down from Group of Pictures (GOP) to frames, slices, macroblocks, and finally to variable sized pixel blocks. Data-level parallelism can be exploited at each level of this data hierarchy, each one having different constraints and requiring different parallelization methodologies.

8.2.1 GOP-level Parallelism

The coarsest grained parallelism is at the GOP level. H.264/AVC can be parallelized at the GOP-level by defining a GOP size of N frames and assigning each GOP to a processor. GOP-level parallelism requires to have access and store all the frames that are part of different GOPs. On the one hand this requires a lot of memory for storing both the compressed and uncompressed data, and on the other hand, it results in a very high latency that cannot be tolerated in some applications. This scheme is therefore not well suited for multicore architectures in which the memory is shared by all the processors and for low latency applications like video conferencing and network streaming.

Scalability of GOP level parallelism in the video decoder depends on ratio of I-frames (that defines the GOP size) which is defined by the encoder. A common value is 4 I-frames per second or less, and usually this ratio change with the input content.

8.2.2 Frame-level Parallelism for Independent Frames

After GOP-level there is frame-level parallelism. In a sequence of frames inside a GOP (see Figure 2.4), some frames are used as reference for other frames (like I and P frames) but some frames (the B-frames in this case) might not. Thus in this case the B-frames can be processed in parallel. To do so, a control processor has to parse the frame headers, identify the independent frames and assign them to different processors.

Frame-level parallelism has scalability problems due to the fact that usually there are no more than two or three B-frames between P frames. This limits the amount of TLP to a few threads. However, the main disadvantage of frame-level parallelism is that, unlike previous video standards, in H.264/AVC B-frames can be used as reference [79]. In such a case, if the decoder wants to exploit frame-level parallelism, the encoder cannot use B-frames as reference. This might increase the bitrate, but more importantly, encoding and decoding are usually completely separated and there is no way for a decoder to enforce its preferences to the encoder.

8.2.3 Slice-level Parallelism

In H.264/AVC, and in most current hybrid video coding standards, each frame can be partitioned into one or more slices. Slices have been included in order to add robustness to the encoded bitstream in the presence of network transmission errors. In order to accomplish this, slices in a frame are completely independent from each other. That

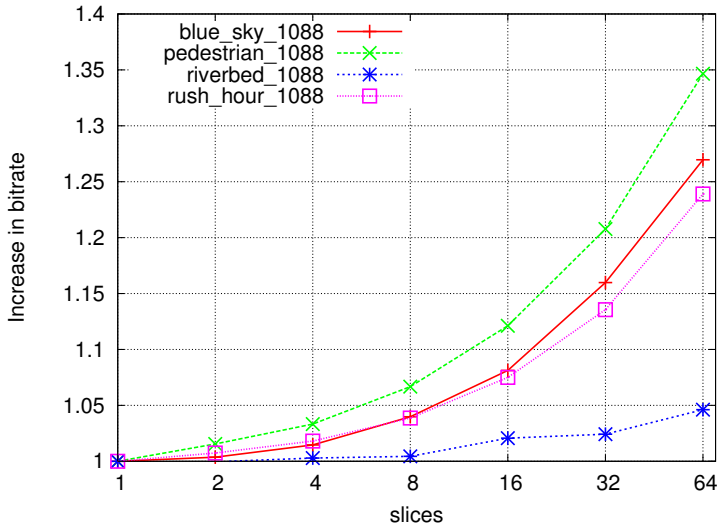


Figure 8.2: Bitrate increase due to slices for 1088p25 inputs.

means that no content of a slice is used to predict elements of other slices in the same frame, and that the search area of a dependent frame can not cross the slice boundary [256, 233]. Although support for slices have been designed for error resilience, it can be used for exploiting TLP because slices in a frame do not have data dependencies. The main advantage of slices is that they can be processed in parallel without ordering constraints and with minimal inter-thread synchronization.

However, there are some disadvantages associated with exploiting TLP at the slice level. The first one is that the number of slices per frame is determined by the encoder. That poses a scalability problem for parallelization at the decoder. If there is no control of what the encoder does then it is possible to receive sequences with one (or few) slice(s) per frame, with a corresponding reduction in the parallelization opportunities.

Furthermore, although slices are completely independent of each other, H.264/AVC includes a deblocking filter that can be applied across slice boundaries. This is an option that is selectable by the encoder, but means that even with an input sequence with multiple slices the deblocking filter crosses slice boundaries, and the filtering process should be performed after the frame processing in a sequential order. This greatly reduces the speedup that could be achieved with slice-level parallelism.

Another problem is load balancing. Usually slices are created with the same number of MBs, and thus can result in an imbalance at the decoder because some slices are decoded faster than others depending on the input content.

Finally, the main disadvantage of slices is the bitrate increase. This is due to the fact that it limits the number of macroblocks that can be used for prediction (motion vector prediction and intra-prediction); it reduces the time that the arithmetic coder has for adapting context probabilities, and finally, it increases the number of headers and start code prefixes used to signal the presence of slices in the bitstream [236].

Figure 8.2 shows the increase in bitrate due to the increase in the number of slices for four different input videos at FHD resolution. The quality is maintained constant at 40 PSNR. When the number of slices increases from one to eight, the increase in bitrate is

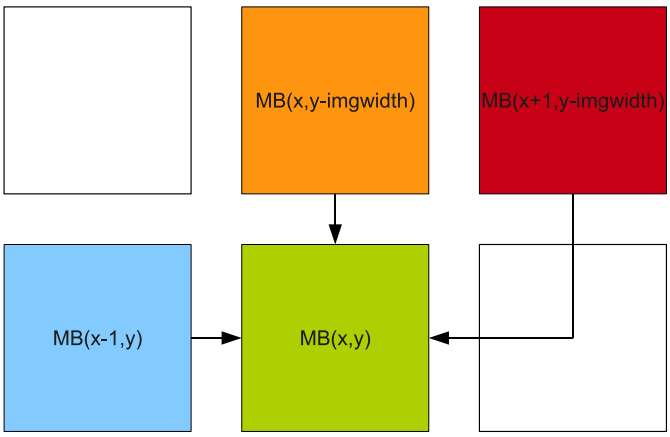


Figure 8.3: Data dependencies at the macroblock level

less than 10%. When going to 32 slices the increase ranges from 3% to 24%, and when going to 64 slices the increase ranges from 4% up to 34%. This increase in the bitrate is unacceptable because in some cases it is in the same range than the compression gains of H.264/AVC compared to previous video coding standards. As a consequence, a large number of slices is not possible.

As shown in the figure, the increase in bitrate depends heavily on the input content. The *riverbed* sequence has a higher density of I-macroblocks, and thus has a large absolute bitrate compared to the other three sequences. Thus, the relative increase in bitrate is much lower than the others. A side consequence can be derived: input videos with poor motion compensation performance can use large number of slices and benefit from a slice-level parallelization.

8.2.4 Macroblock-level Parallelism

The next level in the data hierarchy is the MB. However MBs have intra- and inter-frame dependencies that restrict the processing order and therefore the amount of parallelism that can be extracted. MB-level parallelism can be exploited inside a frame if the dependencies at different kernels within the frame are satisfied. Additionally, MB-level parallelism can be extracted between frames if, in addition to the intra-frame dependencies, inter-prediction dependencies are satisfied. Below we will describe both approaches in detail.

Macroblock-level Parallelism in the Spatial Domain (2D-Wave)

To exploit parallelism between MBs inside a frame it is necessary to take into account the dependencies between them. In H.264/AVC, motion vector prediction, intra-prediction, and the deblocking filter use data from the left, top, and right-top MBs as shown in Figure 8.3. Usually, MBs in a slice are processed in scan order, which means starting from the top left corner of the frame and moving to the right, row after row. In this way, MB dependencies are naturally preserved. Processing MBs in a diagonal wavefront manner satisfies all the dependencies and, at the same time, allows to exploit parallelism between MBs. We refer to this parallelization technique as 2D-Wave.

Resolution		Max. Par. MBs
QCIF	176×144	6
CIF	352×288	11
SD	720×576	23
HD	1280×720	40
FHD	1920×1088	60

Table 8.1: Maximum parallel MBs for several resolutions using the 2D-Wave approach

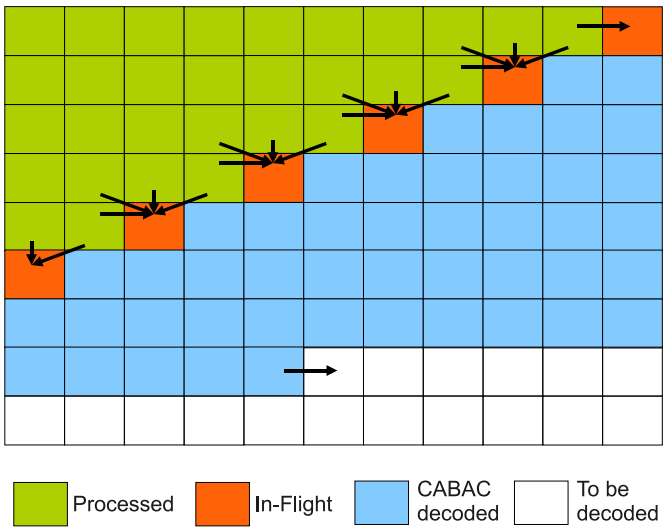
Figure 8.4a depicts an example of QCIF frame (11×9 MBs, 172×144 pixels). After some initial ramp-up, 6 MBs can be processed in parallel. The figure also shows the dependencies that need to be satisfied in order to process each of the parallel MBs. The number of independent MBs in each frame depends on the resolution. Table 8.1 shows the number of independent MBs for different resolutions. For a low resolution like QCIF there are only 6 independent MBs during 4 time slots. For FHD there are 60 independent MBs during 9 slots of time. Figure 8.4b depicts the available MB parallelism over time for the three resolutions under study, assuming that the time to decode a MB is constant.

MB-level parallelism in the spatial domain has some advantages over other schemes for parallelization of H.264. First, this scheme can have a relatively good scalability. As shown before, the number of independent MBs increases with picture resolution. Second, it is possible to achieve a good load balancing if a dynamic scheduling system is used. That is due to the fact that the time to decode a MB is not constant and depends on the data being processed. Load balancing could take place if a dynamic scheduler assigns a MB to a processor once all its dependencies have been satisfied. Additionally, because in MB-level parallelization all the processors/threads run the same program the same set of software optimizations (for exploiting ILP and SIMD) can be applied to all processing elements.

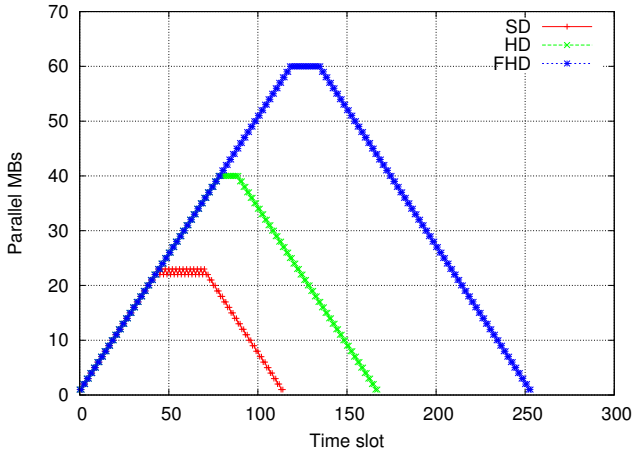
However, this kind of MB-level parallelism has some disadvantages. The first one is the fluctuating number of independent MBs (see Figure 8.4) causing underutilization of cores and decreased total processing rate. The second disadvantage is that entropy decoding cannot be parallelized at the MB level. MBs of the same slice have to be entropy decoded sequentially, which means that only after entropy decoding has been performed the parallel decoding of MBs can start. If entropy decoding is accelerated with specialized hardware, MB-level parallelism could still provide benefits. This problem is considered in detail later.

Macroblock-level Parallelism in the Temporal Domain

In the decoding process the dependency between frames is due to the Motion Compensation (MC) module only. In this stage, the reconstruction of the MB is done using a reference area in a previous frame pointed by a motion vector (MV). MV length is limited by two factors: the H.264/AVC level and the motion estimation algorithm. The H.264/AVC level defines, amongst others, the maximum length of the vertical component of the MV which are 512 pixels vertical and 2048 pixels horizontal [92]. In practice, the motion estimation algorithm defines a maximum search range of dozens of pixels, because a larger search range is very computationally demanding and provides only a small benefit [196].



(a) Data dependencies



(b) Independent macroblocks

Figure 8.4: MB parallelism for a single frame using the 2D-Wave approach

When the reference area has been decoded it can be used by the referencing frame. Thus it is not necessary to wait until a frame is completely decoded before decoding the next frame. The decoding process of the next frame can start after the reference areas of the reference frames are decoded. Figure 8.5 shows an example of two QCIF frames where the second depends on the first. MBs are decoded in scan order and one at a time. The figure shows that one MB in the frame $i+1$ depends on a region that covers 4 MBs in the frame i. After those reference MBs have been decoded, the MB in the frame $i+1$ can be decoded even though frame i is not completely decoded.

The main disadvantage of this scheme is the relatively limited scalability. The number of MBs that can be decoded in parallel is inversely proportional to the length of the vertical motion vector component. Thus for this scheme to be beneficial the encoder should be enforced to heavily restrict the motion search area which in far most cases is not possible. Assuming it would be possible, the minimum search area is around 3

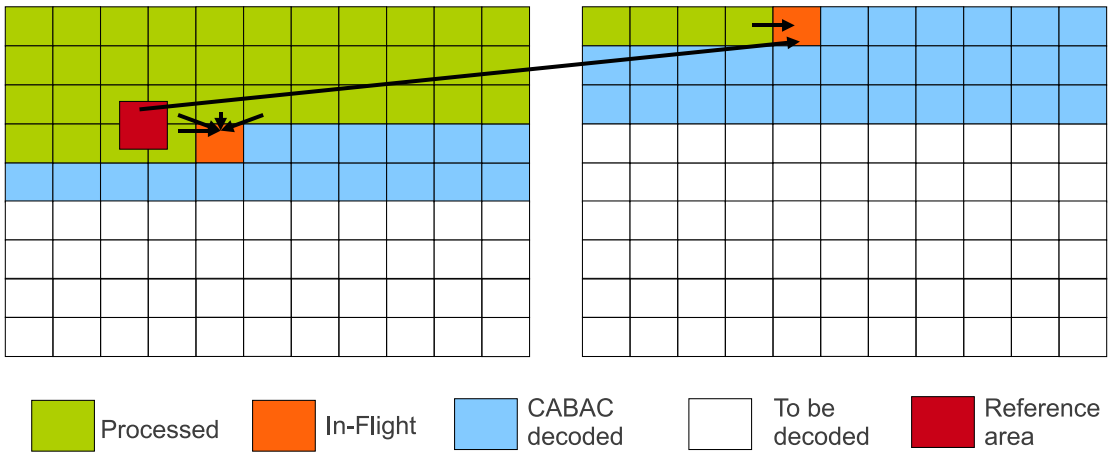


Figure 8.5: MB-level parallelism in the temporal domain in H.264/AVC

MB rows: 16 pixels for the co-located MB, 3 pixels at the top and at the bottom of the MB for sub-sample interpolations and some pixels for motion vectors (at least 10). As a result the maximum parallelism is 14, 17 and 27 MBs for STD, HD and FHD frame resolutions respectively.

The second limitation of this type of MB-level parallelism is poor load-balancing because the decoding time for each frame is different. It can happen that a fast frame is predicted from a slow frame and can not decode faster than the slow frame and remains idle for some time.

Finally, this approach works well for the encoder who has the freedom to restrict the range of the motion search area. In the case of the decoder the motion vectors can have large values (even if the user ask the encoder to restrict them as we will show later) and the number of frames that can be processed in parallel is reduced.

This approach has been implemented in the X264 open source encoder [259]. In this case the encoder limits the vertical motion search range in order to allow a next frame to start the encoding before the current one completes. This approach does not include macroblock-level parallelism and thus only one macroblock per frame at a time is encoded/decoded. Figure 8.6 shows the performance of the X264 encoder in a ccNUMA multiprocessor (SGI Altix with Itanium 2 processors) for the blue_sky FHD input sequence. As a reference, the frame per second results are also displayed. With 4 threads the speedup is 3.48X (efficiency of 87%), with 16 threads speedup is 11.34X (efficiency of 71%) and with 32 threads the speedup is 16.4X (efficiency of 52%) a point from which the performance improvement is saturated.

Combining Macroblock-level Parallelism in the Spatial and Temporal Domains (3D-Wave)

None of the single approaches described in the previous sections scales to future many-core architectures containing 100 cores or more. There is a considerable amount of MB-level parallelism, but in the spatial domain there are phases with a few independent MBs, and in the temporal domain scalability is limited by the height of the frame.

In order to overcome these limitations it is possible to exploit both temporal and spa-

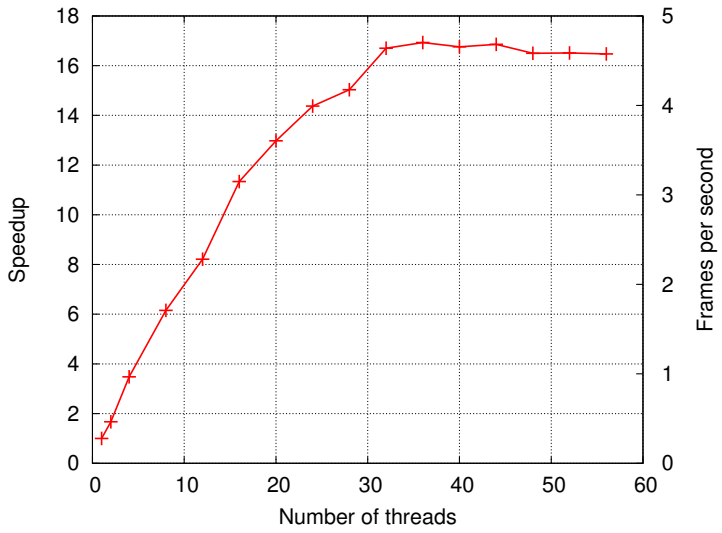


Figure 8.6: Parallel scalability of X264 encoder on an Itanium2 ccNUMA multiprocessor using temporal MB-level parallelism

tial MB-level parallelism. Inside a frame, spatial MB-level parallelism can be exploited using the 2D-Wave scheme mentioned previously. And between frames temporal MB-level parallelism can be exploited simultaneously. Adding the inter-frame parallelism (time) to the 2D-Wave intra-frame parallelism (space) results in a combined 3D-Wave parallelization.

3D-Wave decoding requires a scheduler for assigning MBs to processors. This scheduling can be performed in a static or a dynamic way. Static 3D-Wave exploits temporal parallelism by assuming that the motion vectors have a restricted length, based on that it uses a fixed spatial offset between decoding consecutive frames. Although this approach is suitable for the encoder, it is not so much for the decoder. As it is the encoder that determines the MV length, the decoder has to assume the worst case scenario and a lot of parallelism would be unexploited.

Our proposal is the Dynamic 3D-Wave approach, which to the best of our knowledge has not been reported before. It uses a dynamic scheduling system in which MBs are scheduled for decoding when all the dependencies (intra-frame and inter-frame) have been satisfied. Figure 8.7). The Dynamic 3D-Wave system results in a better thread scalability and a better load-balancing. A more detailed analysis of the Dynamic 3D-Wave is presented in Section 8.4.

8.2.5 Block-level Parallelism

Finally, the finest-grained data-level parallelism is at the block-level. Most of the computations of the H.264/AVC kernels are performed at the block level. This applies, for example, to the interpolations that are done in the motion compensation stage, to the IDCT, and to the deblocking filter. This level of data parallelism maps well to SIMD style of computation [272, 221, 12, 139]. SIMD parallelism is orthogonal to the other levels of parallelism described above and because of that it can be mixed, for example,

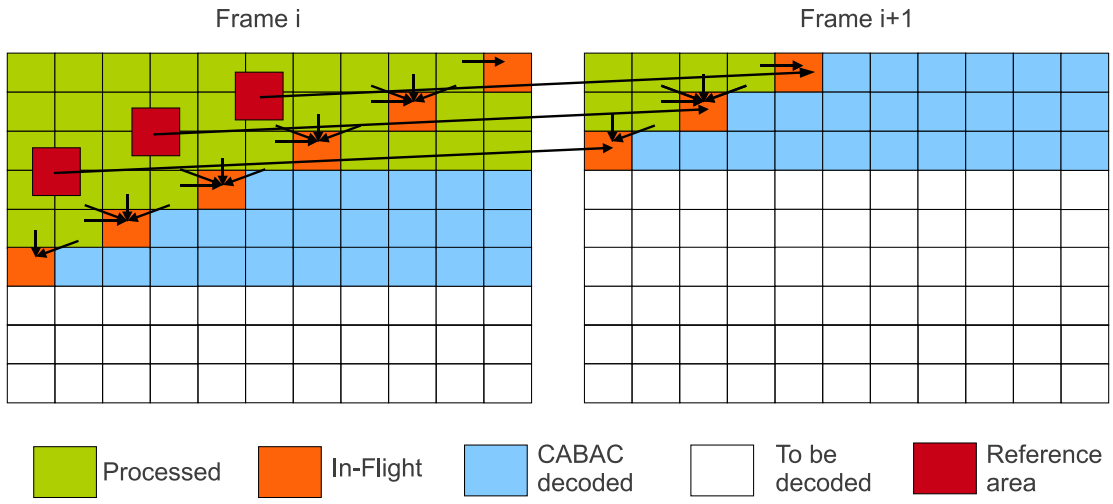


Figure 8.7: 3D-Wave strategy: intra- and inter-frame dependencies between MBs for two QCIF frames

with MB-level parallelization to increase the performance of each thread.

A possible form of block-level parallelism not considered in this work could consist on decoding blocks within a MB in parallel. For example, if a MB consists of 4 8×8 blocks, those blocks can be processed independently provided that their dependencies are satisfied. The main limitations of this strategy are the scalability and load balancing because the number of blocks in a MB depends on the input content and the operations inside each block can be different. Additionally, this fine-grain form of parallelism can suffer more from threading overhead. A similar approach could be the exploitation of parallelism in color structure. Processing of Luma and two Chroma blocks can be done independently. This form of parallelism can be used to increase the scalability of MB-level parallelism, for example by doing MB-level parallelization across cores and color-level parallelization inside each core with multithreading capabilities.

8.3 Parallel Scalability of the Static 3D-Wave

In the previous section we suggested that using the 3D-Wave strategy for decoding H.264 would reveal a large amount of parallelism. We also mentioned that two strategies are possible: a static and a dynamic approach. Zhao and Liang used a method similar to the Static 3D-Wave for encoding H.264 which so far has been the most scalable approach to H.264 coding. However, a limit study to the scalability of this approach is lacking. In order to compare our dynamic approach to this, in this section we analyze the parallel scalability of the Static 3D-Wave.[2].

The Static 3D-Wave strategy assumes a static maximum MV length and thus a static reference range. Figure 8.8 illustrates the reference range concept, assuming a MV range of 32 pixels. The orange MB in $Frame\ i+1$ is the MB currently considered. As the MV can point to any area in the range of [-32,+32] pixels, its reference range is the red

[2]The text and results in this section are based mainly in the work of A. Azevedo from TU Delft and are presented here for completeness of this chapter

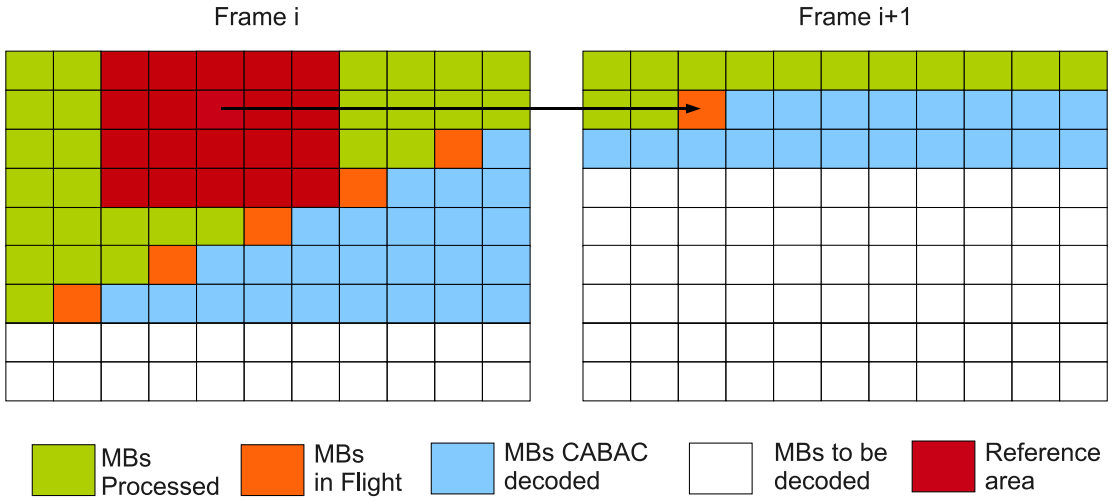


Figure 8.8: Reference range example: read area on frame i is the reference range of the current MB of frame i+1

area in *Frame i*. In the same way, every MB in the wave front of *Frame* $i+1$ has a reference range similar to the presented one, with its respective displacement. Thus, if a minimum offset, corresponding to the MV range, between the two wavefronts is maintained, the frames can be decoded in parallel.

8.3.1 Estimating the Maximum Parallelism

For calculating the number of parallel MBs it is assumed that processing a MB requires one time slot. This is a simplification that helps to estimate the upper bound of parallelism. In reality, different MBs require different processing times and this reduces the parallel scalability. Furthermore, the following conservative assumptions are made to calculate the amount of MB parallelism. First, B frames are used as reference frames. Second, the reference frame is always the previous one. Third, only the first frame of the sequence is an I-frame. These assumptions represent the worst case scenario for the Static 3D-Wave.

The number of parallel MBs is calculated as follows. First, we calculate the MB parallelism function for one frame using the 2D-Wave approach, as in Figure 8.4. Next, given a MV range, we determine the required offset between the decoding of two frames. Finally, we added the MB parallelism function of all frames using that offset.

Formally, let $h_0(t)$ be the MB parallelism function of the 2D-Wave inside frame 0. The MB parallelism function of frame i is computed as $h_i(t) = h_{i-1}(t - \mathit{offset})$ for $i > 1$. The MB parallelism function of the total Static 3D-Wave is given by $H(t) = \sum_i h_i(t)$.

The offset is calculated as follows. For motion vectors with a maximum length of 16 pixels, it is possible to start the decoding of the second frame when the MBs (0,0), (0,1), (1,0), and (1,1) of the first frame have been decoded. Of these, MB (1,1) is the last one decoded. Thus, the next frame can be started decoding at time slot T5, resulting in an offset of 4 time slots. Similarly, for maximum MV ranges of 32, 64, 128, 256, and 512 pixels, we find an offset of 7, 13, 25, 49, and 97 time slots, respectively. In general, for

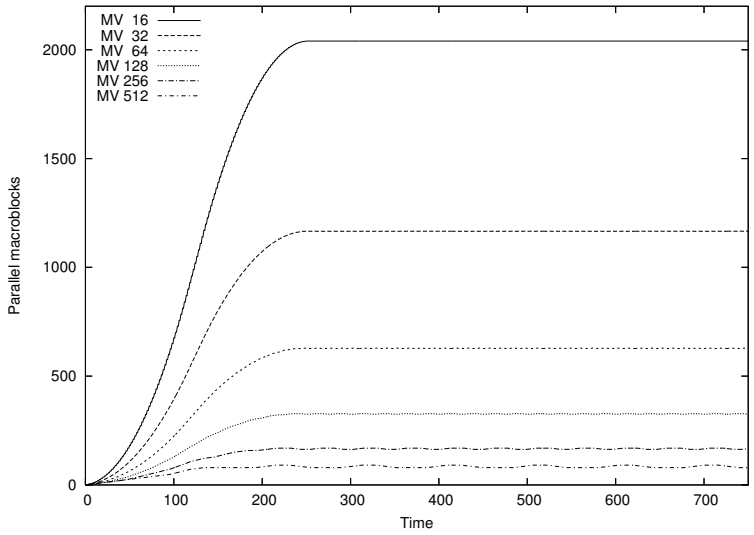


Figure 8.9: Static 3D-Wave: number of parallel MBs for FHD resolution using several MV ranges.

MV range	Max. Par. MBs			Max Frames-in-flight		
	SD	HD	FHD	SD	HD	FHD
512	-	40	91	-	2	3
256	34	76	169	3	4	6
128	66	145	328	5	7	11
64	126	277	629	9	13	20
32	232	515	1166	17	24	37
16	414	900	2040	29	42	64

Table 8.2: Static 3D-Wave: maximum MB parallelism for several MV ranges and all three resolutions.

a MV range of n pixels, ($\lceil n/16 \rceil$ MBs) the offset is $1 + 3 \times \lceil n/16 \rceil$ time slots.

8.3.2 Theoretical Results

We investigate the maximum amount of available MB parallelism using the Static 3D-Wave strategy. For each resolution we applied the Static 3D-Wave to MV ranges of 16, 32, 64, 128, 256, and 512 pixels. The range of 512 pixels is the maximum vertical MV length allowed in level 4.0 (HD and FHD) of the H.264/AVC standard.

Figure 8.9 depicts the amount of MB-level parallelism as a function of time, i.e., the number of MBs that could be processed in parallel in each time slot for a FHD sequence using several MV ranges. The graph depicts the start of the video sequence only. At the end of the movie the MB parallelism drops similar as it increased at startup. For large MV ranges, the curves have a small fluctuation which is less for small MV ranges.

The shape of the curves for SD and HD resolutions are similar and, therefore, are omitted. Instead, Table 8.2 presents the maximum MB parallelism for all resolutions. The table also includes the required number of frames-in-flight for achieving the corresponding number of independent MBs.

	HD				FHD			
	MBs	frames	mv max	mv avg	MBs	frames	mv max	mv avg
rush_hour	2831	139	435	1.8	6133	218	441	2.2
riverbed	4579	228	496	2.2	9169	304	569	2.6
pedestrian	2807	151	554	11	4851	242	554	9.9
blue_sky	2873	140	298	5.1	7327	253	498	5.6

Table 8.3: Maximum available MB parallelism, and frames in flight for normal encoded movies. Also the maximum (mv max) and the average (mv avg) motion vectors (in square pixels) are stated.

The results show that the Static 3D-Wave offers a huge parallelism if the MV length is restricted to a small value and if a high number of frames in flight is allowed. However, in most cases these restrictions cannot be guaranteed and the maximum MV length has to be considered. In such a case, the parallelism drops significant towards the level of the 2D-Wave. The difference is that the Static 3D-Wave has a sustained parallelism while the 2D-Wave approach has little parallelism at the beginning and at the end of processing a frame.

8.4 Parallel Scalability of the Dynamic 3D-Wave

In this section we analyze the maximum available MB parallelism of the dynamic 3D-Wave strategy. This experiments do not consider any practical or implementation issues, but simply explores the limits to the parallelism available in the application[3].

To investigate the amount of parallelism offered by the Dynamic 3D-Wave we analyzed the dependencies of each MB and assigned it a timestamp as follows. The timestamp of a MB is simply the maximum of the timestamps of all MBs upon which it depends (in the same frame as well as in the reference frames) plus one. Because the frames are processed in decoding order not in display order, and within a frame the MBs are processed in scan order, the MB dependencies are observed and it is assured that the MBs on which a MB B depends have been assigned their correct timestamps by the time the timestamp of MB B is calculated. As before, we assume that it takes one time slot to decode a MB.

For each time slot we analyze, first, the number of MBs that can be processed in parallel during that time slot; second, we keep track of the number of frames in flight; and finally, we keep track of the motion vector lengths. Although a comparison of different motion estimation methods was made, here we only present results for the hexagonal fast search algorithm [273] (A more detailed analysis of the effect of motion estimation algorithms can be found in [160])

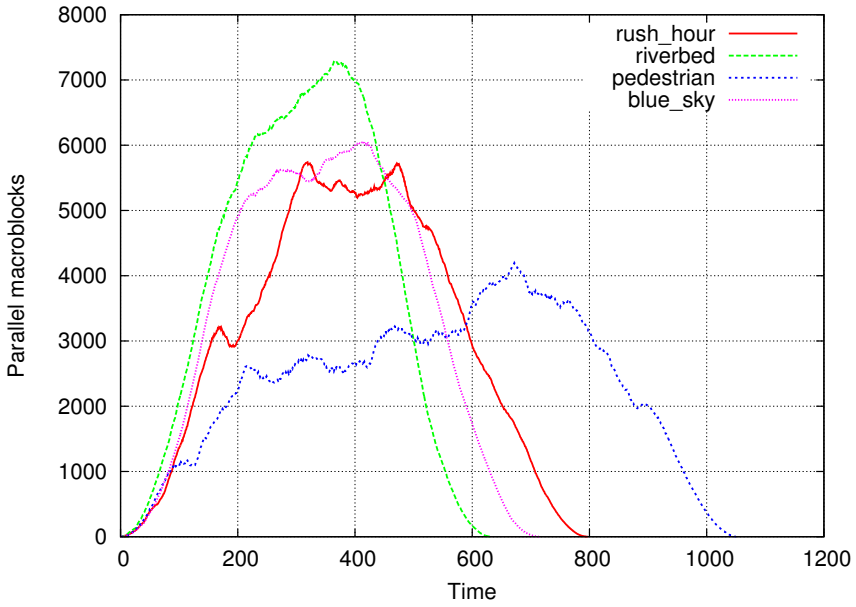


Figure 8.10: Number of parallel MBs for FHD resolution using dynamic 3D-Wave with hexa encoding

8.4.1 Maximum Parallelism

Table 8.3 summarizes the results and shows that a huge amount of MB parallelism is available. Figure 8.10 depicts the MB parallelism time curve for FHD. For the other resolutions the time curves have a similar shape. The MB parallelism time curve shows a ramp up, a plateau, and a ramp down. For example, for *rush_hour* the plateau starts at time slot 300 and last until time slot 650. *Riverbed* exhibits so much MB parallelism that is has a small plateau. Due to the stochastic nature of the movie, the encoder mostly uses intra-coding, resulting in very few dependencies between frames. *Pedestrian* exhibits the least parallelism. The fast moving objects in the movie result in many large motion vectors. Especially objects moving from right to left on the screen causes large offsets between MBs in consecutive frames.

Rather surprising is the maximum MV length found in the movies (see Table 8.3). The search range of the motion estimation algorithm was limited to 24 pixels, but still lengths of more than 500 square pixels are reported. According to the developers of the X264 encoder this is caused by drifting [260]. For each MB the starting MV of the algorithm is predicted using the result of surrounding MBs. If a number of MBs in a row use the motion vector of the previous MB and add to it, the values accumulate and reach large lengths. This drifting happens only occasionally, and does not significantly affect the parallelism using dynamic scheduling, but would force a static approach to use a large offset resulting in little parallelism.

[3]The text and results presented in this section are based on the work of Cor Meenderink from TU Delft and are presented here for completeness of this chapter

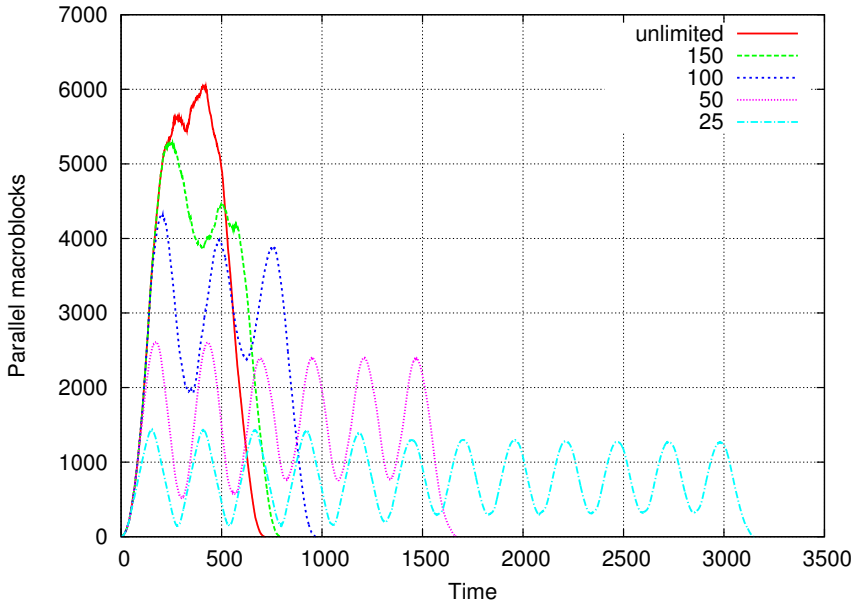


Figure 8.11: Available MB parallelism in FHD blue_sky for several limits of the number of frames in flight.

8.4.2 Effect of Limited Resources

The analysis above reveals that H.264/AVC exhibits significant amounts of MB parallelism. To exploit this type of parallelism on a CMP the decoding of MBs needs to be assigned to the available cores, i.e., MBs map to cores directly. However, even in future manycores the hardware resources (cores, memory, etc) will be limited. We now investigate the impact of resource limitations. Memory requirements are mainly related to the number of frames in flight. Thus limited memory is modeled by restricting the number of frames that can be in flight concurrently.

The experiment was performed for all four movies of the benchmark, for all three resolutions, and both types of motion estimation. The results are similar, thus only the results for the normal encoded blue_sky movie at FHD resolution is presented.

The impact of restricting the number of frames concurrently in flight is shown in Figure 8.11. The MB parallelism curves show large fluctuations, possibly resulting in under utilization of the available cores. These fluctuations are caused by the coarse grain of the limitation. At the end of decoding a frame, a small amount of parallelism is available. The decoding of a new frame, however, has to wait until the frame currently being processed is finished.

Even with a restriction in the number of frames in flight the Dynamic 3D-Wave algorithm is able to extract thousands of independent MBs. It is important to note that this analysis represent the theoretical limit of the available MB-level parallelism in H.264/AVC decoding. An important question that remains open is how to translate this huge amount of parallelism into performance gains taking into account factors like load-balancing, thread synchronization, impact of the sequential parts of the application,

task scheduling, dependency tracking etc.

8.5 Related Work

In this section we present an analysis of the most relevant works about parallel implementation of video decoders. Works are presented using the same classification of function-level and data-level decomposition presented above.

8.5.1 Function-level Parallelism

Lin et al. and Cantineau and Legat have reported functional parallelization of the H.263 and MPEG-2 encoders respectively [151, 39]. They used a multiprocessor (called TMS320C80) composed of one control processor and four DSPs. In the MPEG-2 case, the parallelization allows to encode CIF videos at 25 fps when running at a 50MHz. Oehring et al. have reported an analysis of the parallelization of the MPEG-2 decoder exploiting function level parallelism using a simulator for SMT processors with 4 threads [176].

Gulati and Campbell describe a system for encoding and decoding H.264 on a multiprocessor architecture using a task-level decomposition approach [89]. The multiprocessor includes eight DSPs and four control processors. The decoder is implemented using one control processor and 3 DSPs. The control processor performs the master control of the application and an initial parsing of the bitstream. Entropy decoding, inverse quantization, and IDCT are mapped to the first DSP, motion compensation and intra-prediction to the second, and finally the deblocking filter is mapped to the third DSP. Using this mapping, a pipeline for processing MBs is implemented. This system achieves real-time operation for low resolution video inputs, using the baseline profile. In a similar way, Schoffmann et al. propose a macroblock pipeline model for H.264 decoding [209]. Using an Intel Xeon 4-way SMP 2-SMT architecture their scheme achieves a 2X speed-up.

8.5.2 GOP-level Parallelism

Shen et al. have presented a parallelization of the MPEG-1 encoder for the Intel Paragon MIMD multiprocessor at the GOP level [218]. Their approach scale to a large number of processors but with a large latency. Bilas et al. has described a parallel implementation of the MPEG-2 decoder evaluating GOP and slice-level parallelism on a shared memory multiprocessor [31]. They compare the two levels in terms of speedup, load balancing, memory usage and synchronization overhead.

Rodriguez et al. proposed an encoder that combines GOP-level and slice-level parallelism for encoding real-time H.264 video using clusters of workstations [199]. Frame-level parallelism is exploited by assigning GOPs to nodes in a cluster, and after that, slices are assigned to processors in each cluster node. This methodology is well suited for the encoder, which has the freedom of selecting the GOP size, and can tolerate bigger latencies (in some cases).

8.5.3 Frame-level Parallelism

Chen et al. described an implementation of frame-level parallelism for the H.264/AVC encoder [43]. In this case a combination of frame-level and slice-level parallelism is proposed. First, they exploit frame level parallelism. They do not allow to use B-frames as references and use a static P-B-B-P-B-B sequence of frames. When the limit of frame-level parallelism has been reached they exploit slice-level parallelism. A 4.5X speed-up is achieved in a machine with 8 cores and using 9 slices per frame. Their approach, however, does not scale to more processors because the limits in frame-level parallelism and the bit-rate increase due to slices.

8.5.4 Slice-level Parallelism

There are some works on slice level parallelism for previous video Codecs. Lehtoranta et al. have described a parallelization of the H.263 encoder at the slice level for a multiprocessor system made of 4 DSP cores with one master processor and 3 computing processors [145]. Lee et al. have also reported an implementation of the MPEG-2 decoder exploiting slice-level parallelism for HDTV input videos [137]. Yadav et al. have reported a study on the parallelization of the MPEG-2 decoder for a multiprocessor SoC architecture. They studied slice-level, function-level and stream-level parallelism [263]. Jacobs et al. have analyzed the thread parallelism of MPEG-2, MPEG-4 and H.264 decoders. For MPEG-2 and MPEG-4 they have taken into account MB-level parallelism. For these cases the parallel version scales with the height of the frame. For the H.264 encoder they discarded the idea of implementing MB-level parallelism and went for slice-level [112].

Roitzsch proposed a scheme for exploiting slice-level parallelism in the H.264/AVC decoder by modifying the encoder. The main idea is to overcome the load balancing disadvantage by making an encoder that produces slices that are balanced in their decoding time. The main disadvantages of this approach are that it requires modifications to the encoder in order to exploit parallelism at the decoder, and the inherent loss of coding efficiency due to having a large number of slices [201].

8.5.5 Macroblock-level Parallelism

Previous Codecs

Some works have reported MB level parallelism at the encoder for the ME stage by performing ME in each processing element and replicating the search window. Akramullah et al.have reported a parallelization of the MPEG-2 encoder for cluster of workstations [5]. Taylor et al. have implemented an MPEG-1 encoder in a multiprocessor made of 2048 custom DSPs [242].

H.264/AVC

der Tol et al.proposed the exploitation of MB-level parallelism for optimizing the H.264/AVC decoder [69]. The analysis was made on a multiprocessor system consisting of 8 Trimedia processors with private L1 caches and a shared L2 cache. Also the paper proposed the grouping of MBs into data partitions and a special memory allocation technique for allocating those partitions in the cache. The paper also suggests the combination of

MB-level with frame-level parallelism to increase the availability of independent MBs. The use of frame-level parallelism is determined statically by the length of the motion vectors. Chen et al. evaluated a similar approach: a combination of MB parallelism and frame-level parallelism for the H.264 encoder on Pentium machines with SMT and CMP capabilities [45].

In the above mentioned works the exploitation of frame-level parallelism is limited to two consecutive frames and the identification of independent MBs is done statically by taking into account the limits of the motion vectors. Although this combination of MB and frame-level parallelism increases the amount of independent MBs, it requires that the encoder puts some limits on the length of the motion vectors in order to define statically which MBs of the next frame can start execution while decoding the current frame.

Zhao and Liang presented a combination of frame-level parallelism and macroblock-level parallelism for H.264 encoding [269]. In this approach multiple frames are processed in parallel similar to X264: a new frame is started when the search area in the reference frame has been fully encoded. But in this case macroblock-level parallelism is also exploited by assigning threads to different rows inside a frame. This scheme is a variation of what we call in this paper "Static 3D-Wave". It is static because it depends on a fixed value of the motion vectors in order to exploit frame-level parallelism. This works for the encoder who has the flexibility of selecting the search area for motion estimation, but in the case of the decoder the motion vectors can have big values and the static approach should take that into account. Another limitation of this approach is that all the MBs in the same row are processed by the same processor/thread. This can results in poor load balancing because the decoding time of MBs is not constant. A thread that has finished a fast MB has to wait for a slow MB in the previous row. Additionally this work is focused on the performance improvement on the encoder for slow resolutions with a small number of processors, discarding an analysis for a high number of cores as an impractical case and not taking onto account the peculiarities of the decoder.

Baik et al. developed a hybrid approach combining function-level parallelism and data-level parallelism for optimizing the H.264 decoder on the Cell B.E. architecture. They used three processing units (SPEs) for the motion compensation stage, another for the deblocking filter and the other kernels (entropy decoding, IDCT, IQ and intra-prediction) were assigned to the master processor (PPE) [25].

Seitner et al. presented a comparison of different data-level parallelization approaches based on slice-level, and macroblock-level parallelism. For the latter they analyzed three different configurations such as row processing, column processing and diagonal (2D-Wave) order. They performed a high-level simulation approach for comparing the speedup and waiting time of the different approaches.

Nishihara et al. presented a version of the row-order MB-level parallelization combined with function-level parallelism. An additional runtime partitioning method is proposed to improve load balancing [173]. Sihn et al. applied a similar technique which a combination of function-level and MB-level parallelism is optimized for better load balancing and a reduction of memory accesses [223]

Baker et al. implemented the row variant of intra-frame MB-level parallelism described by Seitner et al. on the Cell B.E. architecture. They mapped entropy decoding onto the PPE processor and macroblock reconstruction onto the SPE processors. Each SPE process one row of MBs. A row approach is well suited for this architecture because

it uses local memories with explicit transfers, and the row order of processing allow to overlap data transfers and computation [26]. Chi et al. implemented a variant of the MB-level row approach and compared it with a centralized task-pool implementation. Due to the locality effect mentioned above the former approach resulted in higher performance and scalability [48]. Cho et al. implemented a similar approach on the cell processor but included a frame-level parallelization of the entropy decoder. In their implementation the master processor (PPE) runs a parser thread and two entropy decoding threads. The SPEs perform the MB reconstruction tasks using the row-order MB-level approach.

Chong et al. proposed another technique for exploiting MB level parallelism in the H.264 decoder by adding a prepass stage [51]. In this stage the time to decode a MB is estimated heuristically using some parts of the compressed information of that MB. Using the information from the preparsing pass a dynamic schedule of the MBs of the frame is calculated. MBs are assigned to processors dynamically according to this schedule. By using this scheme a speedup of 3.5X has been reported on a simulated system with 6 processors for small resolution input videos. Although they present a dynamic scheduling algorithm it is not able to discover all the MB level parallelism that is available in a frame. The preparsing scheme presented can be beneficial for the Dynamic 3D-Wave algorithm.

8.6 Summary

In this chapter we have presented an analysis of different parallelization techniques that can be applied to video decoding applications paying special attention to the H.264/AVC decoder. Techniques like function-level parallelization and several forms of data-level parallelization were analyzed in terms of their scalability. MB-level parallelism was examined in detail because it has the potential of scaling to future manycore architectures. The combination of spatial (intra-frame) and temporal (inter-frame) MB-level parallelism demonstrated to be the most scalable approach. Static and a dynamic variations of this technique were studied in detail. In the static case, temporal MB-level parallelism is based on a predefined value of the motion vectors. Because the decoder can not control the motion vector length it has to assume a worst case resulting in loss of parallelism. The dynamic variant is proposed as a way to increase the amount of independent MBs. In this approach each MB is assigned to a processor when its intra- and inter-dependencies have been processed. A theoretical analysis with real videos show that for FHD resolution there is a parallelism in the range of 4000 to 8000. This requires more than 200 frames in flight. When the number of frames in flight is limited, the number of parallel MBs exhibits fluctuations but still a high degree of parallelism.

A special section was devoted to a review of the most significant works in the field of video decoder parallelization. The abundant works were grouped by the type of parallelization that they use: function-level, some for of data-level of combination of several of them.

Dynamic 3D-Wave has been proposed as a scalable approach for parallel H.264/AVC decoding on manycore architectures. The presented analysis was done without taking into account implementation issues, like scheduling, synchronization, data transfer, memory requirements etc. The impact of these factors and the requirements in both the application and the architecture are going to be analyzed in the next chapters.

9 Scalability of Macroblock-level Parallelism

In this chapter we investigate the scalability of spatial (intra-frame) MacroBlock-level parallelization of the H.264 decoder for High Definition (HD) applications. As it was mentioned in the previous chapter, this technique is able to scale to multicore architectures with tens of cores depending on the resolution. The objective of the previous analysis was to provide upper-bounds to the parallelization under idealistic conditions. Some of these assumptions were: not taking into account variable execution time in MB decoding tasks, assuming zero thread synchronization time and do not take into account the effect of data locality. In this chapter a more detailed model is presented and it is compared with the results of a real implementation.

This chapter includes two parts. First, we present a formal model for predicting the maximum performance of the 2D-Wave algorithm taking into account variable processing time of tasks and thread synchronization overhead. And, second, we present the results of an implementation on a real multiprocessor architecture. This includes a comparison of different scheduling strategies and a profiling analysis for identifying the performance bottlenecks. CABAC entropy decoding and thread synchronization are identified as the main factors that limits the performance of the 2D-Wave decoder.

9.1 Theoretical Analysis

In this section we present a theoretical model that take into account factors that affect the parallel scalability like the variable processing time of the inner kernels and the overhead of thread synchronization.

9.1.1 Formal Model

We can represent the processing of MBs in H.264/AVC decoding as a DAG (Directed Acyclic Graph). Each node in the DAG represents the decoding of one MB by one processor. The decoding of each MB consists of a sequential ordering of kernels applied to some input data. Edges in the graph represent the data dependencies between MBs. Figure 9.1 shows the DAG for a 5×5 MBs sample frame. Each frame in a video sequence can be represented with a finite DAG. The first MB in the frame is the *source node* which has no incoming edges and the last MB in the frame is the *sink node* which has not outgoing edges. We define the *depth* as the length of the longest path from the source node to the sink node. For a finite DAG G representing a frame F we define the computational work T_s as the number of nodes in G, and T_∞ as the depth of G. Although the structure of the dependencies is known the actual shape of the DAG is input dependent and cannot be known before the processing of all nodes.

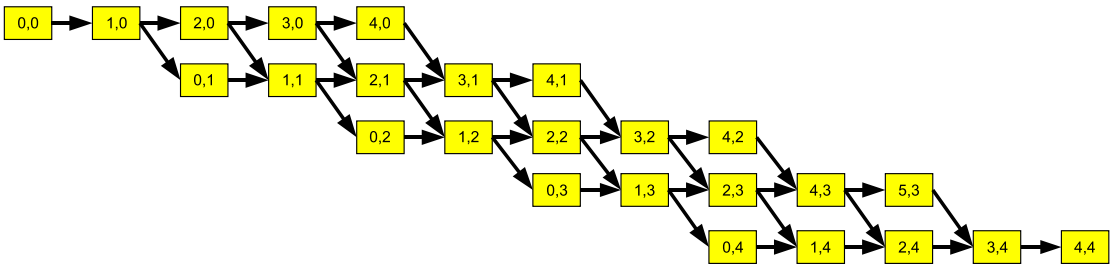


Figure 9.1: Directed Acyclic Graph (DAG) of macroblocks

Video Resolution	Pixel Resolution	MB Resolution	T_s	T_∞	Max. speedup	Max. processors
Ultra High (UHD)	7680x4320	480x320	129600	1018	127.31	240
Quad Full High (QFHD)	3840x2160	240x160	32400	508	63.78	120
Full High (FHD)	1920x1080	120x80	8160	254	32.13	60
High (HD)	1280x720	80x45	3600	168	21.43	40
Standard (SD)	720x576	45x36	1620	115	14.09	23
CIF	352x288	22x18	396	56	7.07	12
QCIF	176x144	11x9	99	27	3.67	6

Table 9.1: Theoretical maximum speedup for different video resolutions

Assuming that the time to process each node in the DAG is constant and that there is not overhead for thread synchronization then we can estimate the theoretical maximum speedup. Let mb_width and mb_height be the width and height of the frame in macroblocks respectively. Then, $T_s = mb\_width * mb\_height$ and $T_\infty = mb\_width + (mb\_height - 1) * 2)$. The maximum speedup (MSU) is defined as:

$$MSU = \frac{mb\_width * mb\_height}{mb\_width + (mb\_height - 1) * 2} \tag{9.1}$$

Taken that into account, we can calculate the maximum number of processors (MP) as:

$$MP = round\left(\frac{mb\_width + 1}{2}\right) \tag{9.2}$$

In Table 9.1, these values are shown for different video resolutions. For FHD resolution the theoretical maximum speedup is 32.13 when using 60 processors.

9.1.2 Abstract Trace-driven Simulation

The theoretical maximum speedup is based on the assumption that MB processing time is constant and there is not thread synchronization overhead. Both assumptions are not true in real applications. On the one hand, although the same set of filters are applied to each MB, the processing time is input dependent because the exact operations that are applied to the image samples depend on conditions of those samples. On the other hand, thread synchronization overhead is not negligible. For processing a MB a system for keeping track of data dependencies is necessary and a scheduling process for assigning tasks to processors have to be done. Those steps require the synchronization of parallel threads.

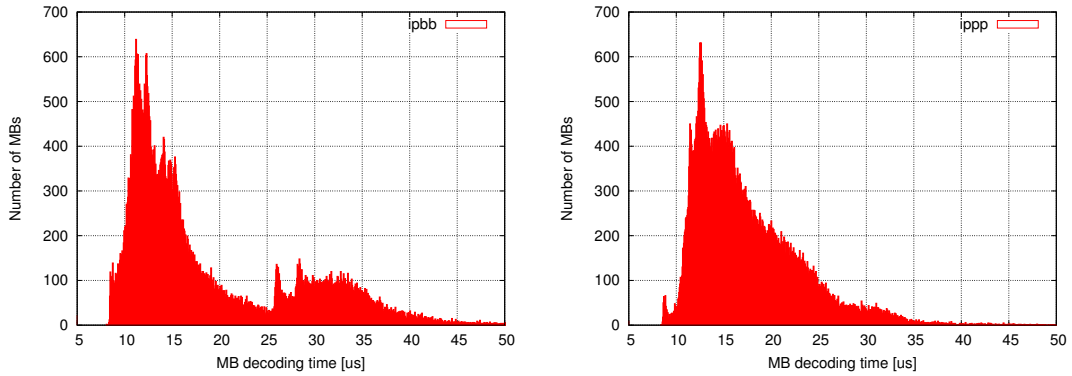
In order to analyze the effects of those conditions we have build an abstract MB trace-driven simulator called *frame_sim*. It creates the DAG for each frame and then calculates the Task Processing Time (TPT) of every node as:

$$TPT(n) = w_n + s_n + MAX\left(TFT(pr_n)\right) \tag{9.3}$$

Where, w_n is the time required to process the task, s_n is the time required for thread synchronization; and $MAX(TFT(pr_n)$ is the maximum task finish time (TFT) of the immediate predecessors tasks of that task. When the DAG has been fully processed we take the data from the *end node* and its finish time represents the best time that we can achieve from the parallel execution of that DAG. Because this is input dependent we have analyzed the DAGs for different frames and different input videos at FHD. The simulator take the execution time of each task from an execution trace that results from the instrumentation of the H.264/AVC decoder.

9.1.3 Effects of Variable Decoding Time

In order to show the level of variability in decoding time, we have constructed an histogram of the time required to process MBs on video with two different coding options: one with P- and B-frames and the other one with only P-frames. Figure 9.2 show the results. Time is collected on an Itanium-II processor machine (described in detail in next chapter). These results show not only variation between MBs on the same video but also between video sequences with different coding options. MB decoding time depends on conditions of the inputs like: the type and position of the MB, type of motion compensation interpolation (integer, fractional), type of deblocking filter, number of non-zero coefficients etc. These values cannot be easily predicted or known in advance.



(a) IPBB sequence

(b) IPPP sequence

Figure 9.2: Histograms of MB processing time on Itanium-II architecture, time in ns.

Using *frame_sim* and taking into account the variation in execution time we create the best schedule for each frame that allows us to compute the maximum speedup. Table 9.2 shows the speedup of the parallel execution for different input videos. It includes the maximum theoretical speedup and the maximum speedup taking into account the variable processing time. In average for all the input videos the speedup is reduced a 33% compared to the theoretical maximum.

Input Video	speedup const. time	speedup var. time	slow-down
1088p25_Blue_sky	32.13	19.22	0.40
1088p25_Pedestrian_area	32.13	21.92	0.31
1088p25_Riverbed	32.13	24.01	0.25
1088p25_Rush_hour	32.13	22.22	0.30

Table 9.2: Maximum speedup taking into account variable MB decoding time.

The values presented in Table 9.2 are average per frame. Actual performance changes from frame to frame due to the differences in input content and type of MBs. The best schedule calculated before can be seen in Figure 9.3 for the four FHD input videos.

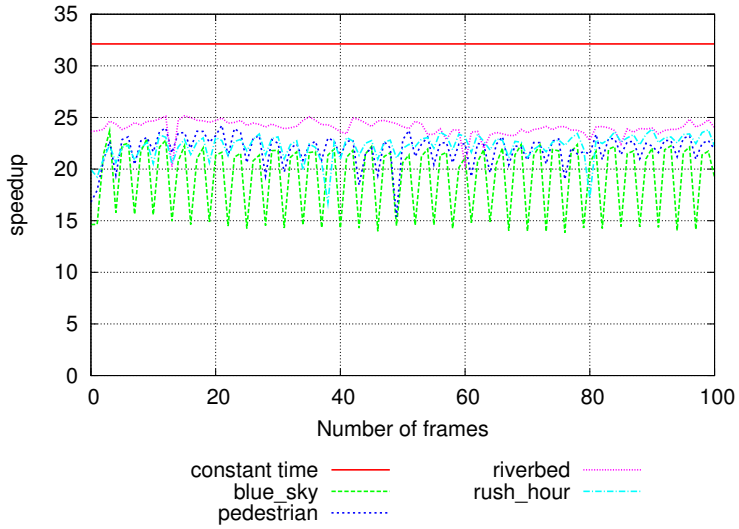


Figure 9.3: Maximum speedup by frame taking to account variable processing time

9.1.4 Effects of Thread Synchronization Overhead

We have modeled the synchronization overhead as an extra time for MB decoding. The base value for the overhead is the average processing time of each MB in a frame. Figure 9.4 shows the average speedup for different FHD video sequences and using the maximum number of processors for this resolution. A zero value represents the maximum speedup taking into account the variable processing time but no synchronization time. As the value of overhead increases the speedup decreases correspondingly. For example, consider the *1088p25_blue_sky* video sequence: with zero synchronization overhead the maximum speedup is 19.23. Adding a synchronization overhead of 1 the speedup reduces to 11.93 (38%).

By using these data a system designer can decide when thread synchronization optimizations are useful in terms of the cost to design and implement them compared to the benefit in speedup. Although synchronization overhead values bigger than the processing time may seem unreasonable we have found in our experiments values up to 12 times the average MB decoding time as we will show later.

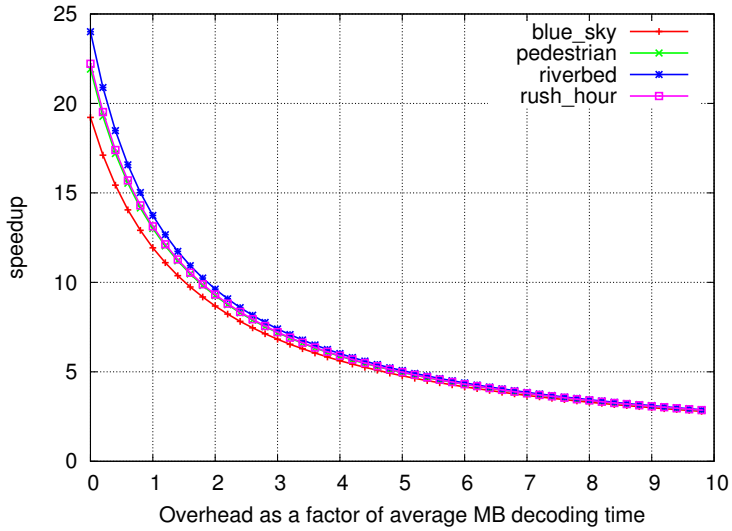


Figure 9.4: Synchronization overhead vs speedup on Itanium-II Architecture.

9.2 Performance Analysis on a Parallel Architecture

In order to validate the results from the theoretical model and the abstract graph simulations we implemented the 2D-Wave algorithm on a real multiprocessor machine.

9.2.1 Implementation Methodology

Our implementation is based on a dynamic task model using task pools. In this model, a set of threads is activated when a parallel region is encountered. In our case, a parallel region is the decoding of all MBs in a frame. Each parallel region is controlled by a frame manager, which consist of a thread pool, a task queue, a dependence table and a control thread, as shown in Figure 9.5.

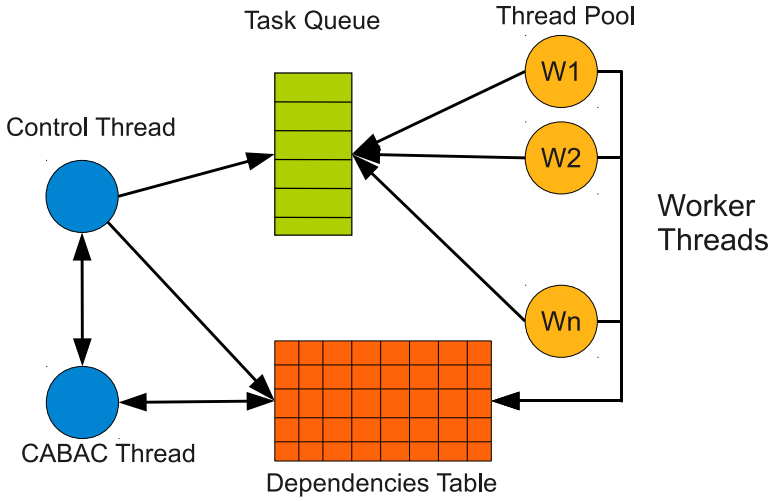


Figure 9.5: Dynamic task model diagram

The thread pool consists of a group of worker threads that wait for work on the task queue[126]. The generation of work on the task queue is dynamic and asynchronous. The dependencies of each MB are expressed in a dependence table. When all the dependencies for a MB are resolved a new task is inserted on the task queue. Any thread from the thread pool can take a task and process it. When a thread finish the processing of a MB it updates the table of dependencies and if it founds that another MB has resolved its dependencies it can insert a new task into the task queue. Finally, the control thread is responsible for handling all the initialization and finalization tasks that are not parallelizable. As a further optimization step a tail-submit method has been considered in which each worker thread can process MBs directly without passing through the task queue.

The dynamic task model was implemented using POSIX threads (P-threads) [106]. Synchronization between threads and the access to the task queue were implemented using blocking synchronization with POSIX real-time semaphores [105]. Atomicity of the read and write operations is guaranteed using P-threads mutexes. Both synchronization APIs are blocking and requires the operating system intervention for the activation of threads. The access to the table of dependencies was implemented with atomic instructions like *dec_and_fetch* [241].

9.2.2 Evaluation Platform

For these experiments we have used a modified version of the FFmpeg H.264/AVC with support for 2D-Wave parallelization. Video inputs are taken from HD-VideoBench using the coding options defined in the benchmark [8].

The application was tested on a SGI Altix which is a distributed shared memory (DSM) machine with a cc-NUMA architecture. The basic building block is this DSM system is the blade. Each blade has two dual-core Intel Itanium processors, 8GB of RAM and an interconnection module. The interconnection of blades is done using a high performance interconnect fabric called NUMAlink-4 capable of 6.4 GB/s peak bandwidth through two 3.2 GB/s unidirectional links (see figure 9.6). The complete machine has 32 nodes with 2 dual-core processors per node for a total of 128 cores and a total of 1TB of RAM.

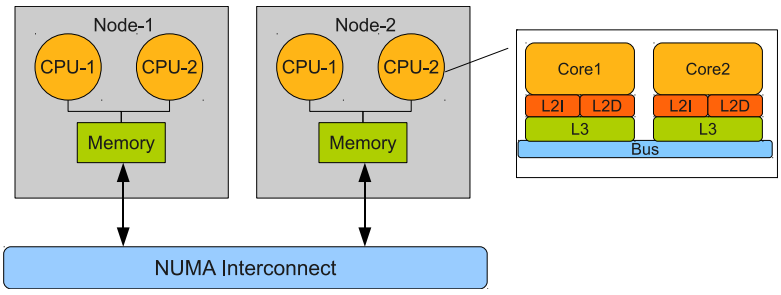


Figure 9.6: Architecture of the cc-NUMA multiprocessor

Each processor in the system is a Dual-Core Intel Itanium2 processor running at 1.6 GHz [38]. This processors has a 6-wide, 8-stage deep, in order pipeline. The resources consist of six integer units, six multimedia units, two load and two store units, three

branch units, two extended-precision floating point units, and one additional single-precision floating point unit per core. The hardware employs dynamic prefetch, branch prediction, a register scoreboard, and non-blocking caches. All the three levels of cache are located on-die. The L3 cache is accessed at core speed, providing up to 8.53 GB/s of data bandwidth. The Front Side Bus (FSB) runs at a frequency of 533 MHz. Main parameters of the processor are listed in Table 9.3

Configuration	SGI Altix
ISA	Itanium 64-bit
SIMD extensions	MMX, SSE, SSE2
Processor	Intel Itanium 2 9030
Technology	90nm
Clock frequency	1.6 GHz
Power	104 W
Level 1 I-cache	16 KB / core
Level 1 D-cache	16 KB / core
Level 2 I-cache	1 MB / core
Level 2 D-cache	256 KB / core
Level 3 cache	8 MB

Table 9.3: Experimentation platform

The compiler used was GCC 4.1.0 and the operating system was Suse Linux with kernel version 2.6.16.27. Profiling information was taken using the Paraver tool with traces generated using the source-to-source Mercurium compiler and the Mintaka trace generator [185].

The application was executed on the SMP machine using "cpusets" and "dplace" options. Cpusets are objects in the Linux kernel that enable to partition the multiprocessor machine by creating separated execution areas. By using them the application has exclusive use of all the processors. With the dplace tool, memory allocation is done taking into account the cc-NUMA architecture and data is allocated in a NUMA friendly way.

9.2.3 Scheduling Strategies

One of the main factors that affects the scalability of the 2D-Wave parallelization is the allocation (or scheduling) of MBs to processors. We have evaluated three different scheduling algorithms: static scheduling, dynamic scheduling and dynamic scheduling with tail submit optimization.

The application was executed for the three scheduling algorithms with 1 to 32 processors. Although the machine has 128 processors, executions with more than 32 processors were not carried out because the speedup limit is always reached before that point. Speed-up is calculated against the original sequential version. The speedup reported (unless the contrary is explicitly said) is for the parallel part which corresponds to the decoding of all MBs in a frame without considering entropy decoding.

In Figure 9.7 the average speedup for the different scheduling approaches is presented. In the next sections each one is going to be discussed in detail.

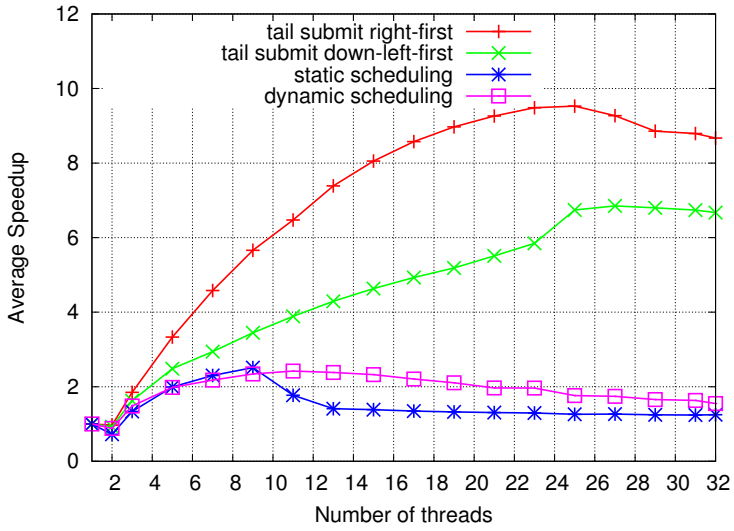


Figure 9.7: Speedup of Macroblock decoding using different scheduling approaches

9.2.4 Static Scheduling

Static scheduling means that the decoding order of MBs is fixed and a master thread is responsible for sending MBs to the decoder threads. The predefined order is a zigzag scan order which can lead to an optimal schedule if MB processing time is constant. When the dependencies of an MB are not ready the master thread waits for them. Figure 9.7 shows the speedup of static scheduling. The maximum speedup reached is 2.51 when using 8 processors (efficiency of 31%). The low scalability is due to the fact that MB processing time is variable, and static scheduling results in load unbalance: most of the time the master thread is waiting for other threads to finish.

9.2.5 Dynamic Scheduling

In this scheme worker threads take MBs from the task queue, process them, update the dependence table and, if that is the case, submit new MBs to the task queue. Production and consumption of MBs is made through the centralized task queue. Figure 9.7 shows the speedup for the dynamic scheduling. A maximum speedup of 2.42 is found when 10 processors are used (efficiency of 24%). This is lower than the maximum speedup for the static scheduling. Although the dynamic scheduling is able to discover more parallelism than static scheduling, the overhead for submitting MBs to (and getting MBs from) the task queue is so big that it jeopardizes the parallelization gains. Most of this overhead comes from the intervention of the OS in the scheduling process and for contention in the access to the queue.

In order to analyze the performance of worker threads we divided the execution of each one into six phases, as follows:

- *get_mb*: Take one element from the task queue.

- *copy_mb*: Copy of entropy decoded parameters to the local thread structures.

- *decode_mb*: Actual work of MB decoding.
- *update_mb*: Update the table of MB dependencies.
- *ready_mb*: Analysis of new ready to process MB.
- *submit_mb*: Put one element into the task queue.

Table 9.4 shows the execution time of the different phases. It can be noted that the execution time of MB decoding (*decodemb*) and the copying of entropy decoding data (*copy_mb*) increase with the number of processors. This is mainly due to the fact that the dynamic scheduling algorithm does not consider data locality when assigning tasks to processors. When a processor takes a MB which has its data dependencies in a remote node, then all the memory accesses should cross the NUMA interconnection network. Other phases that exhibit a major increase in execution time are: *get_mb* and *submit_mb*. This reveals a contention problem because in dynamic scheduling all the worker threads get MBs from (and submit MBs to) a centralized task queue creating an important pressure on it. The last column of the table shows the ratio of actual computation and overhead. The overhead increases significantly when the number of processors goes beyond 8. Those results are consistent with the model presented in Figure 9.4. From this, we can conclude that the synchronization overhead for accessing the centralized task queue becomes the bottleneck.

Threads	decode_mb	copy_mb	get_mb	update_mb	ready_mb	submit_mb	overhead-ratio
1	22.65	1.62	4.89	1.01	2.38	5.03	0.67
4	33.09	2.95	9.71	1.36	2.86	12.44	1.30
8	41.88	3.90	16.67	1.61	3.02	20.84	2.05
16	61.78	5.94	55.95	2.25	3.55	80.28	6.57
24	58.08	5.15	105.03	2.09	3.49	120.37	10.49
32	78.75	7.25	209.37	2.70	4.36	201.01	18.88

Table 9.4: Average execution time of different phases for worker threads with dynamic scheduling, time in us.

9.2.6 Dynamic Scheduling with Tail-submit

As a way to reduce the contention on the task queue, the dynamic scheduling approach was enhanced with a tail submit optimization [95]. With tail submit when a thread founds a ready to process MB it can process that MB directly without any further synchronization. If more than one MB is discovered, one is submitted to the task queue and the other one is processed directly. There are two ordering options for doing the tail submit process: execute directly the right neighbor of the current MB and submit the other, or execute directly the down-left neighbor and submit the other. Figure 9.8 shows an example of a QCIF image indicating the ready to process MBs when some MBs have finished their execution.

Figure 9.7 shows the speedup of tail-submit implementations. The down-left-first version achieves a maximum speedup of 6.85 with 26 processors (efficiency of 26%). The right-first version achieves a maximum speedup of is 9.53 with 24 processors (efficiency of 39.7%). The better scalability of the right-first order is due to the fact the it exploits

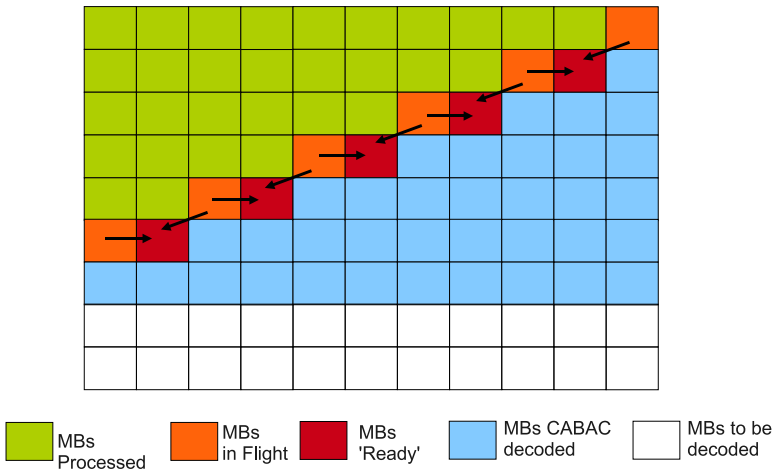


Figure 9.8: Ready to process MBs with 2D-Wave parallelization

the data locality between MBs. Data from the left block is required by the deblocking filter and by using the right-first order the values of the previous MB remain in the cache.

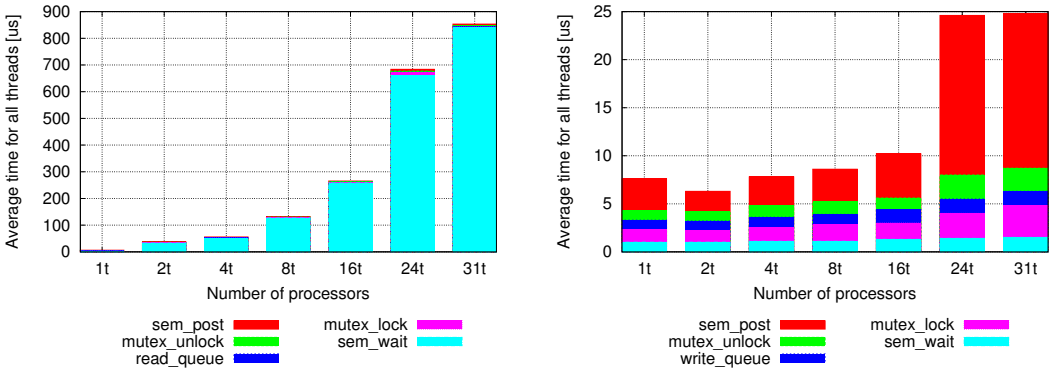
Table 9.5 shows the profiling results for tail submit version with right-first order. In this case, MB decoding time remains almost constant with the number of threads due to the exploitation of the data locality between neighbor MBs. Another effect of the tail submit optimization is the reduction in the time spent in *submit_mb*. This time still increases with the number processors but the absolute value is less than the dynamic scheduling version. With tail submit there is less contention because there are less submissions to the task queue as shown in the last column of the table. The most significant contributor to the execution time is *get_mb* indicating a lack of parallel MBs, meaning that the scalability limit of the tail submit version has been reached.

Threads	decode_mb	copy_mb	get_mb	update_dep	ready_mb	submit_mb	overhead ratio	% of tail submit
1t	21.7	1.5	6.1	1.0	1.0	7.5	0.17	90.8
4t	24.2	1.9	55.9	1.1	1.1	7.8	0.22	79.8
8t	24.9	2.1	132.4	1.3	1.1	8.6	0.30	75.2
16t	27.5	2.4	265.3	1.6	1.1	10.1	0.68	58.5
24t	30.6	2.9	683.7	1.9	1.2	24.6	1.00	51.4
32t	30.1	2.8	853.1	2.1	1.1	24.8	1.85	48.4

Table 9.5: Average execution time of different phases for worker threads with tail submit optimization, time in us.

In order to understand the sources of thread synchronization overhead we have performed a more detailed profiling analysis of the synchronization phases. The main synchronization point is in the task queue for getting and submitting tasks. Access to the queue is controlled by two semaphores: one for available elements and the other for empty slots. The operation of getting (submitting) and element from (to) requires a critical section that is controlled by a mutex. Inside the critical section there is the read (or write) of tasks from (into) the task queue.

Figure 9.9 shows the average execution time of the *get_mb* and *submit_mb* functions. The most important part for the first one is related to the semaphore at the beginning (*sem_wait*) which represents the waiting time for elements in the task queue. As the number of processors increases this value becomes bigger. In the case of a large number of processors the big waiting time indicates that there are no more MBs to process and that the parallelism of the application has reached its limit. In the case of *submit_mb* it can be noted an important execution time increase of the output semaphore release operation. This operation signals the availability of a new element in the task queue; when the number of thread increases the number of waiters in the task queue increases as well. The time required to signal the availability of one MB depends on the number of waiters, which is not a desirable feature. This is an implementation issue of the POSIX *sem_post* API.



(a) get mb

(b) submit mb

Figure 9.9: Distribution of execution time for the get_mb and submit_mb functions in dynamic scheduling with tail submit

9.2.7 Impact of the Sequential Part of the Application

In order to allow a parallel decode of MBs CABAC entropy decoding is decoupled from the MB decoding loop. The decoupling is done by using an intermediate buffer in which the CABAC decoder stores the decoded information for every MB. After finishing the CABAC decoding of a frame the decoder threads start to decode MBs in parallel. Because CABAC decoding cannot be parallelized at MB-level it should be executed sequentially in one processor. According to the Amdahl's law, it can become the limiting factor.

Figure 9.10 shows the execution time of the application including CABAC time. The execution time of MB decoding (*hl_decode_mb*) reduces with the number of processors as a result of the parallel execution. But, the execution time associated with CABAC (*decode_cabac*) augments with the number of processors. This is a side effect of the cc-NUMA memory model. When a new frame is being processed the CABAC decoder should overwrite the values in the intermediate buffer and this generates cache invalidations that go out of the chip and cross the interconnection network to reach the local caches that have these values.

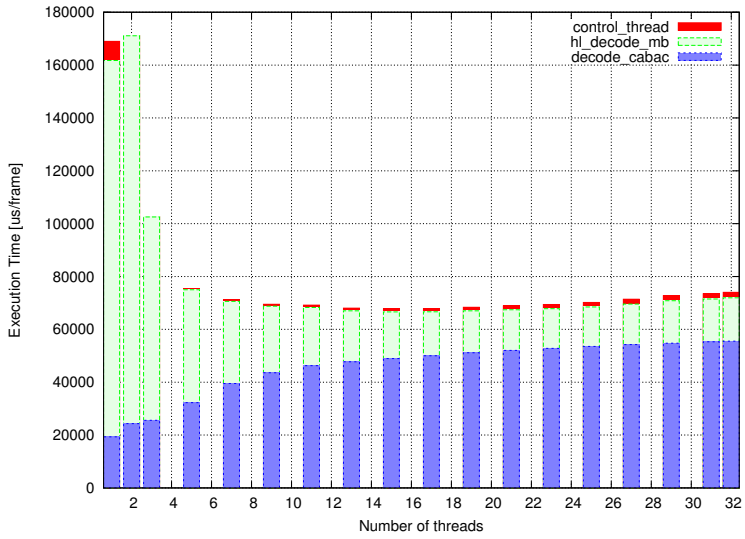


Figure 9.10: Average execution time per frame including CABAC entropy decoding.

As a consequence, in order to translate the 2D-Wave algorithm into complete application speedup it is necessary to accelerate the entropy decoding stage. We will analyze this issue in detail in the next chapter.

9.3 Summary

In this chapter we have investigated the scalability of the 2D-Wave parallelization of the H.264/AVC decoder.

A formal model and an abstract trace driven simulation were used to estimate the impact of variable decoding time and thread synchronization overhead on the maximum performance. Variability in processing time of tasks reduces the maximum speedup and demand the use of dynamic load balancing techniques. The analysis of the thread synchronization allows to estimate the impact of optimizations in the synchronization infrastructure.

A performance analysis of the implementation of the 2D-Wave parallelization on the cc-NUMA machine was presented. Three different scheduling techniques were evaluated. The study shows that the best scheduling strategy is the combination of dynamic scheduling with tail submit optimization. Dynamic scheduling deals with the unbalance that results from variable decoding time, but it suffers from contention on a centralized task queue. By using tail submit optimization the synchronization overhead is reduced and, at the same time, data locality can be exploited reducing the external memory pressure. Although tail-submit results in an important reduction in the synchronization overhead, this remains a main limitation for the MB-level parallelization.

Additionally it was shown that using a single processor for executing the entropy decoder does not provide the performance required to obtain significant benefits for the complete application. Some kind of acceleration is required to make the 2D-Wave parallelization useful.

10 Scalability of Heterogeneous Multicore Architectures for Parallel Video Decoding

In a previous chapter we have presented an analysis of the parallel scalability of H.264/AVC and we have shown that a combination of spatial and temporal macroblock-level parallelism can scale to a large number of processing elements. This was a theoretical study that did not consider practical aspects related to the implementation on actual parallel architectures. In this chapter, we consider the implementation of a highly parallel H.264/AVC decoder that is able to scale for a manycore architecture with 100 cores (or more).

In order to translate task-level parallelism into performance gains both the video decoder and the architecture have been optimized. The video decoder was modified for exploiting coarse-grain frame-level parallelism in the entropy decoding kernel which has been considered the main bottleneck. Second, a heterogeneous combination of cores is evaluated for executing different type of tasks. Third, an evaluation of the performance of accelerated synchronization operations is presented. Finally, an evaluation of the memory requirements of the whole system has been carried out. Experiments conducted using a trace-driven simulation methodology shows that the evaluated system exhibits a good parallel scalability up to 68 cores. At this point the parallel video decoder is able to decode more than 200 HD frames per second using simple low power processors.

10.1 Scalable H.264/AVC Parallelization

We use the dynamic 3D-Wave algorithm (presented in Chapter 8), that exploits intra- and inter-frame MB-level parallelism. It is dynamic because the assignment of MacroBlocks (MBs) to processors is performed at run-time based on data dependencies. This algorithm is able to extract thousands of independent MBs depending on the input content and the number of frames in flight [160].

Intra-frame dependencies are known in compile time and are expressed according to the position of the macroblock in the frame. That includes, 2, 1 or zero dependencies, depending on the availability of neighbor macroblocks. Inter-frame dependencies are discovered at run-time in the motion compensation stage. Because a macroblock can have multiple sub-blocks, each one with its own motion vectors, this can lead to multiple (up to 16) inter-dependencies per macroblock which can result in a big runtime overhead. In order to reduce the number of dependencies we combined the motion vectors of all the sub-blocks into only one motion vector that points to a reference area that include all the reference areas of the sub-blocks. If the reference area of a macroblock is not ready, the thread waits until that area in decoded.

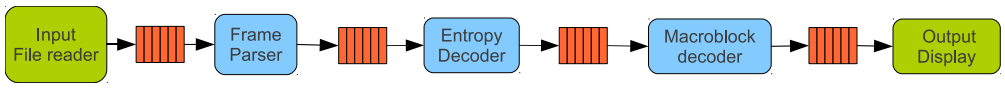


Figure 10.1: H264 decoder with CABAC decoupling

10.2 Solving the Scalability Bottlenecks

In order to translate the task-level parallelism discovered by the dynamic 3D-Wave algorithm into performance gains it is necessary to address two scalability issues: the performance of the entropy decoding stage and the effect of thread synchronization.

10.2.1 Parallelism in the Entropy Decoding Stage

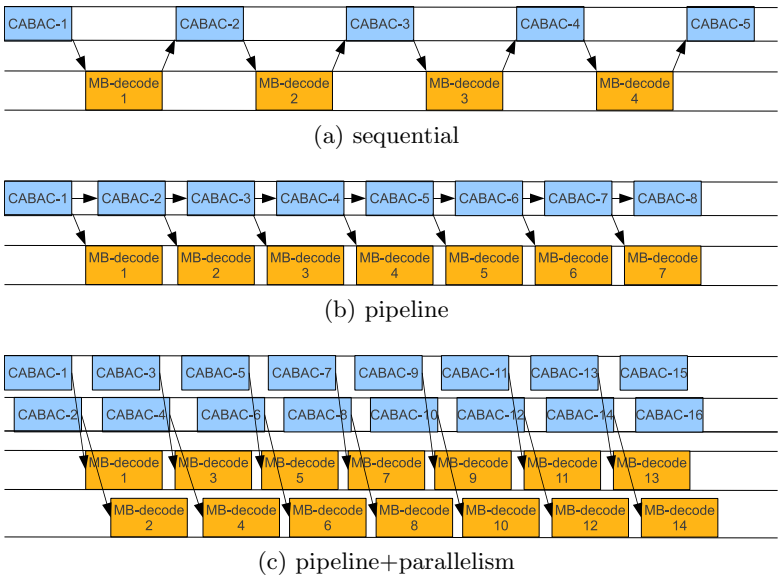
Due to data dependencies entropy decoding has to be performed sequentially for all the macroblocks inside a frame. The main problem for the parallel 3D-Wave decoder is that performance of the whole application is dominated by the performance of the entropy decoding stage, according to Amdahl's law. But, it is possible to parallelize entropy decoding at the frame-level because the context tables that create data dependencies are re-started every frame [156]. The only dependency that exists is related to DIRECT MBs in B-frames [92]. In this type of MBs there is not data transmitted and the motion vector is taken from the co-located MB in a subsequent reference frame. If motion vector prediction is performed at the entropy decoding stage, which is usually the case, then entropy decoding of the current frame should be at least one MB in advance of entropy decoding of the next frame.

In order to parallelize the CABAC entropy decoder it has to be decoupled from the MB decoding kernels. After decoupling, the video decoder can be seen as a macro-pipeline with a front-end and a back-end. In the front-end, there are two stages: a parsing stage that reads the compressed bitstream and parses the frame headers and a CABAC decoding stage. In the back-end there are two stages: one is MB decoding and the other is frame display[1]. The resulting structure of the application is shown in Figure 10.1. This macro-pipeline combines coarse-grain parallelization for the entropy decoder and fine-grain parallelization for MB decoding.

The front-end can use multiple processors to perform entropy decoding of multiple frames in parallel. It communicates with the back-end using a frame-buffer. The size of this buffer (in frames) defines the maximum number of frames that can be "in-flight" at any point in time. Having more frames in-flight improves the utilization of CABAC processors at the cost of more memory. Each CABAC frame requires 10.5MB for FHD resolution.

Figure 10.2 shows a diagram with the general concept of combining multiple levels of parallelism. Figure 10.2a shows the sequential approach in which CABAC decoding of the next frame waits for MB decoding of the current frame to finish. Figure 10.2b illustrates the case when pipelining parallelism is exploited. In this case, CABAC entropy decoding of the next frame can start just at the end of CABAC decoding of the current frame. Finally, Figure 10.2c shows the combination of pipelining with data-level par-

[1] In our case uncompressed frames are not displayed on the screen and we have disabled the writing to an output file, then basically the display stage is only in charge managing the frame buffer



(a) sequential

(b) pipeline

(c) pipeline+parallelism

Figure 10.2: CABAC: multiple frames in flight and frame-level parallelism

allelism both in CABAC decoding and MB decoding. Multiple frames can be entropy decoded and MB decoded in parallel.

A side benefit of this division is that it allows specialization. That means the use of different types of processors for different stages of the pipeline.

10.2.2 Thread Synchronization

Our implementation of the dynamic 3D-Wave algorithm uses a task-based programming model in which a master thread creates and instantiates tasks that are executed by worker threads. Worker threads access a task pool to obtain a new task, make explicit check operations for the input dependencies, process the task, and finally, send notifications to the master thread or to other worker threads. Thread synchronization is required for accessing the task pool and for exchanging signals between dependent tasks.

Worker threads use semaphores for their synchronization operations. This model requires low latency operations compared to the average task processing time and that the synchronization latency does not increase significantly when the number of cores increases.

We evaluate a solution based on on-chip hardware semaphores whit a semantics similar to UNIX real-time semaphores [105] but with reduced latency and overhead. Hardware semaphores are exposed to the software as an ISA extension of the worker processors. The new instructions are described in Table 10.1.

sem_init assigns a hardware semaphore to a logical identifier and sets a initial value. *sem_wait* locks a semaphore and its behavior is similar to a blocking load. *sem_post* unlocks a semaphore in a way similar to a non-blocking store. And *sem_destroy* removes the assignment of the hardware semaphore freeing its resources.

Although hardware semaphores have been described in our context for optimizing a parallel video decoder, they can be used to optimize other parallel applications with

Instruction	Operation	Parameters
sem_init	Initialize a semaphore	sem. ID, init. val.
sem_wait	lock a semaphore	sem. ID
sem_post	unlock a semaphore	sem. ID
sem_destroy	Destroy a semaphore	sem. ID

Table 10.1: Instructions for task synchronization

fine-grain tasks that require low latency synchronization.

10.3 Experimental Methodology

We use a fast trace-driven simulation methodology that allows to simulate systems with large numbers of cores.

10.3.1 H.264/AVC Decoder

For these experiments we used the parallel H.264/AVC decoder that is available from HD-VideoBench [8]. This code is based on the FFmpeg libavcodec library, and includes a parallel H.264/AVC decoder.

The benchmark includes four input sequences at different resolutions but due to space reasons only results for the *1088p25_pedestrian_area* test video are presented. Videos were encoded using the X264 encoder with the following options: 100 frames, P-frames, 1 reference frame, hexagonal motion estimation algorithm (hex) with maximum search range 24, one slice per frame. We restricted the encoder to produce only P-frames to allow parallel CABAC decoding without dependencies. One reference frame was enforced to simplify the tracking of dependencies for the 3D-Wave algorithm. These two simplifications made a significant reduction in implementation complexity without losing the general applicability of the results.

10.3.2 Instrumentation and Trace generation

The parallel H.264/AVC decoder was executed on a real parallel platform on which execution traces were extracted. The execution platform is the SGI Altix machine based on Itanium-II processors described in Chapter 9. Each processor in the system is a Dual-Core Intel Itanium2 processor running at 1.6 GHz [38].

The compiler used was GCC 4.1.0 and the operating system was Linux with kernel version 2.6.16.27. Traces were obtained in Paraver format using the Mintaka code instrumentation tool [185].

The code was executed using a 3 thread configuration. The first thread, called the master thread, is responsible of the main control of the application, bitstream file reading, frame parsing and all the slice initialization and finalization code. The second thread is responsible for CABAC entropy decoding. CABAC results are stored in a frame buffer for later use of by the MB decoding stage. Finally, the third thread is is responsible of MB decoding (more exactly MB reconstruction, including IDCT, IQ, prediction and deblocking filter).

The code was instrumented to obtain traces with CPU phases, synchronization operations and memory accesses. Execution of each thread was divided in different CPU phases and each phase was instrumented including a phase identifier and its execution time.

Synchronization operations are special instrumentation marks that identify when a task either waits for some signal to be ready or posts a signal announcing the availability of some data. For example, CABAC decoding posts a signal every time it finishes the entropy decoding of a frame and the worker thread that process the first MB of a frame issues a wait operation for that signal.

Instrumented memory operations contain contains parameters such as address in main memory, size of the transfer in bytes and direction (read or write).

Paraver traces were processed with the *prv2ttf* tool to produce another trace with a task format. In the final trace there is a collection of separated tasks, each one with information of execution time of kernels, synchronization information and memory accesses. It is important to note that these are not instruction traces, but tasks traces. They contain information of CPU bursts that happen between synchronization or memory operations. The duration of these CPU burst are used for simulating systems with different number of cores as we will explain next. Our simulation methodology is similar, in concept, to the one developed by Seitner et al. [213].

10.3.3 Trace-driven Simulation

For our simulations we have used TaskSim. TaskSim is a trace-driven simulator for accelerator-based multi-core architectures. It targets the simulation of parallel applications coded in a master-worker task offload computational model. It uses task traces that contains information about the inter-task dependencies. That information allows TaskSim to reconstruct the dependencies at simulation time.

The computational CPU phases (bursts), such as task execution, are not simulated in detail. The burst duration is obtained from the trace and is simulated as a single event with the same runtime as the whole burst. Contrarily, the trace time for phases involving access to shared resources in the architecture, such as waiting for DMA transfers or synchronization operations, are discarded, and their timing is simulated in a cycle-accurate way by means of detailed simulation of DMA controllers, caches, interconnection, memory controllers, and DRAM modules [197]..

10.3.4 The SARC Architecture

As a baseline for simulation we have used the SARC architecture [192]. SARC is a heterogeneous multicore composed of a set of processors managed at runtime in a master-worker mode. The architecture includes different type of cores, an on-chip interconnection network, a multi-bank shared on-chip data cache, on-chip memory controllers and external memory as shown in Figure 10.3.

The Master processors (M) start the main() routine of the program, and run the core of the application generating tasks to be off-loaded to the specialized Worker (W) processors, as indicated by a software runtime scheduler. Worker cores execute tasks generated by the Masters ones. In this work we consider two types of workers: MB decoders and CABAC entropy decoders.

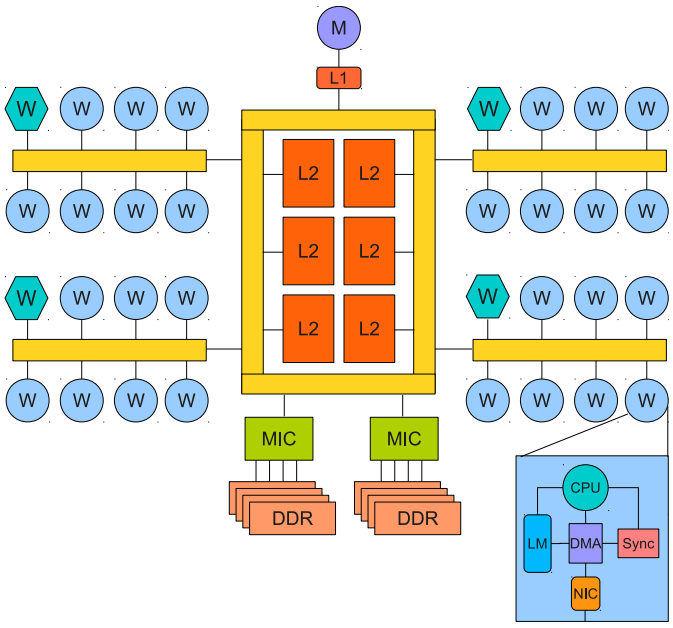


Figure 10.3: Baseline heterogeneous multicore architecture

All processors have direct load/store access to the different scratchpads and the off-chip memory. Workers also have a DMA controller that can transfer data to/from the scratchpad memories, overlapping data transfer and computation.

Figure 10.3 shows a general diagram of the architecture and a detailed view of a worker node. The node is composed of a worker core (W), a local memory (LM), a DMA controller, and a Network Interface Controller (NIC) that arbitrates the accesses to the bus.

The architecture has been extended to include a hardware synchronization facility. It includes a sub-unit in each worker core associated to its DMA controller for handling semaphore operations, and the addition of a global synchronization unit that keeps track of semaphore state. Hardware semaphore operations are handled as special memory operations and use the same interconnection network than other memory accesses. The main difference is that hardware semaphores can be optimized, for example, mapping them to on-chip local memories that are not shared with other data.

As shown in Figure 10.3, workers are organized in clusters of N processors. In a 128-worker configuration, for example, the global NoC connects together 16 clusters of 8 processors.

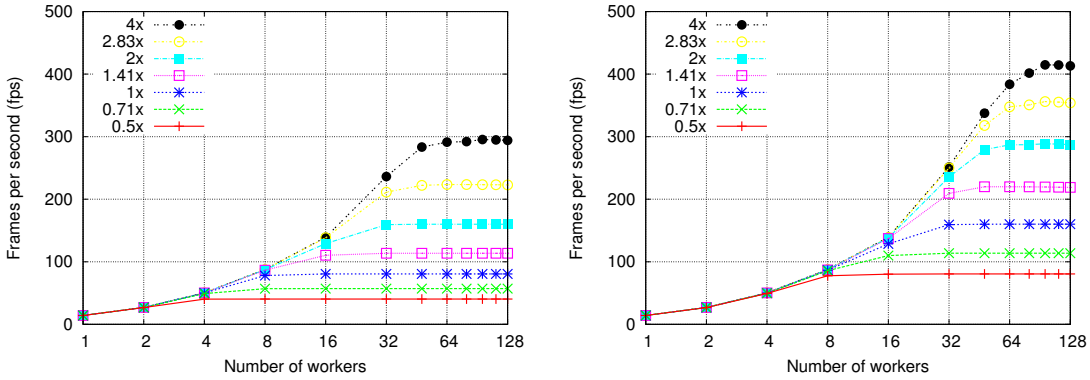
Assuming 128 cores as a maximum we have defined a baseline configuration. The corresponding parameters of the simulator are shown in Table 10.2.

10.4 Experimental Results

In this section we present and discuss the experimental results for the proposed hardware and software optimizations.

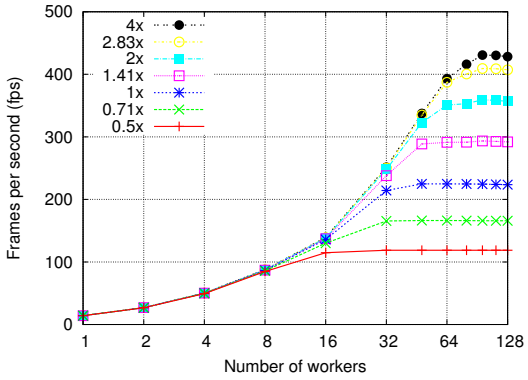
Memory controllers	4 x 2 DDR3 channels
Channel bandwidth	12.8 GB/s (DDR3-1600)
Memory latency	Real DDR3-1600
MIC policy	Closed-page, in-order processing
Shared L2 cache	128 MB (32 x 4 MB), 4-way assoc.
Sync. unit latency	256 cycles
L2 cache latency	40 cycles
Local Store	256 KB, 6 cycles
Interconnection links	8 bytes/cycle (25.6 GB/s)
Intra-cluster NoC	2-bus (51.2 GB/s)
Global NoC	16-bus (409.6 GB/s)

Table 10.2: Baseline simulation parameters

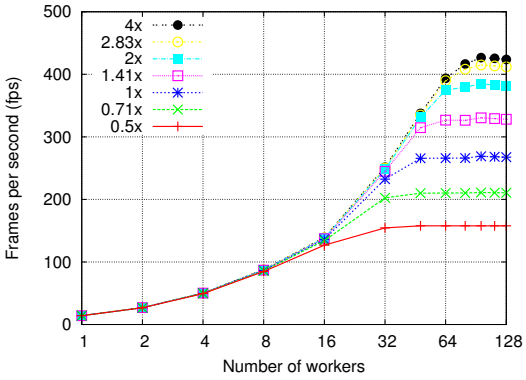


(a) 1 CABAC worker

(b) 2 CABAC workers



(c) 3 CABAC workers



(d) 4 CABAC workers

Figure 10.4: CABAC acceleration and multiple CABAC processors for the 3D-Wave H.264 decoder

10.4.1 Dynamic 3D-Wave with Multiple CABAC Processors

In order to analyze the effect of the parallelization of the CABAC entropy decoding we have conducted an experiment varying the number and acceleration of CABAC processors for a 3D-Wave parallel decoder with a maximum of 8 frames in flight. The resulting performance can be seen in Figure 10.4 for 1, 2, 3 and 4 CABAC cores.

Figure 10.4a shows the results for 1 CABAC core. In this case, and without CABAC

Configuration	Low power	High Perf.
ISA	X86-64	X86-64
SIMD extensions	SSSE3	SSSE3
Processor	Atom 330	Xeon E7310
Cores	2	4
Technology	45 nm	65 nm
Clock frequency	1.6 GHz	1.6 GHz
Power	8 W	80W
Level 1 I-cache	32KB	32KB
Level 1 D-cache	24KB	32KB
Level 2 cache	1 MB	4 MB
Main Memory	1.5 GB	16 GB
Operating System	Linux 2.6.32-24	Linux 2.6.32
Compiler	GCC-4.4.3 -O3	GCC-4.4.1 -O3

Table 10.3: Processor configuration for the heterogeneous system

acceleration (1X), a maximum performance of 80 fps is obtained with 8 MB decoding cores. In order to obtain more performance some acceleration on the CABAC core is needed. For example: 100 fps requires one CABAC core at 1.41X acceleration and 16 worker processors.

When the number of CABAC cores increases the scalability of the whole application improves. The decoder is able to reach 266 fps with 4 CABAC cores at 1X and 48 MB decoder cores. With 4 CABAC cores the CABAC front-end is not longer the bottleneck and the number of frames in flight starts to become the limiting factor. This reflects a clear tradeoff between throughput and latency (and memory usage).

Having multiple CABAC processors with the 3D-Wave algorithm allows to reach a fixed performance target even with de-accelerated CABAC cores. For example, FHDp100 can be reached with 16 MB decoding cores and 1 CABAC core at 1.41X, 2 CABAC cores at 0.71X, or 3 CABAC cores at 0.5X. Multiple de-accelerated cores use less area and power than one highly accelerated one.

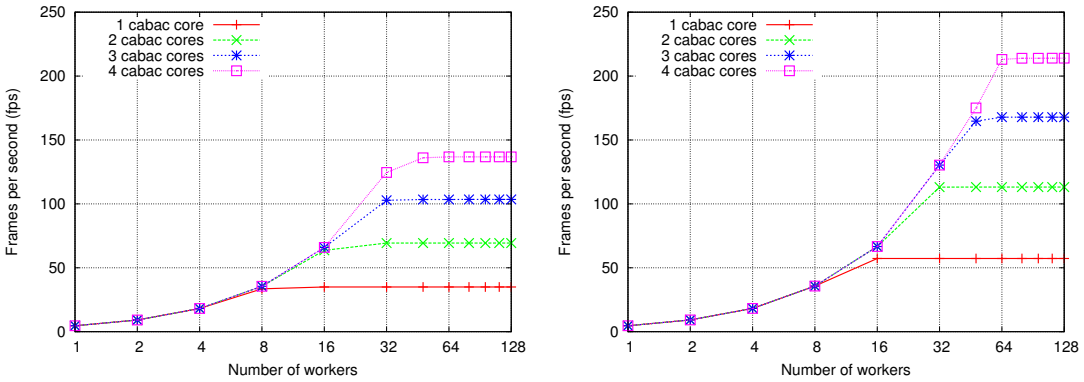
It should be noted that the performance required to meet a real-time target depends on the input content (spatial and temporal characteristics of video). The performance and number of CABAC processors can be adjusted dynamically depending on the requirements on the specific execution of the application. This is an open research area.

10.4.2 Case Study: Heterogeneous Manycore Architecture

In this section we provide results for a different configuration of cores. Instead of using the Intel Itanium-2 cores we use low power processors for MB decoding combined with high performance cores for CABAC decoding. MB cores are SIMD processors based on the Intel Atom architecture which is a low power processor for mobile devices [86]. Entropy decoding cores are superscalar processors based on the Intel Xeon architecture. Both processors have the same ISA and run at the same frequency but have a very different microarchitecture and memory hierarchy (asymmetric cores). Table 10.3 shows the main parameters for both processors.

Figure 10.5 shows the resulting performance in terms of frames per second for the homogeneous (Figure 10.5a) and the heterogeneous cases (Figure 10.5b).

First, it is important to note that it is not possible to decode FHD video in real-time (25 fps) on a single Atom core. Using a homogeneous system based on Atom processors



(a) homogeneous

(b) heterogeneous

Figure 10.5: Parallel H.264 decoder with different type of cores

requires at least 1 CABAC core and 8 MB cores to go beyond 25 fps. Decoding 50 fps requires 2 CABAC cores and 16 MB cores. The system is able to scale up to 48 MB cores with 4 CABAC cores reaching almost 140 fps. Adding more cores (CABAC or MB) results in diminishing benefits due to the limit of CABAC frames in flight.

The heterogeneous configuration improves the performance of all the simulated configurations by 1.64X in average. Using asymmetric cores it is possible to decode 50 fps with 1 CABAC core and 16 MB cores. As a maximum, the system is able to decode 213 fps using 64 MB cores and 4 CABAC cores (plus one master core, for 69 cores in total).

10.4.3 Impact of Thread Synchronization

In order to analyze the impact of the thread synchronization facility we have conducted an experiment in which we assign different latencies to the synchronization operations ranging from 1 to 65536 cycles. As a baseline for the simulations we use an asymmetric system with 4 CABAC cores and 128 MB decoding cores.

Figure 10.6 shows the resulting speedup. In order to achieve a performance close to the maximum (less than 1% slowdown) the average synchronization latency should be less than 1024 cycles. A conservative value of the latency of a hardware semaphore operation is the latency of L2 cache hit. With this latency the performance is almost the same (less than 0.01%) as the peak performance with zero latency. This can only be achieved with on-chip accesses like the offered by the hardware synchronization facility. Latencies bigger than 16000 cycles, which are in the range of the average MB decoding time have more than 2X reduction in performance.

10.4.4 Memory Requirements

Previous experiments have been conducted using a powerful configuration of main memory and cache hierarchy as shown in Table 10.2. In this section we evaluate the actual memory requirements. As a baseline we configured a system with 4 CABAC cores and 64 MB decoding cores (and 1 master core) using the heterogeneous configuration

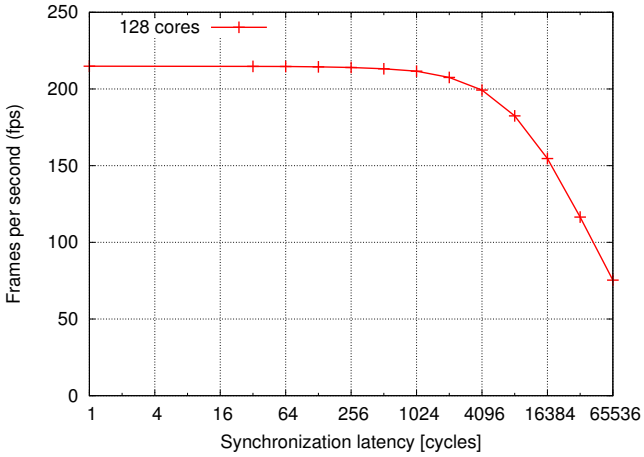
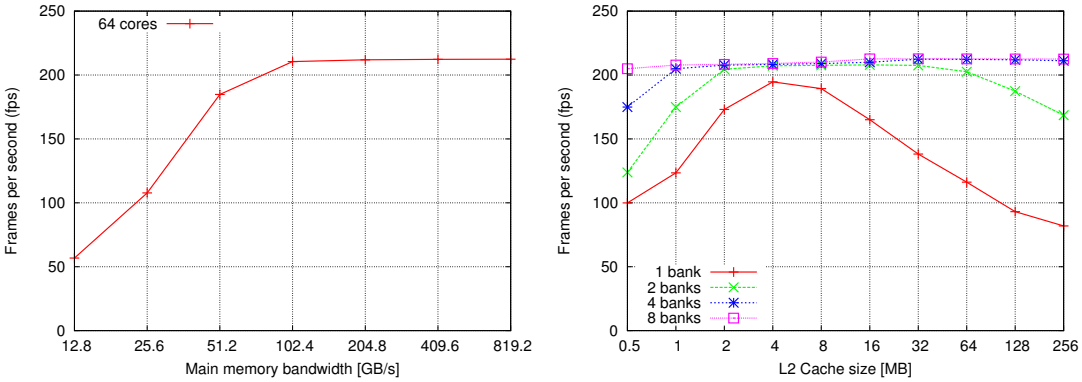


Figure 10.6: Latency effect of synchronization operations



(a) Memory bandwidth impact

(b) Shared data cache impact

Figure 10.7: Memory requirements

presented in section 10.4.2.

Impact of Main Memory Bandwidth

In order to isolate the effects of main memory bandwidth we have disabled the L2 cache. Figure 10.7a shows the results. For reaching the maximum performance, 212 fps, a minimum of 102.4 GB/s are required. This can be provided with 4 MICs each one having two DDR-3-1600 modules.

As we will show in the next section the use of the on-chip cache helps to reduce an important amount of the off-chip traffic.

Impact of Shared Data Cache

In the baseline architecture each processor is equipped with two types of individual memories: a scratchpad and a L1 data cache. In the case of the 3D-Wave decoder all

the accesses to main memory have been implemented using explicit DMA commands. As a result all the local accesses use the scratchpad memory but not the L1 data cache.

A shared multi-bank L2 cache is included in the architecture as it has been explained in section 10.3.4. The shared cache maintains a significant part of the memory accesses on-chip by capturing the references made by DMA controllers. With a multi-bank structure and a careful mapping of accesses to banks it is possible to provide high on-chip bandwidth and low latency access.

In order to determine the best cache configuration we change both the number of banks and the cache size. The latencies of cache banks with difference sizes have been estimated using CACTI 5.3 [247], for 45nm memory technology and the system is simulated with a main memory composed of 1 MIC with two DDR3 channels (12.8 GB/s). Figure 10.7b shows the effect of cache banks. The maximum performance of 212 fps can be reached either having a big cache (4MB) with small number of banks (2 banks), or having a smaller cache (1 MB) with a high number of banks (8 banks). In the first case a larger cache means a higher access latency, while in the second case a banked cache has lower latency but requires more bandwidth on the on-chip interconnection network.

10.5 Related Work

Most of the previous works on MB-level parallelization of the H.264/AVC decoder do not take into account the entropy decoding stage [69, 269, 24, 45]. Instead they assume the availability of some kind of hardware accelerator either in the form of special purpose units inside a media processor or as dedicated hardware accelerators [252, 177, 113]. With these solutions it is possible to achieve the real-time requirements of a target application with a modest hardware investment, but they have a reduced scalability and flexibility.

Recent implementations on the Cell processor proposed a combination of intra- and inter-frame MB-level parallelism. They execute entropy decoding on the PPE processor and MB decoding onto the SPE processors [26, 48, 50].

GPUs are also heterogeneous multicore architectures but some attempts to use them for video decoding exhibit limited performance gains because of the irregular behavior of H.264 decoding [184].

Regarding the problem of thread synchronization there are different approaches reported in the literature. That includes implementations in software, in hardware or in combinations of both. Software implementations can use blocking locks, spin-locks or non-blocking synchronization [126], while hardware ones consist on implementing the basic operations of task pools, like *enqueue* and *dequeue* using hardware controlled data structures [6, 208, 268]. Software synchronization is more flexible but results in operations with higher latency that can degrade performance of fine-grain tasks. Hardware task management can provide faster synchronization at the cost of limiting the flexibility because they are application specific.

By contrast, we support a mixed model in which simple hardware synchronization modules accelerate task scheduling algorithms implemented in software. The evaluated hardware semaphores are similar in concept to previous works like [249, 207]. The main difference is that our approach is oriented to task-based parallel programs, it is not specific to SMT processors and the evaluated hardware semaphores use the same

interconnection network than regular memory operations.

10.6 Summary

We have presented an analysis of the scalability of parallel video decoding on heterogeneous manycore architectures. We have shown that it is possible to remove the entropy decoder bottleneck by exploiting multiple levels of parallelism such as pipeline parallelism, frame-level parallelism and macroblock-level parallelism.

We have demonstrated that is possible to achieve the performance required by high quality applications using processors with multiple simpler cores operated at a reduced rate compared to the base processor. We also presented the performance of a heterogeneous manycore architecture in which simpler low power processors are assigned to data parallel kernels and high performance cores for entropy decoding. Using this configuration it was possible to achieve high performance with a reduced power consumption.

We also evaluated the impact of a hardware accelerated synchronization facility. It allowed to process fine grain dependent tasks with minimal overheads. Finally, we evaluated the memory requirements of the application and we found a realistic memory configuration, in terms of bandwidth and cache size, that allows to reach almost the maximum performance.

In this paper we only consider a static combination of high performance and low power cores. One open research area is the design of dynamic systems in which processor performance can be adjusted at run-time and tasks can be allocated dynamically to heterogeneous processors based on task complexity.

11 Conclusions

The main problem addressed in this thesis has been how to provide the performance required by high quality video decoding applications using programmable processors. As there is not a single solution that can provide all the required performance, and as the demands for performance in the video domain are increasing continuously, the solution adopted has been the simultaneous exploitation of multiple levels of parallelism.

This solution has required two main areas of work. On the one hand, we have modified the video codec software in order to exploit different types of parallelism. On the other hand, we have modified the architecture in order to exploit the type of parallelism that is available in video codec applications.

This combined solution has been carried out following three main requirements. The first one is the use of programmable processors instead of application specific hardware in order to support a complete application domain; H.264/AVC has been a design example rather than a specific problem. The second one is to improve the performance scalability of the application. And finally, the third requirement has been to achieve the required performance with an efficient use of resources specially power consumption.

11.1 Contributions

In this section we are going to summarize the main contributions of this thesis

11.1.1 Scalability of Multidimensional Vector Architectures

We analyzed the scalability of different SIMD extensions for video coding and decoding using the MPEG-2 video codec. We compared the scalability of generic 1D SIMD extensions (like Intel MMX) and a a 2D matrix architecture. Both extensions were scaled by increasing the width of registers and by augmenting the number of functional units.

We have shown that a 2D-vector extension with 128-bit registers has a higher performance and a lower complexity compared to 1D extensions. The performance gains of the 2D-vector architecture are the result of a good matching between data structures in video applications and the matrix architecture.

It was shown that for the type of video codec under study the 2D-vector architecture was reaching the limits of available DLP. Further scaling of matrix registers can not deliver significant performance improvements because the execution time is now dominated by the scalar code. This situation is more notorious in recent video codecs like H.264/AVC in which smaller (and variable size) blocks are used. This can change in the future if the emerging video coding standards addressed specifically for HD resolutions, like HEVC [234], include bigger coding units like 64×64 data blocks.

Related Publications

- Mauricio Alvarez, Friman Sánchez, Esther Salamí, Alex Ramírez, and Mateo Valero. Scalability and Complexity of 2-Dimensional SIMD extensions. In *XV Jornadas de paralelismo*, September 2004

- Friman Sánchez, Mauricio Alvarez, Esther Salamí, Alex Ramírez, and Mateo Valero. On the Scalability of 1- and 2-Dimensional SIMD Extensions for Multimedia Applications. In *ISPASS. IEEE International Symposium on Performance Analysis of Systems and Software*, pages 167–176, March 2005

11.1.2 A Benchmark for High Definition Video Codec Applications

We made a first attempt to characterize the H.264/AVC video decoder using the reference code because it was the only code available at that time. We profiled that code and identified the most time consuming kernels that were, in order of relevance: motion compensation (specially luma and chroma interpolation), deblocking filter, entropy decoding and IDCT. Compared with previous video codecs, those kernels required more computing resources and exhibited a more irregular behavior (due to variable block size, multiple coding options, etc). We performed a SIMD optimization of the most important kernels and the results indicated that SIMD optimizations were not enough to provide the performance required for real-time operation.

When an optimized code became available we compared it with the reference one and we found that the former was, at least, one order of magnitude faster than the latter. With this information we concluded that the use of the reference code end in misleading results, especially for complexity and architectural studies.

The lack of a proper benchmark for video codec applications lead us to create our own benchmark using applications that meet a set of common accepted criteria such as the use of complete applications optimized for high performance, code portability and free license. The benchmark also included a set of test sequences available in HD resolutions.

By meeting all these requirements this benchmark allowed us to make fair comparisons of different video codecs in terms of coding performance and complexity. We have notice that the benchmark has been used by other researchers in the field.

Related Publications

- Mauricio Alvarez, Esther Salami, Alex Ramirez, and Mateo Valero. A Performance Characterization of High Definition Digital Video Decoding Using H.264/AVC. In *IEEE International Symposium on Workload Characterization*, pages 24–33, Oct 2005

- M. Alvarez, E. Salami, A. Ramirez, and M. Valero. HD-VideoBench: A Benchmark for Evaluating High Definition Digital Video Applications. In *IEEE Int. Symp. on Workload Characterization*, pages 120–125, Sept. 2007. URL `http://people.ac.upc.edu/alvarez/hdvideobench`

11.1.3 Efficiency of SIMD extensions for Exploiting DLP

It has been demonstrated that SIMD extensions have some inefficiencies when processing video data. We studied the source of these inefficiencies and analyzed different mechanism to remove or, at least, minimize them. In particular, we analyzed in detail the impact of unaligned accesses to memory and proposed architectural solutions for minimizing the performance loss due to this type of memory operations.

Although there was a wide consensus in the computer architecture and video codec communities that the support for unaligned accesses is a necessity in video processing algorithms, there was not a quantitative analysis showing the performance impact of supporting them.

We presented the hardware and software required to have an efficient support for unaligned accesses and we evaluated its impact on the H.264/AVC decoder application. With the results of our study a designer can trade-off the added complexity of supporting unaligned accesses with the resulting performance benefits. It can be noted that the support for unaligned instructions is just one example of ISA improvements to the data layout of SIMD extensions. Other instructions, like partial load and stores, and indexed accesses could result in more performance gains for these applications.

Related Publications

- Mauricio Alvarez, Esther Salamí, Alex Ramírez, and Mateo Valero. Performance Impact of Unaligned Memory Operations in SIMD Extensions for Video Codec Applications. In *IEEE International Symposium on Performance Analysis of Systems Software, ISPASS 2007*, pages 62–71, April 2007

11.1.4 Thread-level Parallelization of Video Decoding

In a first step, we perform a detailed analysis of the different thread-level parallelization strategies that can be applied to a H.264/AVC video decoder. That includes function-level parallelism and several strategies of data-level parallelism such as frame-level, slice-level and macroblock-level parallelism. We analyzed several techniques and showed that none of them were able to scale to manycore systems.

In order to solve this limitation we proposed a new parallelization algorithm based on intra- and inter-frame macroblock-level parallelism. This technique, called the dynamic 3D-Wave, was able to extract, in theory and depending on the input content, thousands of independent tasks.

It is worth mentioning that this initial study did not include practical aspects regarding the implementation on real platforms. Its objective was to analyze the sources of task-level parallelism in an application that was been traditionally considered hard to parallelize. The revelation of a huge amount of fine-grain parallelism lead us to analyze how to exploit it efficiently in multicore architectures.

Related Publications

- Cor Meenderinck, Arnaldo Azevedo, Mauricio Alvarez, Ben Juurlink, and Alex Ramirez. Parallel Scalability of H.264. In *Workshop on Programmability Issues for Multi-Core Computers (MULTIPROG)*, Jan. 2008

- Cor Meenderinck, Arnaldo Azevedo, Mauricio Alvarez, Ben Juurlink, and Alex Ramirez. Parallel Scalability of Video Decoders. *Journal of Signal Processing Systems*, 57:173–194, November 2009

11.1.5 Scalability of Macroblock-level Parallelism

We implemented intra-frame macroblock-level parallelism in order to study its performance on a real parallel machine and in order to identify sources of overhead or bottlenecks that could limit the parallel scalability.

We built a formal model and an abstract trace-driven simulator to estimate the theoretical limits of the parallelization. Then, we compared these theoretical estimations with an implementation on a cc-NUMA parallel computer. This implementation included the analysis of different scheduling algorithms. One of the observed limitations was thread synchronization overhead, which is the result having fine-grain tasks and the use long latency synchronization operations. We analyzed the impact of using a decentralized scheduling technique (called "tail-submit") that reduces the number of synchronization operations and, at the same time, exploits data locality. Using these technique it was possible to reach a maximum speedup of 10X on a system with 22 processors.

The other limitation was the entropy decoder stage which can not be parallelized at the macroblock-level. The performance of the complete application depends on the performance of this sequential stage, and the applicability of any macroblock-level parallelization strategy depends on finding any method to accelerate CABAC entropy decoding.

Related Publications

- Mauricio Alvarez, Alex Ramírez, Arnaldo Azevedo, Cor Meenderinck, Ben Juurlink, and Mateo Valero. Scalability of Macroblock-level Parallelism for H.264 Decoding. In *The Fifteenth International Conference on Parallel and Distributed Systems (ICPADS'09)*, Dec 2009

- M. Alvarez, A. Ramirez, M. Valero, A. Azevedo, C.H. Meenderinck, and B.H.H. Juurlink. Performance evaluation of macroblock-level parallelization of h.264 decoding on a cc-numa multiprocessor architecture. In *Proceedings of the 4CCC: 4th Colombian Computing Conference*, April 2009

- M. Alvarez, A. Ramirez, M. Valero, A. Azevedo, C.H. Meenderinck, and B.H.H. Juurlink. Performance evaluation of macroblock-level parallelization of h.264 decoding on a cc-numa multiprocessor architecture. *Avances en Sistemas e Informática*, 6(1):219–228, June 2009

- Arnaldo Azevedo, Cor Meenderinck, Ben Juurlink, Andrei Terechko, Jan Hoogerbrugge, Mauricio Alvarez, and Alex Rammirez. Parallel H.264 Decoding on an Embedded Multicore Processor. In *Proceedings of the 4th International Conference on High Performance and Embedded Architectures and Compilers - HIPEAC*, Jan 2009

- Arnaldo Azevedo, Ben Juurlink, Cor Meenderinck, Andrei Terechko, Jan Hoogerbrugge, Mauricio Alvarez, Alex Ramirez, and Mateo Valero. A highly scalable parallel implementation of h.264. *Transactions on High-Performance Embedded Architectures and Compilers*, 4(2), 2009

11.1.6 Scalability of Heterogeneous Manycore Architectures

We removed the main bottlenecks of the parallel H.264 decoder using a heterogeneous manycore architecture.

Firs, we addressed the limitation imposed by the entropy decoding stage. We demonstrated that it is possible to combine multiple levels of parallelism: pipeline parallelism at the frame-level between the entropy decoding and the macroblock decoding stages; frame-level parallelism in the entropy decoding stage and macroblock-level parallelism in the macroblock decoding stage.

Based on that, we proposed a heterogeneous architecture that includes two main type of cores. The first one consists of a large array of relatively simple processors which are very effective for macroblock decoding kernels (like IDCT, motion compensation and deblocking filter). The second one is formed by a small set of high performance cores which are appropriate for general coordination and entropy decoding.

In order to overcome the overheads imposed by thread synchronization the architecture was enhanced with hardware accelerated semaphores. They offer very low latency synchronization operations with low complexity. Hardware semaphores not only speedup the application but they also simplifies the parallel code. It is worth mentioning that this synchronization infrastructure is useful but for other parallel applications with fine grain tasks.

Finally, we evaluated the memory requirements. Our results show that a system with one MIC, two DDR-3 channels, and a shared cache with 1MB and 8 banks can sustain decoding of HD content at a rate of more than 200 fps.

Related Publications

- A. Ramirez, F. Cabarcas, B. Juurlink, M. Alvarez Mesa, A. Azevedo, C. Meenderinck, G. Gaydadjiev, C. Ciobanu, S. Isaza, and F. Sanchez. The SARC Architecture. *IEEE Micro*, 30(5):16–29, Sept/Oct. 2010

- Mauricio Alvarez, Felipe Cabarcas, Alex Ramírez, Cor Meenderinck, Ben Juurlink, and Mateo Valero. Scalability of heterogeneous multicores for parallel video decoding. In *Submitted to the 2011 IEEE Internation Conference on Worload Characterization IISWC*, June 2011

11.1.7 Other Publications

A general publication that will contain a summary of the main contributions of this thesis, among other work in the area of parallelization of video decoding applications, will be published in 2011 as a book.

- Mauricio Alvarez-Mesa, Ben Juurlink, Chi Ching Chi, Arnaldo Azevedo, Cor Meenderinck, and Alex Ramírez. *Scalable Parallel Programming Applied to H.264 Decoding*. Springer, To be published in 2011

11.2 Open Areas for Research

The research done in this thesis opens different areas for further research. Open areas are divided into modifications to video codecs and enhancements to the architecture.

11.2.1 Modifications to Video Codecs

In this section we mention some modifications that can be applied to the video codec design for supporting parallel architectures.

As the process for designing of a new international video codec standard has been started it is a good time to send feedback from the computer architecture domain to the video coding domain about the capabilities and trade-offs of computer architectures. The most important message is that the new generation of video codecs will be executed on parallel platforms. Given that, new video codecs should include some type of support for parallel execution. We are working right now on some proposals for parallelization of the upcoming of the HEVC video codec. Here we present some of these ideas.

Support DLP with Long Vectors

In order to exploit DLP with SIMD and vector extensions is better to have regular and long data structures rather than small and irregular ones like in H.264/AVC. For HD resolutions and beyond it could be possible to have coding blocks with larger size. This can increase compression efficiency and, at the same time, increase computing performance allowing more data to be processed per SIMD instruction. This is the case with the emerging HEVC video standard [234], that allows coding blocks up to 64×64 samples.

Avoid Branches (When Possible)

In most parallel architectures it is better to perform more operations rather than execute branches. Video filters with a lot of branches, like the deblocking filter in H.264/AVC, make parallelization difficult. From the architecture perspective it would be better to apply a more complex filter to all samples rather than detect type of filter for each input sample. The same applies to motion compensation interpolation.

Prefer Arithmetic Operations Rather than Memory Accesses

In h.264/AVC and other video codecs it has been common to reduce the arithmetic complexity by replacing "complex" arithmetic operations like multiplications with accesses to tables of precomputed values. However, in some multicore architectures (like GPUs) it takes more time to access data than to execute an arithmetic operation. Off-chip bandwidth is a scarce resource but arithmetic units are abundant. In such architectures it would be better to computer more, or even recompute, rather than access tables of precomputed values.

Have Clear (and Explicit) Data Dependencies

What makes parallel programming more difficult is identifying implicit data dependencies in algorithms. In video codecs it should be clear that predicting data from neighbor

data structures generates data dependencies and that they reduce the amount of parallelism. Additionally, having dynamic dependencies complicates the scheduling process in parallel systems.

From the parallelization point of view, it would be extremely useful if all the data dependencies of different data units (frames, slices, macroblocks, etc.) were clearly and explicitly defined in the standard. For example, the dependencies of a macroblock with other macroblocks in the same and other frames can be explicitly marked at the beginning of it. This will allow the scheduler to make better decisions of when and where to submit each particular tasks.

Other alternatives is to reduce the dependencies for some kernels, like the deblocking filter or intra-prediction, allowing parallel implementations in massive parallel processors like GPUs.

Support the Parallelization of the Entropy Decoding Stage

As it was mentioned in Chapter 9, one of the main limitations for parallel video decoding is the sequential behavior of the entropy decoding stage, specially with the CABAC algorithm. There are two main limitations: one is that the data dependencies between 'bins' in the coded bitstream which inhibits almost any fine-grain parallelization of the CABAC decoder. The second one is the requirement to execute the CABAC stage sequentially for all the macroblocks in a slice, which limits the macroblock-level parallelization and affects negatively the cache locality because requires frame buffers between entropy decoding and macroblock decoding.

A solution to this problem is to design the entropy coding algorithm taking into account that it will be executed on parallel platforms. The definition of slices and the coding of macroblocks could be changed to allow thread-level parallelism in the entropy decoding stage. One solution could be the use of traditional slices, but this have some limitations like decreased coding efficiency, load balancing and others (see Chapter 8). An alternative could be the use of entropy slices [77] in which the bitstream is split in slices that can exchange information for improving coding efficiency while, at the same time, allow the parallel decoding of each row in wavefront order. Additionally, the organization of syntax elements in the bitstream could be changed for allowing parallel processing of entropy decoding of macroblocks inside each slice/frame. This can be done, for example, by organizing the bitstream according to the type of the syntax elements rather than by complete macroblocks [36]. This scheme can be augmented with estimations of macroblock complexity that can be added as a syntax element, and which can be used as information for a dynamic scheduler in heterogeneous systems.

11.2.2 Modifications to the Architecture

This section includes different proposals for adapting processor architecture and microarchitecture for video decoding applications.

SIMD extensions for Video Decoding Applications

As we have shown in Chapter 4, multidimensional vector architectures are a scalable and efficient solution for processing video data. They offer high performance with less complexity compared to traditional SIMD extensions. In emerging video codecs, like

HEVC, there are proposals for increasing the size of basic coding units [234]. For bigger data blocks vector architectures offer a big advantage over traditional SIMD extensions in terms of performance, complexity and power efficiency.

Emerging SIMD extensions like Intel AVX uses wider registers with 256-bits and have been designed with scalability options to 512 or 1024 bits but they continue to be organized as a lineal register with limited support for multidimensional data structures. A 2D-vector architecture, as we demonstrated in this thesis, can handle in a more efficient way the variable block size that appears in advanced video codecs.

Heterogeneous Multicore Architectures

We have shown that a heterogeneous multicore architecture achieves higher performance for parallel video decoding. In our analysis we used a static combination of cores, which means that we have defined a-priori the performance, type and number of processors. For example, we evaluated the performance of a system with 4 CABAC "high performance cores", 1 master core, and 64 macroblock decoding "simple cores". But, it should be noted that the performance required to meet a real-time target depends on characteristics of the input content. In order to match variable application requirements the performance and number of each type of processors can be adjusted dynamically.

This requires a dynamic scheduling approach in which the scheduling decisions are based on task complexity and the scheduler is aware of processor heterogeneity. In such scenario, task are submitted to processors according to their complexity and the capabilities of the processor. "Simple tasks" are submitted to "simple processors" and "complex tasks" to "complex" ones.

The implementation of this mechanism requires a fast (probably with the help of hardware acceleration) task scheduler that can make scheduling decisions with a very low latency. The problem becomes even more complex when the scheduler is allowed to change the performance of the processors dynamically using, for example, frequency and voltage scaling for reducing power consumption.

Bibliography

[1] V. Agarwal, M.S. Hrishikesh, S.W. Keckler, and D. Burger. Clock rate versus IPC: the end of the road for conventional microarchitectures. In *Proceedings of the 27th International Symposium on Computer Architecture, 2000*, pages 248–259, 2000.

[2] S. Agarwala, T. Anderson, A. Hill, M.D. Ales, R. Damodaran, P. Wiley, S. Mullinnix, J. Leach, A. Lell, M. Gill, A. Rajagopal, A. Chachad, M. Agarwala, J. Apostol, M. Krishnan, Duc Bui, Quang An, N.S. Nagaraj, T. Wolf, and T.T. Elappuparackal. A 600-MHz VLIW DSP. *IEEE Journal of Solid-State Circuits*, 37(11):1532–1544, nov 2002.

[3] N. Ahmed, T. Natarajan, and K. R. Rao. Discrete Cosine Transform. *IEEE Transactions on Computers*, C-23(1):90–93, Jan 1974.

[4] S.M. Akramullah, I. Ahmad, and M.L. Liou. Optimization of h.263 video encoding using a single processor computer: performance tradeoffs and benchmarking. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(8):901–915, aug 2001.

[5] S.M. Akramullah, I. Ahmad, and M.L. Liou. Performance of software-based mpeg-2 video encoder on parallel and distributed systems. *IEEE Transactions on Circuits and Systems for Video Technology*, 7(4):687–695, Aug 1997.

[6] Ghiath Al-Kadi and Andrei Sergeevich Terechko. A hardware task scheduler for embedded video processing. In *Proceedings of the International Conference on High Performance Embedded Architectures and Compilers*, pages 140–152, 2009.

[7] Alexis M. Tourapis. Enhanced predictive zonal search for single and multiple frame motion estimation. In *Proceedings of SPIE Visual Communications and Image Processing 2002*, pages 1069–1079, Jan. 2002.

[8] M. Alvarez, E. Salami, A. Ramirez, and M. Valero. HD-VideoBench: A Benchmark for Evaluating High Definition Digital Video Applications. In *IEEE Int. Symp. on Workload Characterization*, pages 120–125, Sept. 2007. URL `http://people.ac.upc.edu/alvarez/hdvideobench`.

[9] M. Alvarez, A. Ramirez, M. Valero, A. Azevedo, C.H. Meenderinck, and B.H.H. Juurlink. Performance evaluation of macroblock-level parallelization of h.264 decoding on a cc-numa multiprocessor architecture. In *Proceedings of the 4CCC: 4th Colombian Computing Conference*, April 2009.

[10] M. Alvarez, A. Ramirez, M. Valero, A. Azevedo, C.H. Meenderinck, and B.H.H. Juurlink. Performance evaluation of macroblock-level parallelization of h.264 decoding on a cc-numa multiprocessor architecture. *Avances en Sistemas e Informática*, 6(1):219–228, June 2009.

[11] Mauricio Alvarez, Friman Sánchez, Esther Salamí, Alex Ramírez, and Mateo Valero. Scalability and Complexity of 2-Dimensional SIMD extensions. In *XV Jornadas de paralelismo*, September 2004.

[12] Mauricio Alvarez, Esther Salami, Alex Ramirez, and Mateo Valero. A Performance Characterization of High Definition Digital Video Decoding Using H.264/AVC. In *IEEE International Symposium on Workload Characterization*, pages 24–33, Oct 2005.

[13] Mauricio Alvarez, Esther Salamí, Alex Ramírez, and Mateo Valero. Performance Impact of Unaligned Memory Operations in SIMD Extensions for Video Codec Applications. In *IEEE International Symposium on Performance Analysis of Systems Software, ISPASS 2007*, pages 62–71, April 2007.

[14] Mauricio Alvarez, Alex Ramírez, Arnaldo Azevedo, Cor Meenderinck, Ben Juurlink, and Mateo Valero. Scalability of Macroblock-level Parallelism for H.264 Decoding. In *The Fifteenth International Conference on Parallel and Distributed Systems (ICPADS'09)*, Dec 2009.

[15] Mauricio Alvarez, Felipe Cabarcas, Alex Ramírez, Cor Meenderinck, Ben Juurlink, and Mateo Valero. Scalability of heterogeneous multicores for parallel video decoding. In *Submitted to the 2011 IEEE Internation Conference on Worload Characterization IISWC*, June 2011.

[16] Mauricio Alvarez-Mesa, Ben Juurlink, Chi Ching Chi, Arnaldo Azevedo, Cor Meenderinck, and Alex Ramírez. *Scalable Parallel Programming Applied to H.264 Decoding*. Springer, To be published in 2011.

[17] AMD. AMD Phenom II processors, 2010. URL `http://www.amd.com/us/products/desktop/processors/phenom-ii/Pages/phenom-ii.aspx`.

[18] ARM. Cortex A15 processor, 2010. URL `http://www.arm.com/products/processors/cortex-a/cortex-a15.php`.

[19] M. Armstrong, D. Flynn, M. Hammond, S. Jolly, and R. Salmon. High frame-rate television. Technical report, BBC, 2009.

[20] K. Asanovic, J. Beck, B. Irissou, B. Kingsbury, N. Morgan, and J. Wawrzynek. The T0 Vector Microprocessor. In *Hot Chips VII*, pages 187–196, Aug. 1995.

[21] Krste Asanovic, Ras Bodik, Bryan Christopher Catanzaro, Joseph James Gebis, Parry Husbands, Kurt Keutzer, David A. Patterson, William Lester Plishker, John Shalf, Samuel Webb Williams, and Katherine A. Yelick. The landscape of parallel computing research: A view from berkeley. Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, Dec 2006.

[22] A. Azevedo, C.H. Meenderinck, B.H.H. Juurlink, M. Alvarez, and A. Ramirez. Analysis of Video Filtering on the Cell Processor. In *Proceedings of International Symposium on Circuits and Systems (ISCAS)*, pages 488–491, May 2008.

[23] Arnaldo Azevedo, Ben Juurlink, Cor Meenderinck, Andrei Terechko, Jan Hoogerbrugge, Mauricio Alvarez, Alex Ramirez, and Mateo Valero. A highly scalable parallel implementation of h.264. *Transactions on High-Performance Embedded Architectures and Compilers*, 4(2), 2009.

[24] Arnaldo Azevedo, Cor Meenderinck, Ben Juurlink, Andrei Terechko, Jan Hoogerbrugge, Mauricio Alvarez, and Alex Rammirez. Parallel H.264 Decoding on an Embedded Multicore Processor. In *Proceedings of the 4th International Conference on High Performance and Embedded Architectures and Compilers - HIPEAC*, Jan 2009.

[25] Hyunki Baik, Kue-Hwan Sihn, Yun il Kim, Sehyun Bae, Najeong Han, and Hyo Jung Song. Analysis and parallelization of h.264 decoder on cell broadband engine architecture. In *2007 IEEE International Symposium on Signal Processing and Information Technology*, pages 791–795, 2007.

[26] Michael A. Baker, Pravin Dalale, Karam S. Chatha, and Sarma B.K. Vrudhula. A scalable parallel h.264 decoder on the cell broadband engine architecture. In *CODES+ISSS '09: Proceedings of the 7th IEEE/ACM international conference on Hardware/software codesign and system synthesis*, pages 353–362. ACM, 2009.

[27] bdti. Bdti h.264 solution certification benchmark, 2006. http://www.bdti.com.

[28] S. Bell, B. Edwards, J. Amann, R. Conlin, K. Joyce, V. Leung, J. MacKay, M. Reif, Liewei Bao, J. Brown, M. Mattina, Chyi-Chang Miao, C. Ramey, D. Wentzlaff, W. Anderson, E. Berger, N. Fairbanks, D. Khan, F. Montenegro, J. Stickney, and J. Zook. TILE64 - Processor: A 64-Core SoC with Mesh Interconnect. In *Digest of Technical Papers. IEEE International Solid-State Circuits Conference, 2008. ISSCC 2008*, pages 88–598, feb. 2008.

[29] R. Bhargava, L.K. John, B.L. Evans, and R. Radhakrishnan. Evaluating MMX technology using DSP and multimedia applications. In *Proceedings. 31st Annual ACM/IEEE International Symposium on Microarchitecture, 1998. MICRO-31.*, pages 37–46, 30 1998.

[30] V. Bhaskaran, K. Konstantinides, R.B. Lee, and J.P. Beck. Algorithmic and architectural enhancements for real-time MPEG-1 decoding on a general purpose RISC workstation. *IEEE Transactions on Circuits and Systems for Video Technology*, 5(5):380 –386, Oct 1995.

[31] A. Bilas, J. Fritts, and J.P. Singh. Real-time parallel mpeg-2 decoding in software. *Proceedings 11th International Parallel Processing Symposium*, pages 197–203, 1997.

[32] G. Blake, R.G. Dreslinski, and T. Mudge. A survey of multicore processors. *IEEE Signal Processing Magazine*, 26(6):26–37, november 2009.

[33] D. Boggs, A. Baktha, J. Hawkins, D. Marr, J. Miller, P. Roussel, R. Singhal, B. Toll, and K. S. Venkatraman. The Microarchitecture of the Intel Pentium 4 Processor on 90nm Techonology. *Intel Technology Journal*, 08(01):7–23, February 2004.

[34] S. Borkar. Design challenges of technology scaling. *IEEE Micro*, 19(4):23 –29, jul. 1999.

[35] Shekhar Borkar. Thousand core chips: a technology perspective. In *DAC '07: Proceedings of the 44th annual conference on Design automation*, pages 746–749. ACM, 2007.

[36] Madhukar Budagavi, Vivienne Sze, Mehmet Umut Demircin, Salih Dikbas, Minhua Zhou, and Anantha P. Chandrakasan. Description of video coding technology proposal by Texas Instruments Inc. Technical Report JCTVC-A101, Joint Collaborative Team on Video Coding (JCT-VC), April 2010.

[37] P. c. Tseng, Y. c. Chang, Y. w. Huang, H. c. Fang, C. t. Huang, and L. g. Chen. Advances in Hardware Architectures for Image and Video Coding - A Survey. *Proceedings of the IEEE*, 93(1):184–197, Jan 2005.

[38] McNairy Cameron and Soltis Don. Itanium 2 processor microarchitecture. *IEEE Micro*, 23:44–55, March 2003.

[39] O. Cantineau and J.-D. Legat. Efficient parallelisation of an mpeg-2 codec on a tms320c80 video processor. *International Conference on Image Processing, ICIP 98*, 3:977–980, 1998.

[40] D.A Carlson, R.W Castelino, and R.O Mueller. Multimedia extensions for a 550-MHz RISC microprocessor. *IEEE Journal of Solid-State Circuits*, 32(11):1618 – 1624, 1997.

[41] F. Casalino, G. di Cagno, and R. Luca. MPEG-4 video decoder optimization. In *IEEE International Conference on Multimedia Computing and Systems, 1999*, pages 363–368 vol.1, jul 1999.

[42] Jamil Chaoui, Ken Cyr, Jean-Pierre Giacalone, Sebastien de Gregorio, Yves Masse, and Yeshwant Muthusamy. OMAP: Enabling Multimedia Applications in Third Generation (3G) Wireless Terminals. White paper, Texas Instruments, 2000.

[43] Y.-K. Chen, X. Tian, S. Ge, and M. Girkar. Towards Efficient Multi-level Threading of H.264 Encoder on Intel Hyper-threading Architectures. In *Proceedings International Parallel and Distributed Processing Symposium*, Apr 2004.

[44] Y.-K. Chen, E. Q. Li, X. Zhou, , and S. Ge. Implementation of H.264 Encoder and Decoder on Personal Computers. *Journal of Visual Communications and Image Representations*, 2006.

[45] Y.K. Chen, E.Q. Li, X. Zhou, and S. Ge. Implementation of H. 264 Encoder and Decoder on Personal Computers. *Journal of Visual Communications and Image Representation*, 17, 2006.

[46] D. Cheresiz, B.H.H. Juurlink, S. Vassiliadis, and H. A. G. Wijshoff. Performance Scalability of Multimedia Instruction Set Extensions. In *Proceedings of Euro-Par 2002 Parallel processing*, pages 849–861, September 2002.

[47] Nagai-Man Cheung, Xiaopeng Fan, O.C. Au, and Man-Cheung Kung. Video Coding on Multicore Graphics Processors. *IEEE Signal Processing Magazine*, 27 (2):79–89, march 2010.

[48] Chi Ching Chi, Ben Juurlink, and Cor Meenderinck. Evaluation of parallel H.264 decoding strategies for the Cell Broadband Engine. In *Proceedings of the 24th ACM International Conference on Supercomputing*, pages 105–114, 2010.

[49] Chih-Da Chien, Chien-Chang Lin, Yi-Hung Shih, He-Chun Chen, Chia-Jui Huang, Cheng-Yen Yu, Chih-Liang Chen, Ching-Hwa Cheng, and Jiun-In Guo. A 252kgate/71mW Multi-Standard Multi-Channel Video Decoder for High Definition Video Applications. In *IEEE International Solid-State Circuits Conference, ISSCC*, pages 282–603, feb. 2007.

[50] Yongjin Cho, Seungkyun Kim, Jaejin Lee, and Heonshik Shin. Parallelizing the H.264 decoder on the cell BE architecture. In *Proceedings of the tenth ACM international conference on Embedded software*, pages 49–58, 2010.

[51] Jike Chong, Nadathur Rajagopalan Satish, Bryan Catanzaro, Kaushik Ravindran, and Kurt Keutzer. Efficient parallelization of h.264 decoding with macro block level scheduling. In *IEEE International Conference on Multimedia and Expo*, pages 1874–1877, July 2007.

[52] G.J. Conklin, G.S. Greenbaum, K.O. Lillevold, A.F. Lippman, and Y.A. Reznik. Video coding for streaming media delivery on the internet. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(3):269–281, mar 2001.

[53] T. M. Conte, P. K. Dubey, M. D. Jennings, R. B. Lee, A. Peleg, S. Rathnam, M. Schlansker, P. Song, and A. Wolfe. Challenges to Combining General Purpose and Multimedia Processors. *IEEE Computer*, 30(12):33–37, Dec. 1997.

[54] T.M. Conte, K.N. Menezes, P.M. Mills, and B.A. Patel. Optimization of Instruction Fetch Mechanisms for High Issue Rates. In *22nd Annual International Symposium on Computer Architecture*, pages 333–344, June 1995.

[55] J. Corbal, R. Espasa, and M. Valero. On the Efficiency of Reductions in micro-SIMD Media Extensions. In *International Conference on Parallel Architectures and Compilation Techniques (PACT'01)*, September 2001.

[56] Jesus Corbal, Roger Espasa, and Mateo Valero. Exploiting a New Level of DLP in Multimedia Applications. In *32nd international symposium on Microarchitecture*, pages 72–79, 1999.

[57] Jesus Corbal, Roger Espasa, and Mateo Valero. MOM: a matrix SIMD instruction set architecture for multimedia applications. In *Supercomputing '99: Proceedings of the 1999 ACM/IEEE conference on Supercomputing*, page 15, 1999.

[58] ADM Corp. ATI Stream SDK OpenCL Programming Guide (v1.05, 2010. URL `http://developer.amd.com/gpu/ATIStreamSDK/assets/ATI_Stream_SDK_OpenCL_Programming_Guide.pdf`.

[59] AMD Corp. ATI AvivoTM HD. Technology Brief, 2008. URL `http://ati.amd.com/technology/Avivo/pdf/ATI_Avivo_HD_tech_brief.pdf`.

[60] NVIDIA Corp. Nvidia cuda programming guide, 2010. URL `http://developer.download.nvidia.com/compute/cuda/3_2/toolkit/docs/CUDA_C_Programming_Guide.pdf`.

[61] NVIDIA Corp. Video Decode and Presentation API for Unix, 2010. URL `ftp://download.nvidia.com/XFree86/vdpau/doxygen/html/index.html`.

[62] Intel Corporation. Microprocessor quick reference guide, 2008. URL `http://www.intel.com/pressroom/kits/quickrefyr.htm`.

[63] Intel Corporation. Intel Advanced Vector Extensions Programming Reference, 2010. URL `http://software.intel.com/en-us/avx/`.

[64] Intel Corporation. Intel core i7-970 processor, 2010. URL `http://ark.intel.com/Product.aspx?id=47933`.

[65] Intel Corporation. Block-Matching In Motion Estimation Algorithms Using Streaming SIMD Extensions 3. Application note, Intel Corporation, 2003.

[66] A. Dasu and S. Panchanathan. A Survey of Media Processing Approaches. *IEEE Transactions on Circuits and Systems for Video Technology*, 12(8):633–645, Aug 2002.

[67] Marc Davis. Garage cinema and the future of media technology. *Communications of the ACM*, 40(2):42–48, 1997.

[68] I. Defee. Software decoding of HDTV. *IEEE Transactions on Consumer Electronics*, 45(4):1277–1283, nov 1999.

[69] E. B. Van der Tol, E. G. T. Jaspers, and R. H. Gelderblom. Mapping of h.264 decoding on a multiprocessor architecture. In *Proceedings of SPIE*, 2003.

[70] K. Diefendorff and P.K. Dubey. How Multimedia Workloads Will Change Processor Design. *IEEE Micro*, 30(9):43–45, Sept 1997.

[71] K. Diefendorff, P.K. Dubey, R. Hochsprung, and H. Scales. AltiVec Extension to PowerPC Accelerates Media Processing. *IEEE Micro*, 20(2):85–95, April 2000.

[72] Alexandre E. Eichenberger, Peng Wu, and Kevin O'Brien. Vectorization for SIMD Architectures with Alignment Constraints. In *PLDI '04: Proceedings of the ACM SIGPLAN 2004 conference on Programming language design and implementation*, pages 82–93, June 2004.

[73] B. Erol, F. Kossentini, and H. Alnuweiri. Efficient coding and mapping algorithms for software-only real-time video coding at low bit rates. *IEEE Transactions on Circuits and Systems for Video Technology*, 10(6):843–856, sep 2000.

[74] Roger Espasa. Jinks: A parametrizable simulator for vector architectures. Technical Report UPC-CEPBA-1995-31, Universitat Politècnica de Catalunya, 1995.

[75] Roger Espasa, Mateo Valero, and James E. Smith. Vector architectures: past, present and future. In *ICS '98: Proceedings of the 12th international conference on Supercomputing*, pages 425–432, 1998.

[76] ffmpeg. FFmpeg Multimedia System., 2005. http://ffmpeg.mplayerhq.hu/.

[77] D.F. Finchelstein, V. Sze, and A.P. Chandrakasan. Multicore Processing and Efficient On-Chip Caching for H.264 and Future Video Decoders. *IEEE Transactions on Circuits and Systems for Video Technology*, 19(11):1704–1713, nov. 2009.

[78] Joseph A. Fisher. Very Long Instruction Word architectures and the ELI-512. In *ISCA '83: Proceedings of the 10th annual international symposium on Computer architecture*, pages 140–150, 1983.

[79] M. Flierl and B. Girod. Generalized B pictures and the draft H.264/AVC video-compression standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):587–597, July 2003.

[80] M.J. Flynn. Very high-speed computing systems. *Proceedings of the IEEE*, 54 (12):1901–1909, dec. 1966.

[81] International Technology Roadmap for Semiconductors. International technology roadmap for semiconductors 2009 update system drivers, 2009. URL `http://www.itrs.net/Links/2009ITRS/2009Chapters_2009Tables/2009_SysDrivers.pdf`.

[82] Freedesktop.org. VAAPI (Video Acceleration API), 2007. URL `http://www.freedesktop.org/wiki/Software/vaapi`.

[83] J. Fridman. Data Alignment for Sub-word Parallelism in DSP. In *IEEE Workshop on Signal Processing Systems SiPS 99*, pages pages 251–260, Oct 1999.

[84] J. Fritts, W. Wolf, and B. Liu. Understanding Multimedia Application Characteristics for Designing Programmable Media Processors. In *SPIE Photonics West, Media Processors '99*, pages 2–13, 1999.

[85] Jason E. Fritts, Frederick W. Steiling, and Joseph A. Tucek. MediaBench II Video: Expediting the Next Generation of Video Systems Research. In *Proceedings of SPIE. Embedded Processors for Multimedia and Communications II*, pages 79–93, 2005.

[86] G. Gerosa, S. Curtis, M. D'Addeo, Bo Jiang, B. Kuttanna, F. Merchant, B. Patel, M. Taufique, and H. Samarchi. A Sub-1W to 2W Low-Power IA Processor for Mobile Internet Devices and Ultra-Mobile PCs in 45nm Hi-k; Metal Gate CMOS. In *IEEE International Solid-State Circuits Conference, 2008. ISSCC 2008*, pages 256–611, feb. 2008.

[87] Khronos Group. OpenCL Specification, Version: 1.1, 2010. URL `http://www.khronos.org/registry/cl/specs/opencl-1.1.pdf`.

[88] Michael Gschwind, H. Peter Hofstee, Brian Flachs, Martin Hopkins, Yukio Watanabe, and Takeshi Yamazaki. Synergistic Processing in Cell's Multicore Architecture. *IEEE Micro*, 26(2):10–24, 2006.

[89] Amit Gulati and George Campbell. Efficient mapping of the h.264 encoding algorithm onto multiprocessor dsps. In *Proc. Embedded Processors for Multimedia and Communications II*, volume 5683, pages 94–103, March 2005.

[90] H.261. H.261 : Video codec for audiovisual services at p x 384 kbit/s - Recommendation H.261 (11/88), 1988.

[91] H.261. H.263 : Video coding for low bit rate communication, 1995.

[92] h264. ISO/IEC 14496-10 and ITU-T Rec H.264, Advanced Video Coding, 2003.

[93] John L. Henning. Spec cpu2006 benchmark descriptions. *SIGARCH Comput. Archit. News*, 34(4):1–17, 2006.

[94] M. Holliman and Y.-K. Chen. MPEG Decoding Workload Characterization. In *Proceedings of Workshop on Computer Architecture Evaluation Using Commercial Workloads*, Feb 2003.

[95] Jan Hoogerbrugge and Andrei Terechko. A Multithreaded Multicore System for Embedded Media Processing. *Transactions on High-Performance Embedded Architectures and Compilers*, 3(2):168–187, June 2008.

[96] M. Horowitz, A. Joch, and F. Kossentini. H.264/AVC Baseline Profile Decoder Complexity Analyis. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):704–716, July 2003.

[97] Y. Hu, A. Simpson, K. McAdoo, and J. Cush. A high definition H.264/AVC hardware video decoder core for multimedia SoC's. In *IEEE International Symposium on Consumer Electronics*, pages 385–389, sept. 2004.

[98] D. A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, January 1952.

[99] Christopher J. Hughes, Praful Kaul, Sarita V. Adve, Rohit Jain, Chanik Park, and Jayanth Srinivasan. Variability in the Execution of Multimedia Applications and Implications for Architecture. In *Proceedings of the 28th Annual International Symposium on Computer Architecture*, pages 254–265, 2001.

[100] Guan Hui and Wang Hongpeng. Research of parallel decoding algrithm in h.264 on tile64. In *2nd IEEE International Conference on Broadband Network Multimedia Technology, 2009. IC-BNMT '09.*, pages 500 –503, oct. 2009.

[101] IBM. Power ISA Version 2.06. Book I: Power ISA User Instruction Set Architecture. User's manual, IBM Corp., 2009.

[102] IBM. IBM PowerPC 970FX RISC Microprocessor User's Manual. User's manual v1.6, IBM Corp., Feb 2006.

[103] IBM. PowerPC Microprocessor Family: Vector/SIMD Multimedia Extension Technology Programming Environments Manual. User's manual, IBM Corp., 2005.

[104] Ismail Khalil Ibrahim, editor. *Handbook of research on mobile multimedia.* Information Science Reference, 2 edition, 2009.

[105] IEEE. IEEE Standard for Information Technology - Portable Operating System Interfaces (POSIX) - Part 1: System Application Program Interface (API) - Amendment 1: Realtime Extension [C language]. *IEEE Std 1003.1b-1993*, 1994.

[106] IEEE. IEEE Std. 1003.1c-1995. Threads extension, The Open Group Base Specifications Issue 6, section 2.9. *IEEE Std*, 1995.

[107] M. Ikekawa, D. lshii, E. Murata, K. Numata, Y. Takamizawa, and M. Tanaka. A Real-time Software MPEG-2 Decoder For Multimedia PCs. In *Consumer Electronics, 1997. Digest of Technical Papers. ICCE., International Conference on*, pages 2–3, 11-13 1997.

[108] Apple Inc. Performance and Debugging. Tools Overview, 2005.

[109] Texas Instruments. OMAP 4 mobile applications platform, 2010. URL `http://www.ti.com/lit/swpt034`.

[110] ITU-T and ISO. Joint collaborative team on video coding (jct-vc), 2010. URL `http://www.itu.int/en/ITU-T/studygroups/com16/video/Pages/jctvc.aspx`.

[111] V. Iverson, J. McVeigh, and B. Reese. Real-time H.24-AVC codec on Intel architectures. In *International Conference on Image Processing, 2004. ICIP '04*, volume 2, pages 757–760 Vol.2, 24-27 2004.

[112] T.R. Jacobs, V.A. Chouliaras, and D.J. Mulvaney. Thread-parallel mpeg-2, mpeg-4 and h.264 video encoders for soc multi-processor architectures. *IEEE Transactions on Consumer Electronics*, 52(1):269–275, Feb. 2006.

[113] Yahya Jan and Lech Jozwiak. CABAC Accelerator Architectures for Video Compression in Future Multimedia: A Survey. In *SAMOS '09: Proceedings of the 9th International Workshop on Embedded Computer Systems: Architectures, Modeling, and Simulation*, pages 24–35, Berlin, Heidelberg, 2009. Springer-Verlag.

[114] jm. H.264/AVC Software Coordination, 2005. http://iphome.hhi.de/suehring/tml/.

[115] Ben Juurlink, Stamatis Vassiliadis, Dmitri Tcheressiz, and Harry A.G. Wijshoff. Implementation and Evaluation of the Complex Streamed Instruction Set. In *International Conference on Parallel Architectures and Compilation Techniques*, pages 73–82, September 2001.

[116] J. A. Kahle, M. N. Day, H. P. Hofstee, C. R. Johns, T. R. Maeurer, and D. Shippy. Introduction to the Cell Multiprocessor. *IBM Journal of Research and Development*, 49(4/5):589, 2005.

[117] R. Kalla, Balaram Sinharoy, and J.M. Tendler. IBM Power5 chip: a dual-core multithreaded processor. *IEEE Micro*, 24(2):40–47, mar. 2004.

[118] R. Kalla, B. Sinharoy, W.J. Starke, and M. Floyd. Power7: IBM's Next-Generation Server Processor. *IEEE Micro*, 30(2):7–15, mar. 2010.

[119] Chung-Hyo Kim and In-Cheol Park. High speed decoding of context-based adaptive binary arithmetic codes using most probable symbol prediction. In *Proceedings, 2006 IEEE International Symposium on Circuits and Systems, ISCAS 2006.*, page 4 pp., 2006.

[120] M. Kimura, K. Iwata, S. Mochizuki, H. Ueda, M. Ehama, and H. Watanabe. A Full HD Multistandard Video Codec for Mobile Applications. *Micro, IEEE*, 29 (6):18–27, nov.-dec. 2009.

[121] David Kirk and Wen mei Hwu. *Programming Massively Parallel Processors: A Hands-on Approach*. Morgan Kaufmann, 2010.

[122] Michael Kistler, Michael Perrone, and Fabrizio Petrini. Cell Multiprocessor Communication Network: Built for Speed. *IEEE Micro*, 26(3):10–23, 2006.

[123] Rob Koenen. Mpeg4, multimedia for our time. *IEEE Spectrum*, 30(9):26–34, Feb 1999.

[124] P. Kollig, C. Osborne, and T. Henriksson. Heterogeneous multi-core platform for consumer multimedia applications. In *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, pages 1254–1259, april 2009.

[125] P. Kongetira, K. Aingaran, and K. Olukotun. Niagara: a 32-way multithreaded Sparc processor. *IEEE Micro*, 25(2):21–29, mar. 2005.

[126] Matthias Korch and Thomas Rauber. A comparison of task pools for dynamic load balancing of irregular algorithms. *Concurr. Comput. : Pract. Exper.*, 16(1): 1–47, 2003. ISSN 1532-0626.

[127] Adi Kouadio. Hdtv services, trends and implementations. Technical report, ABU Digital Broadcasting Symposium 2009, Kuala Lumpur, Malaysia, 2009.

[128] C. E. Kozyrakis and D. A. Patterson. A New Direction for Computer Architecture Research. *IEEE Computer*, 31(11):24–32, Nov 1998.

[129] C.E Kozyrakis and D.A. Patterson. Scalable Vector Processors for Embedded Systems. *IEEE Micro*, 23(6):36–45, Nov–Dec 2003.

[130] Andreas Krall and Sylvain Lelait. Compilation Techniques for Multimedia Processors. *International Journal of Parallel Programming*, 28(4):347–361, Aug 2000.

[131] I. Kuroda and T. Nishitani. Multimedia Processors. *Proceedings of the IEEE*, 86 (6):1203–1221, June 1998.

[132] V. Lappalainen, T.D. Hamalainen, and P. Liuha. Overview of research efforts on media ISA extensions and their usage in video coding. *IEEE Transactions on Circuits and Systems for Video Technology*, 12(8):660–670, aug 2002.

[133] V. Lappalainen, A. Hallapuro, and T. D. Hamalainen. Complexity of Optimized H.26L Video Decoder Implementation. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):717–725, July 2003.

[134] S. Larsen and S. Amarasinghe. Exploiting Superword Level Parallelism With Multimedia Instruction Sets. In *Proceedings of the SIGPLAN '00 Conference on Programming Language Design and Implementation*, June 2000.

[135] H. Q. Le, W. J. Starke, J. S. Fields, F. P. O'Connell, D. Q. Nguyen, B. J. Ronchetti, W. M. Sauer, E. M. Schwarz, and M. T. Vaden. IBM POWER6 microarchitecture. *IBM Journal of Research and Development*, 51(6):639–662, nov. 2007.

[136] Chunho Lee, Miodrag Potkonjak, and William H. Mangione-Smith. MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communicatons Systems. In *30th International Symposium on Microarchitecture*, pages 330–335, 1997.

[137] C.L. Lee, Cheng S. Ho, Shwu-Fang Tsai, Ching-Fu Wu, Jui-Ying Cheng, Li-Wei Wang, and C. Wang. Implementation of digital hdtv video decoder by multiple multimedia video processors. *International Conference on Consumer Electronics, 1996*, pages 98–, 1996.

[138] E.A. Lee and D.G. Messerschmitt. Synchronous data flow. *Proceedings of the IEEE*, 75(9):1235–1245, sept. 1987.

[139] J. Lee, S. Moon, and W. Sung. H.264 Decoder Optimization Exploiting SIMD Instructions. In *Asia-Pacific Conference on Circuits and Systems*, Dec. 2004.

[140] R. B. Lee. Accelerating Multimedia with Enhanced Microprocessors. *IEEE Computer*, 15(2):22–32, April 1995.

[141] R. B. Lee. Subword Parallelism with MAX-2. *IEEE Micro*, 16(4):51–59, Aug. 1996.

[142] R. B. Lee and M. D. Smith. Media Processing: a New Design Target. *IEEE Micro*, 16(4):43–45, Aug. 1996.

[143] R.B. Lee. Realtime MPEG video via software decompression on a PA-RISC processor. In *Compcon '95.'Technologies for the Information Superhighway'*, pages 186–192, 1995.

[144] Ruby Lee and Larry Mcmahan. Mapping of Application Software to the Multimedia Instructions of GeneralPurpose Microprocessors. In *Proceedings of Multimedia Hardware Architectures 1997, SPIE Symposium on Electronic Imaging: Science and Technology*, pages 122–133, Feb 1997.

[145] O. Lehtoranta, T. Hamalainen, and J. Saarinen. Parallel implementation of h.263 encoder for cif-sized images on quad dsp system. *The 2001 IEEE International Symposium on Circuits and Systems, ISCAS 2001*, 2:209–212 vol. 2, 6-9 May 2001.

[146] F. E. Levine and C. P. Roth. A programmer's vew of performance monitoring in the powerpc microprocessor. *IBM Journal of Research and Development*, 41(3): 345, 1997.

[147] Markus Levy. Evaluating Digital Entertainment System Performance. *IEEE Computer*, 38(7):68–72, 2005.

[148] Heng Liao and Andrew Wolfe. Available Parallelism in Video Applications. In *International Symposium on Microarchitecture*, pages 321–329, 1997.

[149] libmpeg2. Libmpeg2. A Free MPEG-2 Video Stream Decoder, 2005. http://libmpeg2.sourceforge.net/.

[150] C.-C. Lin, J.-W. Chen, H.-C. Chang, Y.-C. Yang, Y.-H. O. Yang, M.-C. Tsai, J.-I. Guo, and J.-S. Wang. A 160K Gates/4.5 KB SRAM H.264 Video Decoder for HDTV Applications. *IEEE Journal of Solid-State Circuits*, 42(1):170–182, jan. 2007.

[151] W. Lin, K.H. Goh, B.J. Tye, G.A. Powell, T. Ohya, and S. Adachi. Real time h.263 video codec using parallel dsp. *International Conference on Image Processing*, 2: 586–589, 1997.

[152] P. List, A. Joch, J. Lainema, G. Bjntegaard, and M. Karczewicz. Adaptive Deblocking Filter. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):614–619, July 2003.

[153] T.-M. Liu, T.-A. Lin, S.-Z. Wang, W.-P. Lee, J.-Y. Yang, K.-C. Hou, and C.-Y. Lee. A 125 uW , Fully Scalable MPEG-2 and H.264/AVC Video Decoder for Mobile Applications. *IEEE Journal of Solid-State Circuits*, 42(1):161 –169, jan. 2007.

[154] T. Lyon, E. Delano, C. McNairy, and D. Mulla. Data cache design considerations for the itanium 2 processor. In *IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pages 11–20, 2002.

[155] H.S. Malvar, A. Hallapuro, M. Karczewicz, and L. Kerofsky. Low-complexity Transform and Quantization in H.264/AVC. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):598–603, July 2003.

[156] D. Marpe, H. Schwarz, and T. Wiegand. Context-based adaptive binary arithmetic coding in the H.264/AVC video compression standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):620–636, July 2003.

[157] Ketan Mayer-Patel, Brian C. Smith, and Lawrence A. Rowe. The Berkeley Software MPEG-1 Video Decoder. In *MULTIMEDIA '93: Proceedings of the first ACM international Conference on Multimedia*, pages 75–82, Oct 1993.

[158] Ketan Mayer-Patel, Brian C. Smith, and Lawrence A. Rowe. The berkeley software mpeg-1 video decoder. *ACM Trans. Multimedia Comput. Commun. Appl.*, 1(1): 110–125, 2005.

[159] Cor Meenderinck, Arnaldo Azevedo, Mauricio Alvarez, Ben Juurlink, and Alex Ramirez. Parallel Scalability of H.264. In *Workshop on Programmability Issues for Multi-Core Computers (MULTIPROG)*, Jan. 2008.

[160] Cor Meenderinck, Arnaldo Azevedo, Mauricio Alvarez, Ben Juurlink, and Alex Ramirez. Parallel Scalability of Video Decoders. *Journal of Signal Processing Systems*, 57:173–194, November 2009.

[161] P. Merkle, K. Müller, and T. Wiegand. 3D video: acquisition, coding, and display. *IEEE Transactions on Consumer Electronics*, 56(2):946–950, may 2010.

[162] Gordon Moore. Cramming more components onto integrated circuits. *Electronics Magazine*, 1965.

[163] T. Moriyoshi, H. Shinohara, T. Miyazaki, and I. Kuroda. Real-time software video codec with a fast adaptive motion vector search. In *IEEE Workshop on Signal Processing Systems, 1999. SiPS 99*, pages 44–53, 1999.

[164] Z.J.A. Mou, D.S. Rice, and Wei Ding. VIS-based native video processing on UltraSPARC. In *Image Processing, 1996. Proceedings., International Conference on*, volume 1, pages 153–156 vol.2, 16-19 1996.

[165] M. Moudgill, J-D. Wellman, and J. Moreno. Environment for PowerPC Microarchitecture Exploration. *IEEE Micro*, 19(3):15–25, May-Jun 1999.

[166] mpeg1. ISO/IEC 11172, Information Technology: Coding of Motion Pictures and Associated Audio for Digital Storage Media at up to 1.5 Mbps, 1993.

[167] mpeg2. ISO/IEC 13818, Information Technology: Generic Coding of Motion Pictures and Associated Audio Information, 1995.

[168] mpeg2. ISO/IEC 14496-2. Information Technology: Coding of Audio-visual Objects – Part 2: Visual, 2001.

[169] mssg. MSSG: MPEG Software Simulation Group, 1994. http://www.mpeg.org/MPEG/MSSG/.

[170] U.G. Nawathe, M. Hassan, K.C. Yen, A. Kumar, A. Ramachandran, and D. Greenhill. Implementation of an 8-Core, 64-Thread, Power-Efficient SPARC Server on a Chip. *IEEE Journal of Solid-State Circuits*, 43(1):6–20, jan. 2008.

[171] Huy Nguyen and Lizy Kurian John. Exploiting SIMD Parallelism in DSP and Multimedia Algorithms Using the AltiVec Technology. In *International Conference on Supercomputing*, pages 11–20, 1999.

[172] T.P. Nguyen, A. Zakhor, and K. Yelick. Performance analysis of an H.263 video encoder for VIRAM. In *2000 International Conference on Image Processing*, pages 98–101, 2000.

[173] K. Nishihara, A. Hatabu, and T. Moriyoshi. Parallelization of H.264 video decoder for embedded multicore processor. In *2008 IEEE International Conference on Multimedia and Expo*, pages 329–332, 2008.

[174] Dorit Nuzman and Richard Henderson. Multi-platform auto-vectorization. In *International Symposium on Code Generation and Optimization. CGO'06*, pages 281–294, March 2006.

[175] S. Oberman, G. Favor, and F. Weber. AMD 3DNow! technology: architecture and implementations. *IEEE Micro*, 19(2):37–48, mar/apr 1999.

[176] H. Oehring, U. Sigmund, and T. Ungerer. Mpeg-2 video decompression on simultaneous multithreaded multimedia processors. *International Conference on Parallel Architectures and Compilation Techniques, 1999.*, pages 11–16, 1999.

[177] R.R. Osorio and J.D. Bruguera. An FPGA architecture for CABAC decoding in manycore systems. In *International Conference on Application-Specific Systems, Architectures and Processors, 2008. ASAP 2008*, pages 293–298, July 2008.

[178] J. Ostermann, J. Bormans, P. List, D. Marpe, M. Narroschke, F. Pereira, T. Stockhammer, and T. Wedi. Video Coding with H.264/AVC: Tools, Performance, and Complexity. *IEEE Circuits and Systems Magazine*, 4(1):7–28, Jan 2004.

[179] M. Paganini. Nomadik: A mobile multimedia application processor platform. In *Design Automation Conference, 2007. ASP-DAC '07. Asia and South Pacific*, pages 749–750, jan. 2007.

[180] A. Patterson and J. Hennessy. *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann Publishers, San Francisco, USA, 1996.

[181] Alex Peleg and Uri Weiser. MMX Technology Extension to the Intel Architecture. *IEEE Micro*, 16(4):42–50, Aug. 1996.

[182] F. Pereira and I. Burnett. Universal multimedia experiences for tomorrow. *IEEE Signal Processing Magazine*, 20(2):63–73, Mar 2003.

[183] D.C. Pham, T. Aipperspach, D. Boerstler, M. Bolliger, R. Chaudhry, D. Cox, P. Harvey, P.M. Harvey, H.P. Hofstee, C. Johns, J. Kahle, A. Kameyama, J. Keaty, Y. Masubuchi, M. Pham, J. Pille, S. Posluszny, M. Riley, D.L. Stasiak, M. Suzuoki, O. Takahashi, J. Warnock, S. Weitzel, D. Wendel, and K. Yazawa. Overview of the architecture, circuit design, and physical implementation of a first-generation cell processor. *IEEE Journal of Solid-State Circuits*, 41(1):179–196, jan. 2006.

[184] Bart Pieters, Dieter Van Rijsselbergen, Wesley De Neve, and Rik Van de Walle. Performance evaluation of H.264/AVC decoding and visualization using the GPU. In *Applications of Digital Image Processing XXX*, page 669606, 2007.

[185] V. Pillet, J. Labarta, T. Cortes, and S. Girona. Paraver: A tool to visualize and analyze parallel code. In Patrick Nixon, editor, *Proceedings of WoTUG-18: Transputer and occam Developments*, pages 17–31, mar 1995. ISBN 90 5199 222 X.

[186] P. Pirsch and H.-J. Stolberg. VLSI Implementations of Image and Video Multimedia Processing Systems. *IEEE Transactions on Circuits and Systems for Video Technology*, 8(7):878–891, Nov 1998.

[187] P. Pirsch, N. Demassieux, and W. Gehrke. VLSI Architectures for Video Compression - A Survey. *Proceedings of the IEEE*, 83(2):220–246, Feb 1995.

[188] P. Pirsch1, C. Reuter1, J. P. Wittenburg1, M. B. Kulaczewski1, and H.-J. Stolberg1. Architecture Concepts for Multimedia Signal Processing. *The Journal of VLSI Signal Processing*, 29(3):157–165, Nov 2001.

[189] Charles Poynton. *Digital Video and HDTV. Algorithms and Interfaces*. Morgan Kaufmann, 2003.

[190] Francisca Quintana, Jesus Corbal, Roger Espasa, and Mateo Valero. Adding a Vector Unit on a Superscalar Processor. In *International Conference on Supercomputing*, pages 1–10, June 1999.

[191] S. K. Raman, V. Pentkovski, and J. Keshav. Implementing Streaming SIMD Extensions on the Pentium III Processor. *IEEE Micro*, 20(4):47–57, Aug. 2000.

[192] A. Ramirez, F. Cabarcas, B. Juurlink, M. Alvarez Mesa, A. Azevedo, C. Meenderinck, G. Gaydadjiev, C. Ciobanu, S. Isaza, and F. Sanchez. The SARC Architecture. *IEEE Micro*, 30(5):16–29, Sept/Oct. 2010.

[193] Parthasarathy Ranganathan, Sarita V. Adve, and Norman P. Jouppi. Performance of Image and Video Processing with General-Purpose Processors and Media ISA Extensions. In *International Symposium on Computer Architecture*, pages 124–135, 1999.

[194] S. Rathnam and G. Slavenburg. An architectural overview of the programmable multimedia processor, TM-1. In *Compcon '96. 'Technologies for the Information Superhighway' Digest of Papers*, pages 319–326, 25-28 1996.

[195] Iain E. G. Richardson. *H.264 and MPEG-4. Video Compression for Next-generation Multimedia*. Wiley, Chichester, England, 2004.

[196] Iain E. G. Richardson. *Video Codec Design: Developing Image and Video Compression Systems*. John Wiley and Sons, 2002.

[197] Alejandro Rico, Felipe Cabarcas, Antonio Quesada, Milan Pavlovic, Augusto Javier Vega, Carlos Villavieja, Yoav Etsion, and Alex Ramírez. Scalable Simulation of Decoupled Accelerator Architectures. Technical Report UPC-DAC-RR-2010-14, Universitat Politècnica de Catalunya (UPC), 2010.

[198] Scott Rixner, William J. Dally, Brucek Khailany, Peter Mattson, Ujval J. Kapasi, , and John D. Owens. Register organization for media processing. In *Tenth International Symposium on High Performance Computer Architecture*, January 2000.

[199] A. Rodriguez, A. Gonzalez, and M. P. Malumbres. Hierarchical Parallelization of an H.264/AVC Video Encoder. In *PARELEC '06: Proceedings of the international symposium on Parallel Computing in Electrical Engineering*, pages 363–368, 2006.

[200] N.J. Rohrer, M. Canada, E. Cohen, M. Ringler, M. Mayfield, P. Sandon, P. Kartschoke, J. Heaslip, J. Allen, P. McCormick, T. Pfluger, J. Zimmerman, C. Lichtenau, T. Werner, G. Salem, M. Ross, D. Appenzeller, and D. Thygesen. Powerpc 970 in 130 nm and 90 nm technologies. In *2004 IEEE International Solid-State Circuits Conference, 2004. Digest of Technical Papers. ISSCC*, pages 68–69 Vol.1, feb. 2004.

[201] M. Roitzsch. Slice-Balancing H.264 Video Encoding for Improved Scalability of Multicore Decoding. In *Work-in-Progress Proceedings of the 27th IEEE Real-Time Systems Symposium (RTSS)*, 2006.

[202] Eric Rotenberg, Jim Smith, and Steve Bennett. Trace Cache: a Low Latency Approach to High Bandwidth Instruction Fetching. In *29th Annual International Symposium on Computer Architecture*, page 24, 1996.

[203] S. Larsen and E. Witchel and S. Amarasinghe. Techniques for Increasing and Detecting Memory Alignment. Research Report MIT-LCS-TM-621, MIT Laboratory for Computer Science, Nov. 2001.

[204] Amir Said. Introduction to Arithmetic Coding - Theory and Practice. Technical Report HPL-2004-76, HP Laboratories Palo Alto, 2004.

[205] Amir Said. Comparative Analysis of Arithmetic Coding Computational Complexity. Technical Report HPL-2004-75, HP Laboratories Palo Alto, 2004.

[206] E. Salamí, J. Corbal, R. Espasa, and M. Valero. An Evaluation of Different DLP alternatives for the Embedded Media Domain. In *1st Workshop on Media Processors and DSPs*, Nov. 1999.

[207] Daniel Sanchez, Richard Yoo, and Christos Kozyrakis. Flexible Architectural Support for Fine-grain Scheduling. In *Fifteenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2010)*, pages 311–322, 2010.

[208] Kumar Sanjeev, Hughes Christopher J., and Nguyen Anthony. Carbon: architectural support for fine-grained parallelism on chip multiprocessors. In *Proceedings of the 34th annual international symposium on Computer architecture*, pages 162–173, 2007.

[209] Klaus Schoffmann, Markus Fauster, Oliver Lampl, and Laszlo Böszörmeny. An Evaluation of Parallelization Concepts for Baseline-Profile Compliant H.264/AVC Decoders. In *Lecture Notes in Computer Science. Euro-Par 2007 Parallel Processing*, August 2007.

[210] H. Schwarz, D. Marpe, and T. Wiegand. Overview of the Scalable Video Coding Extension of the H.264/AVC Standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 17(9):1103–1120, sept. 2007.

[211] L. Seiler, D. Carmean, E. Sprangle, T. Forsyth, P. Dubey, S. Junkins, A. Lake, R. Cavin, R. Espasa, E. Grochowski, T. Juan, M. Abrash, J. Sugerman, and P. Hanrahan. Larrabee: A many-core x86 architecture for visual computing. *IEEE Micro*, 29(1):10–21, jan. 2009.

[212] Florian H. Seitner, Ralf M. Schreier, Michael Bleyer, and Margrit Gelautz. Evaluation of data-parallel splitting approaches for H.264 decoding. In *Proceedings of the 6th International Conference on Advances in Mobile Computing and Multimedia*, pages 40–49, 2008.

[213] Florian H. Seitner, Michael Bleyer, Margrit Gelautz, and Ralf M. Beuschel. Development of a high-level simulation approach and its application to multicore video decoding. *IEEE Trans. Cir. and Sys. for Video Technol.*, 19:1667–1679, November 2009.

[214] R. Selvaggi and L. Pearlstein. Broadcom mediaDSP: A Platform for Building Programmable Multicore Video Processors. *IEEE Micro*, 29(2):30–45, march-april 2009.

[215] Freescale Semiconductor. AltiVec Technology Programming Interface Manual. User manual Altivecpim/D 6/1999, Freescale Semiconductor, 1999.

[216] N. Seshan. High VelociTI Processing [Texas Instruments VLIW Architecture]. *IEEE Signal Processing Magazine*, 15(2):86–101, Mar 1998.

[217] C. E. Shannon. A mathematical theory of communication. *Bell system technical journal*, 27:379–423, 1948.

[218] K. Shen, L. A. Rowe, and E. J. Delp. Parallel implementation of an MPEG-1 encoder: faster than real time. In *Proc. SPIE Vol. 2419, p. 407-418, Digital Video Compression: Algorithms and Technologies 1995*, volume 2419, pages 407–418, 1995.

[219] T. T. Shih, C. L. Yang, and Y. S. Tung. Workload Characterization of the H.264/AVC Decoder. In *Proceeding of the 5th Pacific Rim Conference on Multimedia*, Dec. 2004.

[220] J.L. Shin, K. Tam, D. Huang, B. Petrick, H. Pham, Changku Hwang, Hongping Li, A. Smith, T. Johnson, F. Schumacher, D. Greenhill, A.S. Leon, and A. Strong. A 40nm 16-core 128-thread CMT SPARC SoC processor. In *2010 IEEE International Solid-State Circuits Conference (ISSCC)*, pages 98–99, feb. 2010.

[221] H. Shojania, S. Sudharsanan, and Chan Wai-Yip. Performance Improvement of the H.264/AVC Deblocking Filter Using SIMD instructions. In *Proceedings of IEEE International Symposium on Circuits and Systems ISCAS*, May 2006.

[222] SIG. Mips Extension for Digital Media With 3D. Technical report, MIPS Technologies, Inc, 1997.

[223] Kue-Hwan Sihn, Hyunki Baik, Jong-Tae Kim, Sehyun Bae, and Hyo Jung Song. Novel approaches to parallel h.264 decoder on symmetric multicore systems. In *IEEE International Conference on Acoustics, Speech and Signal Processing, 2009. ICASSP 2009*, pages 2017–2020, 2009.

[224] T. Sikora. MPEG Digital Video–Coding Standards. *IEEE Signal Processing Magazine*, 14(5):82–100, 1997.

[225] T. Sikora. Trends and Perspectives in Image and Video Coding. *Proceedings of the IEEE*, 93(1):6–17, Jan 2005.

[226] N. Slingerland and A. J. Smith. Measuring the Performance of Multimedia Instruction Sets. *IEEE Transactions on Computers*, 51(11):1317–1332, Nov 2002.

[227] N. T. Slingerland and A. J. Smith. Multimedia Instruction Sets for General Pupose Microprocessors: A Survey. Technical Report CSD-00-1124, UCB, Dec. 1999.

[228] Nathan T. Slingerland and Alan Jay Smith. Cache Performance for Multimedia Applications. In *ICS '01: Proceedings of the 15th International Conference On Supercomputing*, pages 204–217, 2001.

[229] Nathan T. Slingerland and Alan Jay Smith. Design and Characterization of the Berkeley Multimedia Workload. *Multimedia Systems*, 8(4):315–327, 2002.

[230] Amitabh Srivastava and Alan Eustace. ATOM: A system for building customized program analysis tools. *ACM SIGPLAN '94 Conference on Programming Language Design and Implementation*, 29(6):196–205, June 1994.

[231] Masayuki Sugawara. Super hi-vision — research on a future ultra-hdtv system. Technical report, European Broadcasting Union, 2008.

[232] G. J. Sullivan and T. Wiegand. Video Compression–From Concepts to the H.264/AVC Standard. *Proceedings of the IEEE*, 93(1):18–31, Jan 2005.

[233] G. J. Sullivan, P. Topiwala, and A. Lutrha. The H.264/AVC Advanced Video Coding Standard: Overview and Introduction to the Fidelity Range Extensions. In *SPIE Conference on Applications of Digital Image Processings*, Aug 2004.

[234] Gary J. Sullivan and Jens-Rainer Ohmb. Recent developments in standardization of high efficiency video coding (HEVC). In *SPIE Applications of Digital Image Processing*, Sept 2010.

[235] V. Sze, D.F. Finchelstein, M.E. Sinangil, and A.P. Chandrakasan. A 0.7-V 1.8-mW H.264/AVC 720p Video Decoder. *IEEE Journal of Solid-State Circuits*, 44 (11):2943–2956, nov. 2009.

[236] Vivienne Sze and Anantha P. Chandrakasan. A high throughput CABAC algorithm using syntax element partitioning. In *Proceedings of the 16th IEEE international conference on Image processing*, pages 773–776, 2009.

[237] Friman Sánchez, Mauricio Alvarez, Esther Salamí, Alex Ramírez, and Mateo Valero. On the Scalability of 1- and 2-Dimensional SIMD Extensions for Multimedia Applications. In *ISPASS. IEEE International Symposium on Performance Analysis of Systems and Software*, pages 167–176, March 2005.

[238] D. Talla, L.K. John, V. Lapinskii, and B.L. Evans. Evaluating signal processing and multimedia applications on SIMD, VLIW and superscalar architectures. In *Proceedings of the International Conference on Computer Design*, pages 163–172, 2000.

[239] Deepu Talla, Lizy Kurian John, and Doug Burger. Bottlenecks in Multimedia Processing with SIMD Style Extensions and Architectural Eenhancements. *IEEE Transactions on Computers*, 52(8):1015–1031, Aug. 2003.

[240] A. Tamhankar and K. R. Rao. An Overview of H.264/MPEG-4 PART 10. In *4th EURASIP Conference focused on Video/Image Processing and Multimedia Communications*, pages 1–51, July 2003.

[241] Gadi Taubenfeld. *Synchronization Algorithms and Concurrent Programming.* Prentice Hall, 2006.

[242] H.H. Taylor, D. Chin, and A.W. Jessup. A mpeg encoder implementation on the princeton engine video supercomputer. *Data Compression Conference, 1993. DCC '93.*, pages 420–429, 1993.

[243] J. M. Tendler, J. S. Dodson, J. S. Fields, H. Le, and B. Sinharoy. POWER4 system microarchitecture. *IBM Journal of Research and Development*, 46(1):5–25, jan. 2002.

[244] test sequences. MPEG-Test Sequences, 2005. http://www.ldv.ei.tum.de/liquid.php?page=70.

[245] Texas Instruments. TMS320C64x/C64x+ DSP CPU and Instruction Set Reference Guide. User manual SPRU732C, Texas Instruments, 2005.

[246] S. Thakkar and T. Huff. The Internet Streaming SIMD extensions. *IEEE Computer*, 32(12):26–34, Dec. 1999.

[247] Shyamkumar Thoziyoor, Naveen Muralimanohar, and Norman P. Jouppi. Cacti 5.0. Technical Report HPL-2007-167, Advanced Architecture Laboratory HP Laboratories, 2007.

[248] M. Tremblay, J. M. O'Connor, V. Narayanan, and H. Liang. VIS Speeds New Media Processing. *IEEE Micro*, 16(4):51–59, Aug. 1996.

[249] D.M. Tullsen, J.L. Lo, S.J. Eggers, and H.M. Levy. Supporting fine-grained synchronization on a simultaneous multithreading processor. In *High-Performance Computer Architecture, 1999. Proceedings. Fifth International Symposium On*, pages 54–58, jan 1999.

[250] Yi-Shin Tung, Chia-Chiang Ho, and Ja-Ling Wu. Mmx-based dct and mc algorithms for real-time pure software mpeg decoding. In *IEEE International Conference on Multimedia Computing and Systems, 1999*, pages 357–362 vol.1, jul 1999.

[251] J.-W. van de Waerdt and S. Vassiliadis. Instruction set architecture enhancements for video processing. In *ASAP 2005. 16th IEEE International Conference on Application-Specific Systems, Architecture Processors, 2005*, pages 146–153, 23-25 2005.

[252] Jan-Willem van de Waerdt, Stamatis Vassiliadis, Sanjeev Das, Sebastian Mirolo, Chris Yen, Bill Zhong, Carlos Basto, Jean-Paul van Itegem, Dinesh Amirtharaj, Kulbhushan Kalra, Pedro Rodriguez, and Hans van Antwerpen. The TM3270 Media-Processor. In *MICRO 38: Proceedings of the 38th annual IEEE/ACM International Symposium on Microarchitecture*, pages 331–342, Nov 2005.

[253] A. Vetro, T. Wiegand, and G.J. Sullivan. Overview of the Stereo and Multiview Video Coding Extensions of the H.264/MPEG-4 AVC Standard. *Proceedings of the IEEE*, 99(4):626–642, april 2011.

[254] T. Wedi and H. G. Musmann. Motion- and Aliasing-Compensated Prediction for Hybrid Video Coding. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):577–586, July 2003.

[255] D. Wentzlaff, P. Griffin, H. Hoffmann, Liewei Bao, B. Edwards, C. Ramey, M. Mattina, Chyi-Chang Miao, J.F. Brown, and A. Agarwal. On-Chip Interconnection Architecture of the Tile Processor. *IEE Micro*, 27(5):15–31, sep. 2007.

[256] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A.Luthra. Overview of the H.264/AVC Video Coding Standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):560–576, July 2003.

[257] Ian H. Witten, Radford M. Neal, and John G. Cleary. Arithmetic coding for data compression. *Commun. ACM*, 30(6):520–540, 1987.

[258] W. Wolf. Multiprocessor system-on-chip technology. *Signal Processing Magazine, IEEE*, 26(6):50–54, november 2009.

[259] x264. X264. A Free H.264/AVC Encoder, 2006. http://developers.videolan.org/x264.html.

[260] x264. X264-devel – Mailing list for x264 developers, July-August 2007. URL `http://mailman.videolan.org/listinfo/x264-devel`. subject: out-of-range motion vectors.

[261] Z. Xu, S. Sohoni, R. Min, and Y. Hu. An Analysis of Cache Performance of Multimedia Aplications. *IEEE Transactions on Computers*, 53(1):20–38, Jan 2004.

[262] xvid. XviD. An ISO MPEG-4 Compliant Video Codec, 2005. http://www.xvid.org.

[263] Ganesh Yadav, R.K. Singh, and Vipin Chaudhary. On Implementation of MPEG-2 Like Real-Time Parallel Media Applications on MDSP SoC Cradle Architecture. In *Lecture Notes in Computer Science. Embedded and Ubiquitous Computing*, July 2004.

[264] Sudhakar Yalamanchili. Class notes: Multicore Computing Evolution, 2006.

[265] Chenggang Yan, Feng Dai, and Yongdong Zhang. Parallel deblocking filter for h.264/avc on the tilera many-core systems. In *Proceedings of the 17th international conference on Advances in multimedia modeling*, pages 51–61, 2011.

[266] K. C. Yeager. The mips r10000 superscalar microprocessor. *IEEE Micro*, 16(2), April 1996.

[267] Yongseok Yi and In-Cheol Park. High-Speed H.264/AVC CABAC Decoding. *IEEE Transactions on Circuits and Systems for Video Technology*, 17(4):490–494, april 2007.

[268] Fan Dong-rui Yuan Nan, Yu Lei. An Efficient and Flexible Task Management for Many Cores. *Transactions on High-Performance Embedded Architectures and Compilers*, 4(3), 2009.

[269] Zhuo Zhao and Ping Liang. Data partition for wavefront parallelization of H.264 video encoder. In *IEEE International Symposium on Circuits and Systems.*, 2006.

[270] Chang-Guo Zhou, Ihtisham Kabir, Leslie Kohn, Aman Jabbi, D. Rice, and Xio-Ping Hu. MPEG video decoding with the UltraSPARC visual instruction set. In *Compcon '95.'Technologies for the Information Superhighway'*, pages 470 –477, 5-9 1995.

[271] Minhua Zhou and R. Talluri. DSP-based real-time video decoding. In *Consumer Electronics, 1999. ICCE. International Conference on*, pages 296–297, 1999.

[272] X. Zhou, E. Q. Li, and Y.-K. Chen. Implementation of H.264 Decoder on General-Purpose Processors with Media Instructions. In *Proceedings of SPIE Conference on Image and Video Communications and Processing*, 2003.

[273] Ce Zhu, Xiao Lin, and Lap-Pui Chau. Hexagon-based Search Pattern for Fast Block Motion Estimation. *IEEE Transactions on Circuits and Systems for Video Technology*, 12(5):349–355, May 2002.