J Sign Process Syst
DOI 10.1007/s11265-008-0256-9

Parallel Scalability of Video Decoders

Cor Meenderinck · Arnaldo Azevedo · Ben Juurlink · Mauricio Alvarez Mesa · Alex Ramirez

Received: 14 September 2007 / Revised: 4 July 2008 / Accepted: 23 July 2008

Abstract An important question is whether emerging and future applications exhibit sufficient parallelism, in particular thread-level parallelism, to exploit the large numbers of cores future chip multiprocessors (CMPs) are expected to contain. As a case study we investigate the parallelism available in video decoders, an important application domain now and in the future. Specifically, we analyze the parallel scalability of the H.264 decoding process. First we discuss the data structures and dependencies of H.264 and show what types of parallelism it allows to be exploited. We also show that previously proposed parallelization strategies such as slice-level, frame-level, and intra-frame macroblock (MB) level parallelism, are not sufficiently scalable. Based on the observation that inter-frame dependencies have a limited spatial range we propose a new parallelization strategy, called Dynamic 3D-Wave. It allows certain MBs of consecutive frames to be decoded in parallel. Using this new strategy we analyze the limits to the available MB-level parallelism in H.264. Using real movie sequences we find a maximum MB parallelism ranging from 4000 to 7000. We also perform a case study to assess the practical value and possibilities of a highly parallelized H.264 application. The results show that H.264 exhibits sufficient parallelism to efficiently exploit the capabilities of future manycore CMPs.

Keywords H.264 · Chip multiprocessors · Scalability · Parallel processing · Video codecs

C. Meenderinck (✉) · A. Azevedo · B. Juurlink
Delft University of Technology, Delft, The Netherlands
e-mail: Cor@ce.et.tudelft.nl

A. Azevedo
e-mail: Azevedo@ce.et.tudelft.nl

B. Juurlink
e-mail: Benj@ce.et.tudelft.nl

M. Alvarez Mesa
Technical University of Catalonia (UPC), Barcelona, Spain
e-mail: alvarez@ac.upc.edu

A. Ramirez
Barcelona Supercomputing Center (BSC), Barcelona, Spain
e-mail: alex.ramirez@bsc.es

1 Introduction

We are witnessing a paradigm shift in computer architecture towards chip multiprocessors (CMPs). In the past performance has improved mainly due to higher clock frequencies and architectural approaches to exploit instruction-level parallelism (ILP), such as pipelining, multiple issue, out-of-order execution, and branch prediction. It seems, however, that these sources of performance gains are exhausted. New techniques to exploit more ILP are showing diminishing results while being costly in terms of area, power, and design time. Also clock frequency increase is flattening out, mainly because pipelining has reached its limit. As a result, industry, including IBM, Sun, Intel, and AMD, turned to CMPs.

At the same time we see that we have hit the power wall and thus performance increase can only be achieved without increasing overall power. In other words, power efficiency (performance per watt or performance per transistor) has become an important metric for evaluation. CMPs allow power efficient computation. Even if real-time performance can be achieved with few cores, using manycores improves power efficiency as, for example, voltage/frequency scaling can be applied. That is also the reason why low power architectures are the first to apply the CMP paradigm on large scale [1, 2].

It is expected that the number of cores on a CMP will double every three year [3], resulting in an estimate of 150 high performance cores on a die in 2017. For power efficiency reasons CMPs might as well consist of many simple and small cores which might count up to thousand and more [4]. A central question is whether applications scale to such large number of cores. If applications are not extensively parallelizable, cores will be left unused and performance might suffer. Also, achieving power efficiency on manycores relies on thread-level parallelism (TLP) offered by applications.

In this paper we investigate this issue for video decoding workloads by analyzing their parallel scalability. Multimedia applications remain important workloads in the future and video codecs are expected to be important benchmarks for all kind of systems, ranging from low power mobile devices to high performance systems. Specifically, we analyze the parallel scalability of an H.264 decoder by exploring the limits to the amount of TLP. This research is similar to the video applications limit study of ILP presented in [5] in 1996. Investigating the limits to TLP, however requires to set a bound to the granularity of threads. Without this bound a single instruction can be a thread, which is clearly not desirable. Thus, using emerging techniques such as light weight micro-threading [6], possibly even more TLP can be exploited. However, we show that considering down to macroblock (MB) level granularity, H.264 contains sufficient parallelism to sustain a manycore CMP.

This paper is organized as follows. In Section 2 a brief overview of the H.264 standard is provided. Next, in Section 3 we describe the benchmark we use throughout this paper. In Section 4 possible parallelization strategies are discussed. In Section 5 we analyze the parallel scalability of the best previously proposed parallelization technique. In Section 6 we propose the Dynamic 3D-Wave parallelization strategy. Using this new parallelization strategy in Section 7 we investigate the parallel scalability of H.264 and show it exhibits huge amounts of parallelism. Section 8 describes a case study to assess the practical value of a highly parallelized decoder. In Section 9 we provide an overview of related work. Section 10 concludes the paper and discusses future work.

2 Overview of the H.264 Standard

Currently, the best video coding standard, in terms of compression and quality is H.264 [7]. It is used in HD-DVD and blu-ray Disc, and many countries are using/will use it for terrestrial television broadcast, satellite broadcast, and mobile television services. It has a compression improvement of over two times compared to previous standards such as MPEG-4 ASP, H.262/MPEG-2, etc. The H.264 standard [8] was designed to serve a broad range of application domains ranging from low to high bitrates, from low to high resolutions, and a variety of networks and systems, e.g., Internet streams, mobile streams, disc storage, and broadcast. The H.264 standard was jointly developed by ITU-T Video Coding Experts Group and ISO/IEC Moving Picture Experts Group (MPEG). It is also called MPEG-4 part 10 or AVC (advanced video coding).

Figures 1 and 2 depict block diagrams of the decoding and the encoding process of H.264. The main kernels are Prediction (intra prediction or motion estimation), Discrete Cosine Transform (DCT), Quantization, Deblocking filter, and Entropy Coding. These kernels operate on MBs, which are blocks of 16×16 pixels, although the standard allows some kernels to operate on smaller blocks, down to 4×4. H.264 uses the YCbCr color space with mainly a 4:2:0 subsampling scheme.

A movie picture is called a frame and can consist of several slices or slice groups. A slice is a partition of a frame such that all comprised MBs are in scan order (from left to right, from top to bottom). The Flexible

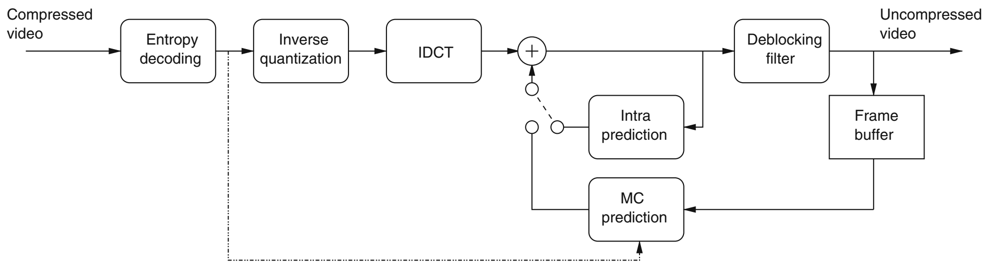

Figure 1 Block diagram of the decoding process.

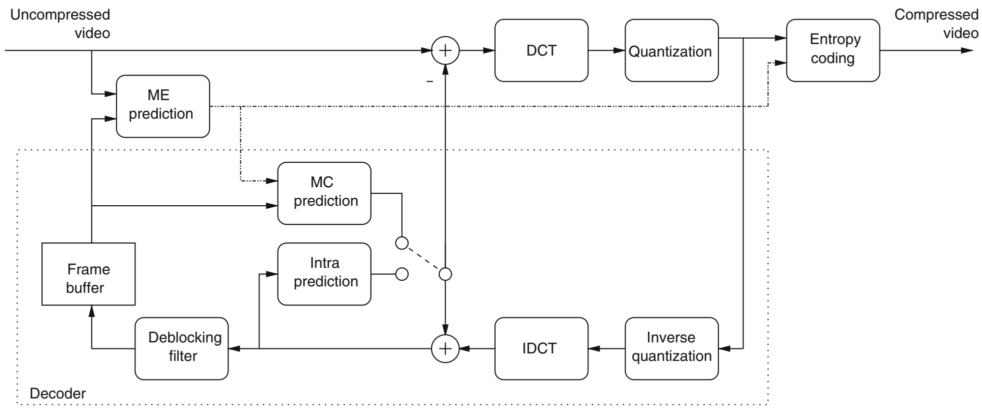

Figure 2 Block diagram of the encoding process.

Macroblock Ordering feature allows slice groups to be defined, which consist of an arbitrary set of MBs. Each slice group can be partitioned into slices in the same way a frame can. Slices are self contained and can be decoded independently.

H.264 defines three main types of slices/frames: I, P, and B-slices. An I-slice uses intra prediction and is independent of other slices. In intra prediction a MB is predicted based on adjacent blocks. A P-slice uses motion estimation and intra prediction and depends on one or more previous slices, either I, P or B. Motion estimation is used to exploit temporal correlation between slices. Finally, B-slices use bidirectional motion estimation and depend on slices from past and future [9]. Figure 3 shows a typical slice order and the dependencies, assuming each frame consist of one slice only. The standard also defines SI and SP slices that are slightly different from the ones mentioned before and which are targeted at mobile and Internet streaming applications.

The H.264 standard has many options. We briefly mention the key features and compare them to previous standards. Table 1 provides an overview of the discussed features for MPEG-2, MPEG-4 ASP, and H.264. The different columns for H.264 represent profiles which are explained later. For more details the interested reader is referred to [10, 11].

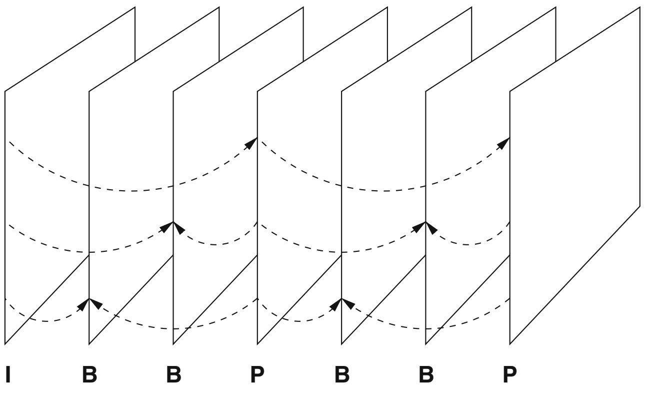

Figure 3 A typical slice/frame sequence and its dependencies.

Motion estimation Advances in motion estimation is one of the major contributors to the compression improvement of H.264. The standard allows variable block sizes ranging from 16×16 down to 4×4, and each block has its own motion vector(s). The motion vector is quarter sample accurate. Multiple reference frames can be used in a weighted fashion. This significantly improves coding occlusion areas where an accurate prediction can only be made from a frame further in the past.

Intra prediction Three types of intra coding are supported, which are denoted as Intra_4×4, Intra_8×8 and Intra_16×16. The first type uses spatial prediction on each 4×4 luminance block. Eight modes of directional prediction are available, among them horizontal, vertical, and diagonal. This mode is well suited for MBs with small details. For smooth image areas the Intra_16×16 type is more suitable, for which four prediction modes are available. The high profile of H.264 also supports intra coding on 8×8 luma blocks. Chroma components are estimated for whole MBs using one specialized prediction mode.

Discrete cosine transform MPEG-2 and MPEG-4 part 2 employed an 8×8 floating point transform. However, due to the decreased granularity of the motion estimation, there is less spatial correlation in the residual signal. Thus, standard a 4×4 (that means 2×2 for chrominance) transform is used, which is as efficient as a larger transform [12]. Moreover, a smaller block size reduces artifacts known as ringing and blocking. An optional feature of H.264 is Adaptive Block size Transform, which adapts the block size used for DCT to the size used in the motion estimation [13]. Furthermore,

Table 1 Comparison of video coding standards and profiles.

	MPEG-2	MPEG-4 ASP	H.264 BP	H.264 MP	H.264 XP	H.264 HiP
Picture types	I, P, B	I, P, B	I, P	I, P, B	I, P, B, SI, SP	I, P, B
Flexible macroblock ordering	No	No	Yes	No	Yes	No
Motion block size	16×16	16×16, 16×8, 8×8	16×16, 16×8, 8×16, 8×8, 8×4, 4×8, 4×4	16×16, 16×8, 8×16, 8×8, 8×4, 4×8, 4×4	16×16, 16×8, 8×16, 8×8, 8×4, 4×8, 4×4	16×16, 16×8, 8×16, 8×8, 8×4, 4×8, 4×4
Multiple reference frames	No	No	Yes	Yes	Yes	Yes
Motion pel accuracy	1, 1/2	1, 1/2, 1/4	1, 1/2, 1/4	1, 1/2, 1/4	1, 1/2, 1/4	1, 1/2, 1/4
Weighted prediction	No	No	No	Yes	Yes	Yes
Transform	16×16	8×8 DCT	4×4 integer DCT	4×4 integer DCT	4×4 integer DCT	4×4, 8×8 integer DCT
In-loop deblocking filter	No	No	Yes	Yes	Yes	Yes
Entropy coding	VLC	VLC	CAVLC, UVLC	CAVLC, UVLC, CABAC	CAVLC, UVLC	CAVLC, UVLC, CABAC

to prevent rounding errors that occur in floating point implementations, an integer transform was chosen.

Deblocking filter Processing a frame in MBs can produce blocking artifacts, generally considered the most visible artifact in prior standards. This effect can be resolved by applying a deblocking filter around the edges of a block. The strength of the filter is adaptable through several syntax elements [14]. While in H.263+ this feature was optional, in H.264 it is standard and it is placed within the motion compensated prediction loop (see Fig. 1) to improve the motion estimation.

Entropy coding There are two classes of entropy coding available in H.264: Variable Length Coding (VLC) and Context Adaptive Binary Arithmetic Coding (CABAC). The latter achieves up to 10% better compression but at the cost of large computational complexity [15]. The VLC class consists of Context Adaptive VLC (CAVLC) for the transform coefficients, and Universal VLC (UVLC) for the small remaining part. CAVLC achieves large improvements over simple VLC, used in prior standards, without the full computational cost of CABAC.

The standard was designed to suite a broad range of video application domains. However, each domain is expected to use only a subset of the available options. For this reason profiles and levels were specified to mark conformance points. Encoders and decoders that conform to the same profile are guaranteed to interoperate correctly. Profiles define sets of coding tools and algorithms that can be used while levels place constraints on the parameters of the bitstream.

The standard initially defined two profiles, but has since then been extended to a total of 11 profiles, including three main profiles, four high profiles, and four all-intra profiles. The three main profiles and the most important high profile are:

- **Baseline Profile (BP):** the simplest profile mainly used for video conferencing and mobile video.
- **Main Profile (MP):** intended to be used for consumer broadcast and storage applications, but overtaken by the high profile.
- **Extended Profile (XP):** intended for streaming video and includes special capabilities to improve robustness.
- **High Profile (HiP)** intended for high definition broadcast and disc storage, and is used in HD DVD and Blu-ray.

Besides HiP there are three other high profiles that support up to 14 bits per sample, 4:2:2 and 4:4:4 sampling, and other features [16]. The all-intra profiles are similar to the high profiles and are mainly used in professional camera and editing systems.

In addition 16 levels are currently defined which are used for all profiles. A level specifies, for example, the upper limit for the picture size, the decoder processing rate, the size of the multi-picture buffers, and the video bitrate. Levels have profile independent parameters as well as profile specific ones.

3 Benchmark

Throughout this paper we use the HD-VideoBench [17], which provides movie test sequences, an encoder

(X264 [18]), and a decoder (FFmpeg [19]). The benchmark contains the following test sequences:

- **rush_hour:** rush-hour in Munich city; static background, slowly moving objects.
- **riverbed:** riverbed seen through waving water; abrupt and stochastic changes.
- **pedestrian:** shot of a pedestrian area in city center; static background, fast moving objects.
- **blue_sky:** top of two trees against blue sky; static objects, sliding camera.

All movies are available in three formats: 720×576 (SD), 1280×720 (HD), 1920×1088 (FHD). Each movie has a frame rate of 25 frames per second and has a length of 100 frames. For some experiments longer sequences were required, which we created by replicating the sequences. Unless specified otherwise, for SD and HD we used sequences of 300 frames while for FHD we used sequences of 400 frames.

The benchmark provides the test sequences in raw format. Encoding is done with the X264 encoder using the following options: 2 B-frames between I and P frames, 16 reference frames, weighted prediction, hexagonal motion estimation algorithm (hex) with maximum search range 24, one slice per frame, and adaptive block size transform. Movies encoded with this set of options represent the typical case. To test the worst case scenario we also created movies with large motion vectors by encoding the movies with the exhaustive search motion estimation algorithm (esa) and a maximum search range of 512 pixels. The first movies are marked with the suffix 'hex' while for the latter we use 'esa'.

4 Parallelizing H.264

The coding efficiency gains of advanced video codecs like H.264, come at the price of increased computational requirements. The demands for computing power increases also with the shift towards high definition resolutions. As a result, current high performance uniprocessor architectures are not capable of providing the required performance for real-time processing [20–23]. Figure 4 depicts the performance of the FFmpeg H.264 decoder optimized with SIMD instructions on a superscalar processor (Intel IA32 Xeon processor at 2.4 GHz with 512KB of L2 cache) and compare it with other video codecs. For FHD resolution this high performance processor is not able to achieve real-time operation. And, the trend in future video applications is toward higher resolutions and higher quality video systems that in turn require more performance. As, with the shift towards the CMP paradigm, most hardware performance increase will come from the capability of running many threads in parallel, it is necessary to parallelize H.264 decoding.

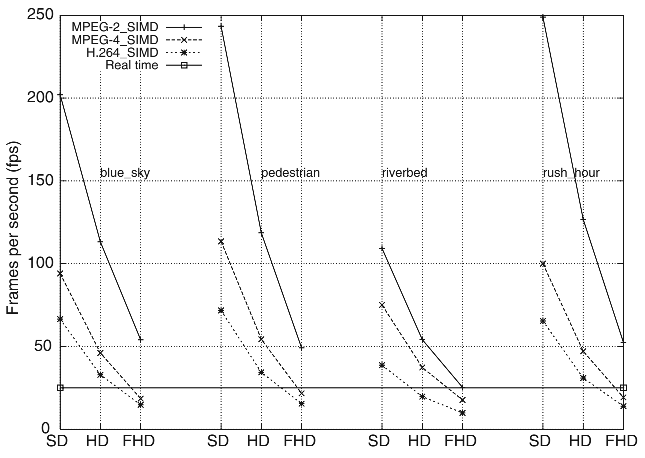

Figure 4 H.264 decoding performance.

Moreover, the power wall forces us to compute power efficient. This requires the utilization of all the cores CMPs offer, and thus applications have to be parallelizable to a great extend. In this section we analyze the possibilities for parallelization of H.264 and compare them in terms of communication, synchronization, load balancing, scalability and software optimization.

The H.264 codec can be parallelized either by a task-level or data-level decomposition. In Fig. 5 the two approaches are sketched. In task-level decomposition individual tasks of the H.264 Codec are assigned to processors while in data-level decomposition different portions of data are assigned to processors running the same program.

4.1 Task-level Decomposition

In a task-level decomposition the functional partitions of the algorithm are assigned to different processors. As shown in Fig. 1 the process of decoding H.264 consists of performing a series of operations on the coded input bitstream. Some of these tasks can be done in parallel. For example, Inverse Quantization (IQ) and the Inverse Transform (IDCT) can be done in parallel with the Motion Compensation (MC) stage. In Fig. 5a the tasks are mapped to a 4-processor system. A control processor is in charge of synchronization and parsing the bitstream. One processor is in charge of Entropy Decoding, IQ and IDCT, another one of the prediction stage (MC or IntraP), and a third one is responsible for the deblocking filter.

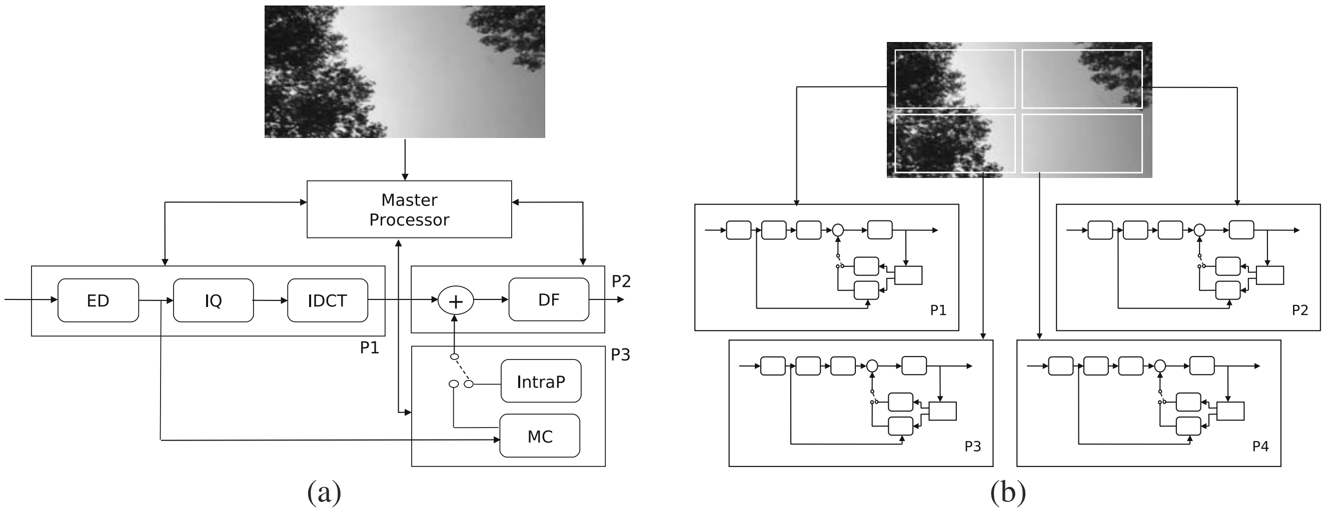

Figure 5 H.264 parallelization techniques. **a** Task-level decomposition. **b** Data-level decomposition.

Task-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 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. This should be performed by a control processor and adds significant overhead.

The main drawbacks, however, of task-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 task-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 reimplement the task partitioning which is a complex task and at some point it could not provide the required performance for high throughput demands. Finally from the software optimization perspective the task-level decomposition requires that each task/processor implements a specific software optimization strategy, i.e., the code for each processor is different and requires different optimizations.

4.2 Data-level Decomposition

In a data-level decomposition the work (data) is divided into smaller parts and each assigned to a different processor, as depicted in Fig. 5b. Each processor runs the same program but on different (multiple) data elements (SPMD). In H.264 data decomposition can be applied at different levels of the data structure (see Fig. 6), which goes down from Group of Pictures (GOP), to frames, slices, MBs, and finally to variable sized pixel blocks. Data-level parallelism can be exploited at each level of the data structure, each one having different constraints and requiring different parallelization methodologies.

4.2.1 GOP-level Parallelism

The coarsest grained parallelism is at the GOP level. H.264 can be parallelized at the GOP-level by defining a GOP size of *N* frames and assigning each GOP to

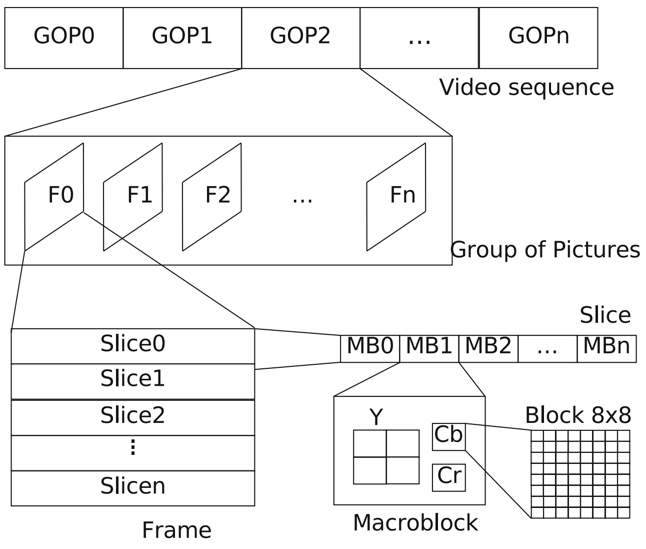

Figure 6 H.264 data structure.

a processor. GOP-level parallelism requires a lot of memory for storing all the frames, and therefore this technique maps well to multicomputers in which each processing node has a lot of computational and memory resources. However, parallelization at the GOP-level 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, because of cache pollution.

4.2.2 Frame-level Parallelism for Independent Frames

After GOP-level there is frame-level parallelism. As shown in Fig. 3 in a sequence of I-B-B-P frames inside a GOP, 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 can assign independent frames 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 B frames can be used as reference [24]. 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.

4.2.3 Slice-level Parallelism

In H.264 and in most current hybrid video coding standards each picture is 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 and losses. In order to accomplish this, slices in a frame should be completely independent from each other. That 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 [10, 16]. Although support for slices have been designed for error resilience, it can be used for exploiting TLP because slices in a frame can be encoded or decoded in parallel. The main advantage of slices is that they can be processed in parallel without dependency or ordering constraints. This allows exploitation of slice-level parallelism without making significant changes to the code.

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 level. If there is no control of what the encoder does then it is possible to receive sequences with few (or one) slices per frame and in such cases there would be reduced parallelization opportunities. The second disadvantage comes from the fact that in H.264 the encoder can decide that the deblocking filter has to be applied across slice boundaries. This greatly reduces the speedup achieved by 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 content of the slice.

Finally, the main disadvantage of slices is that an increase in the number of slices per frame increases the bitrate for the same quality level (or, equivalently, it reduces quality for the same bitrate level). Figure 7 shows the increase in bitrate due to the increase of the number of slices for four different input videos at three different resolutions. The quality is maintained constant (40 PSNR). When the number of slices increases from one to eight, the increase in bitrate is 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%. For some applications this increase in the bitrate is unacceptable and thus 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 is encoded with very little motion estimation, and thus has a large absolute bitrate compared to the other three sequences.

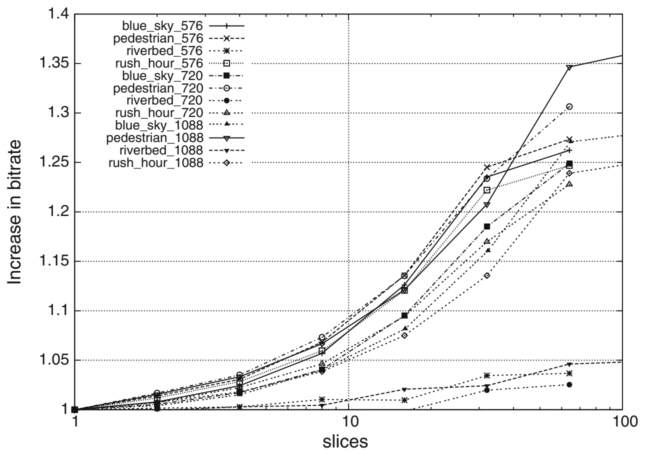

Figure 7 Bitrate increase due to slices.

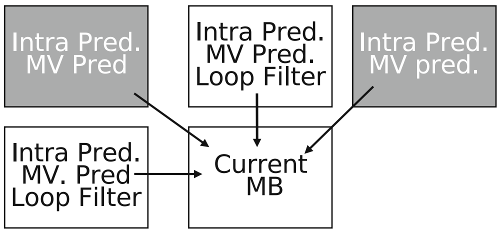

Figure 8 Dependencies between neighboring MBs in H.264.

Thus, the relative increase in bitrate is much lower than the others.

4.2.4 *Macroblock-level Parallelism*

There are two ways of exploiting MB-level parallelism: in the spatial domain and/or in the temporal domain. In the spatial domain MB-level parallelism can be exploited if all the intra-frame dependencies are satisfied. In the temporal domain MB-level parallelism can be exploited if, in addition to the intra-dependencies, inter-frame dependencies are satisfied.

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

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. To exploit parallelism between MBs inside a frame it is necessary to take into account the dependencies between them. In H.264, motion vector prediction, intra prediction, and the deblocking filter use data from neighboring MBs defining a structured set of dependencies. These dependencies are shown in Fig. 8. MBs can be processed out of scan order provided these dependencies are satisfied. 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.

Figure 9 depicts an example for a 5×5 MBs image (80×80 pixels). At time slot T7 three independent MBs can be processed: MB (4,1), MB (2,2) and MB (0,3). The figure also shows the dependencies that need to be satisfied in order to process each of these MBs. The number of independent MBs in each frame depends on the resolution. In Table 2 the number of independent MBs for different resolutions is stated. For a low resolution like QCIF there are only 6 independent MBs during 4 time slots. For High Definition (1920×1088) there are 60 independent MBs during 9 slots of time. Figure 10 depicts the available MB parallelism over

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

Resolution		MBs	Slots
QCIF	176×144	6	4
CIF	352×288	11	8
SD	720×576	23	14
HD	1280×720	40	6
FHD	1920×1088	60	9

Also the number of times slots this maximum is available is stated.

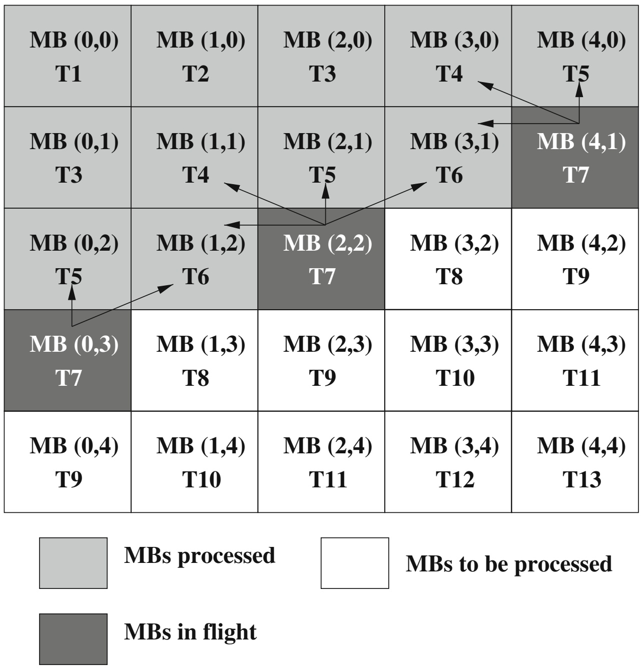

Figure 9 2D-Wave approach for exploiting MB parallelism in the spatial domain. The *arrows* indicate dependencies.

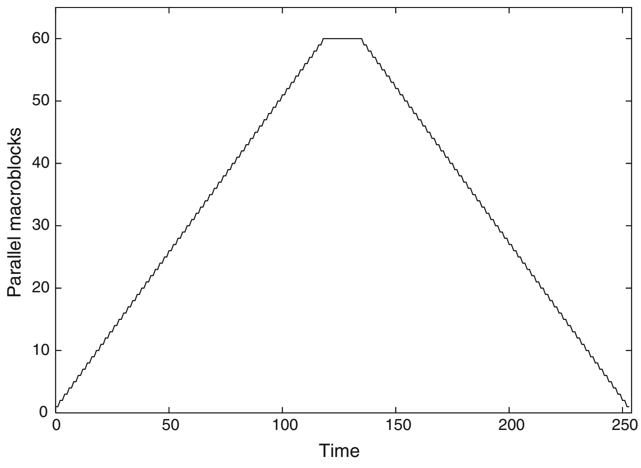

Figure 10 MB parallelism for a single FHD frame using the 2D-Wave approach.

time for a FHD resolution frame, assuming that the time to decode a MB is constant.

MB-level parallelism in the spatial domain has many advantages over other schemes for parallelization of H.264. First, this scheme can have a good scalability. As shown before the number of independent MBs increases with the resolution of the image. 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 Fig. 10) 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. If entropy decoding is accelerated with specialized hardware MB level parallelism could still provide benefits.

4.2.6 *Macroblock-level Parallelism in the Temporal Domain*

In the decoding process the dependency between frames is in the MC module only. MC can be regarded as copying an area, called the reference area, from the reference frame, and then to add this predicted area to the residual MB to reconstruct the MB in the current frame. The reference area is pointed to by a Motion Vector (MV). Although the limit to the MV length is defined by the standard as 512 pixels vertical and 2048 pixels horizontal, in practice MVs are within the range of dozens of pixels.

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 11 shows an example of two frames where the second depends on the first. MBs are decoded in scan order and one at a time. The figure shows that MB (2, 0) of frame $i + 1$ depends on MB (2, 1) of frame i which has been decoded. Thus this MB can be decoded even though frame i is not completely decoded.

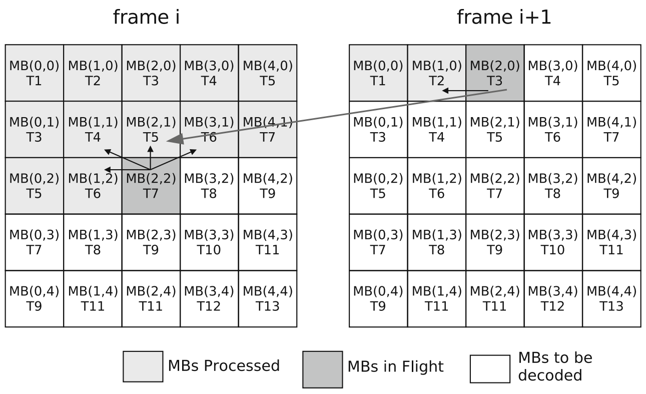

Figure 11 MB-level parallelism in the temporal domain in H.264.

The main disadvantage of this scheme is the 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 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 and in the Intel Integrated Performance Primitives See Section 9.

4.2.7 *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 manycore 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,

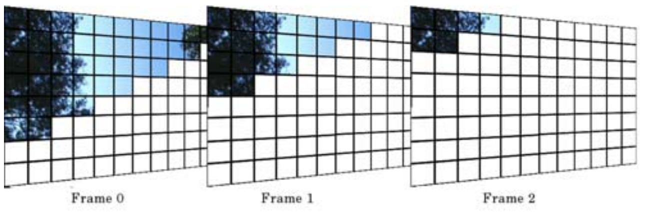

Figure 12 3D-Wave strategy: frames can be decoded in parallel because inter frame dependencies have limited spatial range.

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 spatial 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. Figure 12 illustrates this way of parallel decoding of MBs.

3D-Wave MB 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. This static approach has been implemented in previous work (See Section 9) for the H.264 encoder. 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 (see Fig. 13). 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 6.

4.2.8 Block-level Parallelism

Finally, the finest grained data-level parallelism is at the block-level. Most of the computations of the H.264 kernels are performed at the block level. This applies, for example, to the interpolations that are done in the MC stage, to the IDCT, and to the deblocking filter. This level of data parallelism maps well to SIMD instructions [20, 25–27]. SIMD parallelism is orthogonal to the other levels of parallelism described above and because of that it can be mixed, for example, with MB-level parallelization to increase the performance of each thread.

5 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 [28] 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 his approach is lacking. In order to compare our dynamic approach to his, in this section we analyze the parallel scalability of the Static 3D-Wave.

The Static 3D-Wave strategy assumes a static maximum MV length and thus a static reference range. Figure 14 illustrates the reference range concept,

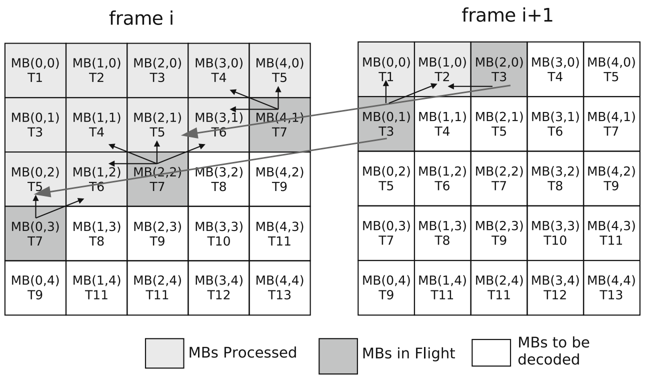

Figure 13 3D-Wave strategy: intra- and inter-frame dependencies between MBs.

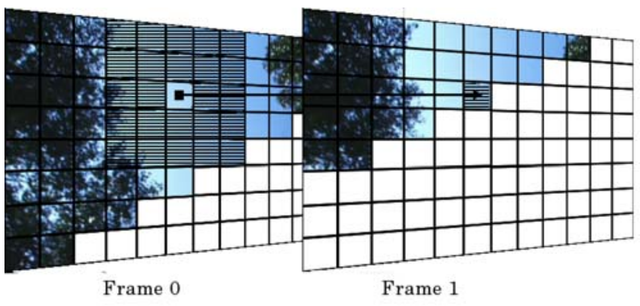

Figure 14 Reference range example: hashed area on frame 0 is the reference range of the hashed MB of frame 1.

assuming a MV range of 32 pixels. The hashed MB in Frame 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 hashed area in Frame 0. In the same way, every MB in the wave front of Frame 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.

As in Section 4.2, to calculate the number of parallel MBs, it is assumed that processing a MB requires one time slot. In reality, different MBs require different processing times, but this can be solved using a dynamic scheduling system. 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 Fig. 10. 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 the offset. Formally, let $h_0(t)$ be the MB parallelism function of the 2D-Wave inside frame 0. This graph of this function is depicted in Fig. 10 for a FHD frame. Then 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 (1,2) and (2,1) of the first frame have been decoded. Of these, MB (1,2) is the last one decoded, namely at time slot T6 (see Fig. 9). Thus, the next frame can be started decoding at time slot T7, 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 9, 15, 27, 51, and 99 time slots, respectively. In general, for a MV range of n pixels, the offset is $3 + 3 \times \lceil n/16 \rceil$ time slots.

First, 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 standard.

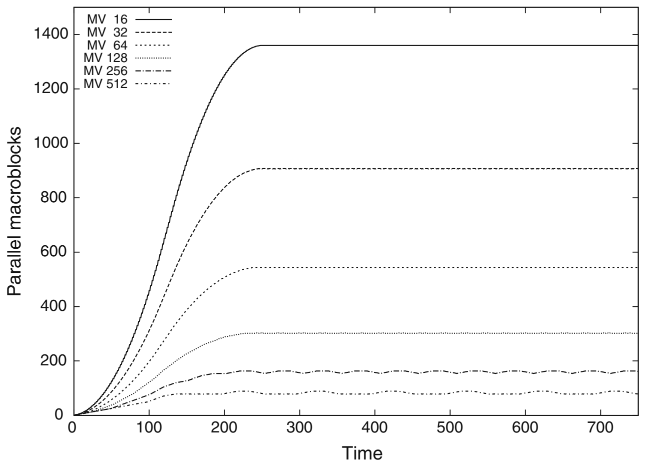

Figure 15 Static 3D-Wave: number of parallel MBs for FHD resolution using several MV ranges.

Figure 15 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. Figure 16 shows the corresponding number of frames in flight for each time slot. The shape of the curves for SD and HD resolutions are similar and, therefore, are omitted. Instead, Table 3 presents the maximum MB parallelism and frames in flight for all resolutions.

The results show that the Static 3D-Wave offers significant parallelism if the MV length is restricted to 16 pixels. However, in most cases this restriction

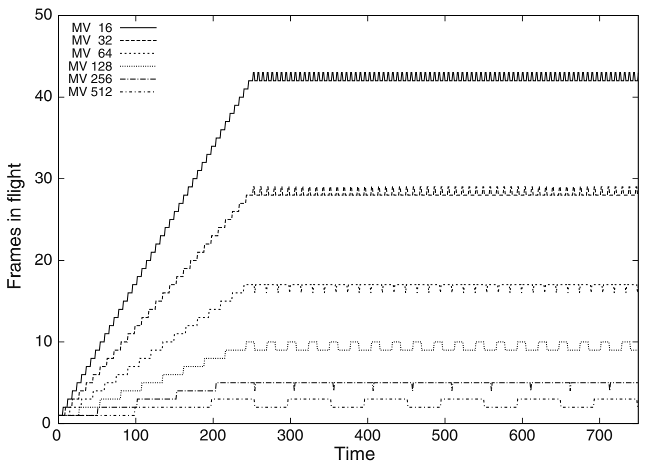

Figure 16 Static 3D-Wave: number of frames in flight for FHD resolution using several MV ranges.

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

MV range	Max # parallel MBs			Max # frames in flight		
	SD	HD	FHD	SD	HD	FHD
512	–	36	89	–	2	3
256	33	74	163	3	4	5
128	60	134	303	5	7	10
64	108	240	544	8	12	17
32	180	400	907	13	19	29
16	276	600	1360	20	28	43

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 approach has little parallelism at the beginning and at the end of processing a frame.

6 Scalable MB-level Parallelism: The Dynamic 3D-Wave

In the previous section we showed that the best parallelization strategy proposed so far has limited value for decoding H.264 because of poor scalability. In Section 4.2.7 we proposed the dynamic 3D-Wave as a scalable solution and explained its fundamentals. In this section we further develop the idea and discuss some of the important implementation issues.

6.1 Static vs Dynamic Scheduling

The main idea of the 3D-wave algorithm is to combine spatial and temporal MB-level parallelism in order to increase the scalability and improve the processor efficiency. But in turn it poses a new problem: the assigning of MBs to processors. In static 3D-wave MBs are assigned to processors based on a fixed description of the data flow of the application. For spatial MB-level parallelism it assigns rows of MBs to processors and start new rows only when a fixed MB offset has been completed. For temporal MB-level parallelism it only starts a new frame when a known offset in the reference frame has been processed. If the processing of the dependent MB (in the same or in a different frame) is taking more time than the predicted by the static schedule the processor has to wait for the dependent MB to finish. This kind of static scheduling algorithms work well for applications with fixed execution times and fixed precedence constraints. In the case of H.264 decoding these conditions are not met. First, the decoding time of a MB depends on the input data of the MB and its properties (type, size, etc). Second, the temporal dependencies of a MB are input dependent and can not be predicted or restricted by the decoder application. As a result, a static scheduler can not discover all the available parallelism and can not sustain a high efficiency and scalability.

The dynamic 3D-wave tries to solve the above mentioned issues by using a dynamic scheduler for assigning independent MBs to threads. The main idea is to assign a MB to a processor as soon as its spatial and temporal dependencies have been resolved. The differences in processing time and the input dependent precedence constraring are considered by trying to process the next ready MB not the next MB in scan, row or frame order. This implies keeping-track both spatial and temporal MB dependencies at run time and generating a dynamic input-dependent data-flow. The global result is a reduction in the time that a thread spend waiting for ready-to-process MBs.

6.2 Implementation Issues of Dynamic Scheduling

This dynamic scheduling can be implemented in various ways. Several issues play a role in this matter such as thread management overhead, data communication overhead, memory latency, centralized vs distributed control, etc. Many of these issues are new to the CMP paradigm and are research projects in itself.

A possible implementation that we are investigating is based on a dynamic task model (also known as a work-queuing model) and is depicted in Fig. 17. In this model a set of threads is created/activated when a parallel region is encountered. In the case of the Dynamic 3D-Wave a parallel region is the decoding of all MBs in a frame/slice. Each parallel region is

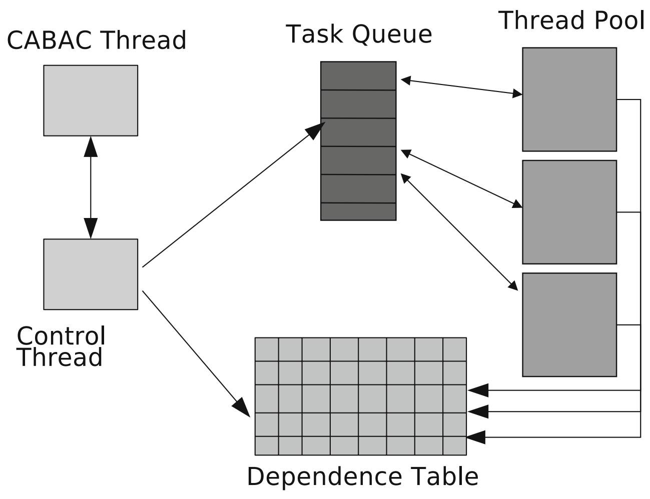

Figure 17 Workqueue model for dynamic 3D-Wave scheduling.

controlled by a frame manager, which consist of a thread pool, a task queue, a dependence table and a control thread as showed in Fig. 17. The thread pool consists of a group of worker threads that wait for work on the task queue. 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 (inter- and intra-frame) for a MB are resolved a new task is inserted on the task queue. Any thread from the task queue can take this task and process it. When a thread finishes the processing of a task it updates the table of dependencies and if it founds that at least one MB has resolved its dependencies the current thread can decode it directly. If that thread wakes-up more than one MB it process one directly and submit the others into the task queue. If at the end of the update of the dependence table there are not a new ready MBs the thread goes to the task pool to ask for a new MB to be decoded. This technique has the advantage of distributed control, low thread management overhead, and also exploits spatial locality, reducing the data communication overhead. In this scheme the control thread is responsible only for handling all the initialization and finalization tasks of each that are not parallelizable, avoiding centralized schedulers that impose a big overhead.

Thread synchronization and communication overhead can also be minimized by taking advantage of clustering patterns in the dependencies and by using efficient hardware primitives for accessing shared structures and managing thread queues [6, 29]. Furthermore, to reduce data communication groups of MBs can be assigned to cores, possibly speculating on the MVs. If cores and memory are placed in a hierarchy, memory latency can be reduced by decoding spatially neighboring MBs in the same leaf of the hierarchy.

6.3 Support in the Programming Model

Currently most of the implementation of this dynamic system should be done with low-level threading APIs like POSIX threads. That is due to the fact that in current programming models for multicore architectures there is not a complete support for a dynamic work-queue model for irregular applications like H.264 decoding. For handling irregular loops with dynamic generation of threads there has been some proposals of extensions to current parallel programming models like OpenMP [30, 31]. A feature that is missing is the possibility of specifying the structured dependencies among the different units of work. Currently this has to be implemented manually by the application programmer.

6.4 Managing Entropy Decoding

The different set of dependencies in the kernels imply that it is necessary to exploit both task- and data-level parallelism efficiently. Task-level decomposition is required because entropy decoding has parallelism at the slice and frame level only. We envision the following mapping to a heterogeneous CMP. Entropy decoding is performed on a high frequency core optimized for bit-serial operations. This can be an 8 or 16 bit RISC like core and therefore can be very power efficient. Several of these cores can operate in parallel each decoding a slice or a frame. The tasks for these cores can be spawned by another core that scans the bitstream, detects the presence or absence of slices, and provides each entropy decoder the start address of its bitstream. Once entropy decoding is performed, the decoding of MBs can be parallelized over as many cores there are available and as long as there are MBs available for decoding. The more cores are used, the lower the frequency and/or voltage can be assuming real-time performance. Thus the power efficiency scales with the number of cores.

One issue that has to be mentioned here is Amdahl's law for multicores, which states that the total speedup is limited by the amount of serial code. Considering a MB-level parallelization there is serial code at the start of each frame and slice. However, using the 3D-Wave strategy, this serial code can be hidden in the background as it can be processed during the time that other frames or slices provide the necessary parallelism.

6.5 Open Issues

Despite the fact that there are many issues to be resolved in this area, it is of major importance to highly parallelize applications. It seems to be clear that in the future there will be manycores. However, what is currently being investigated by computer architecture researchers, just like the authors are doing, is what types of cores should be used, how the cores should be connected, what memory structure should be used, how the huge amount of threads should be managed, what programming model should be used, etc. To analyze design choices like these, it is necessary to analyze the characteristics of highly parallel applications. A scalability analysis like this is of great value in this sense.

7 Parallel Scalability of the Dynamic 3D-Wave

Now we have established a parallelization strategy that we expect to be scalable, we move on to our main goal:

Table 4 Maximum available MB parallelism, and frames in flight for normal encoded movies.

	SD				HD				FHD			
	MBs	Frames	Max mv	Avg mv	MBs	Frames	Max mv	Avg mv	MBs	Frames	Max mv	Avg mv
Rush_hour	1199	92	380.5	1.3	2911	142	435	1.8	5741	211	441	2.2
Riverbed	1511	119	228.6	2	3738	185	496	2.2	7297	245	568.9	2.6
Pedestrian	1076	95	509.7	9.2	2208	131	553.5	11	4196	205	554.2	9.9
Blue_sky	1365	99	116	4.4	2687	135	298	5.1	6041	214	498	5.6

Also the maximum and the average motion vectors (in square pixels) are stated.

analyzing the parallel scalability of H.264 decoding. Remember we perform this analysis in the context of emerging manycore CMPs, that offer huge amounts of threads. In the previous section we explained how we envision to decouple entropy decoding by using task-level parallelism. Thus it scales with the number of slices and frames used in the 3D-Wave MB decoding. However, the parallelism is relatively small compared to the parallelism provided by the MB decoding. Therefore omitting the parallelism in entropy decoding does not significantly influence the results. Of course this assumes the entropy decoders are fast enough to support real-time decoding.

To investigate the amount of parallelism we modified the FFmpeg H.264 decoder to analyze real movies, which we took from the HD-VideoBench. 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,[1] and within a frame the MBs are processed from left to right and from top to bottom, 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.

To start, the maximum available MB parallelism is analyzed. This experiment does not consider any practical or implementation issues, but simply explores the limits to the parallelism available in the application. We use the modified FFmpeg as described before and 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. Finally, we keep track of the motion vector lengths. All experiments are performed for both normal encoded and esa encoded movies. The first uses a hexagonal (hex) motion estimation algorithm with a search range of 24 pixels, while the later uses an exhaustive search algorithm and results in worst case motion vectors.

Table 4 summarizes the results for normal encoded movies and shows that a huge amount of MB parallelism is available. Figure 18 depicts the MB parallelism time curve for FHD, while Fig. 19 depicts the number of frames in flight. 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 blue_sky the plateau starts at time slot 200 and last until time slot 570. 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 4). 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 [32]. For each MB

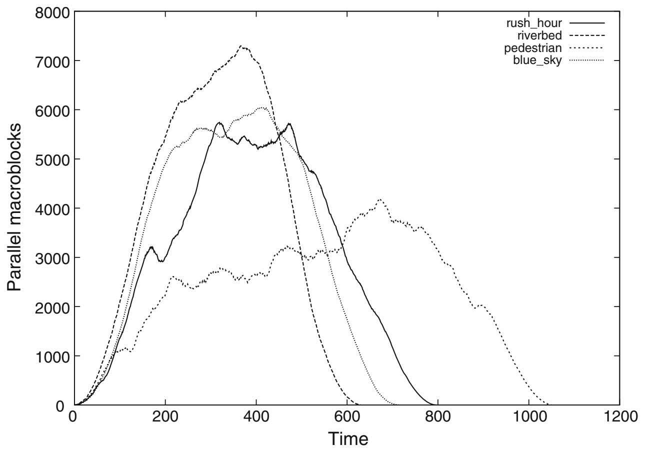

Figure 18 Number of parallel MBs for FHD resolution using a 400 frames sequence.

[1]The decoding order of frames is not equal to display order. The sequence I-B-B-P as in Fig. 3 is decoded as I-P-B-B sequence.

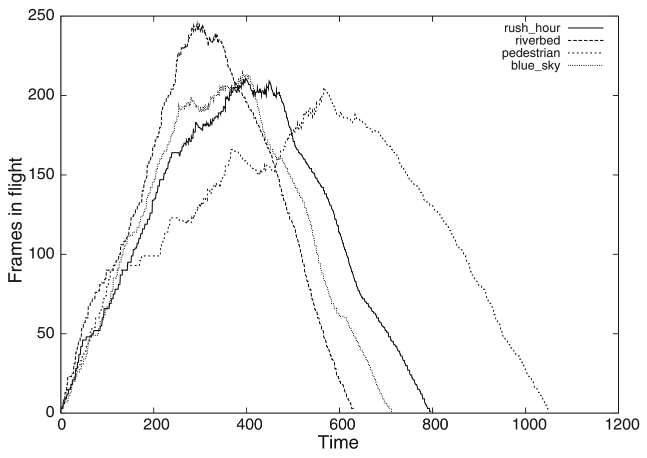

Figure 19 Number of frames in flight for FHD resolution using a 400 frames sequence.

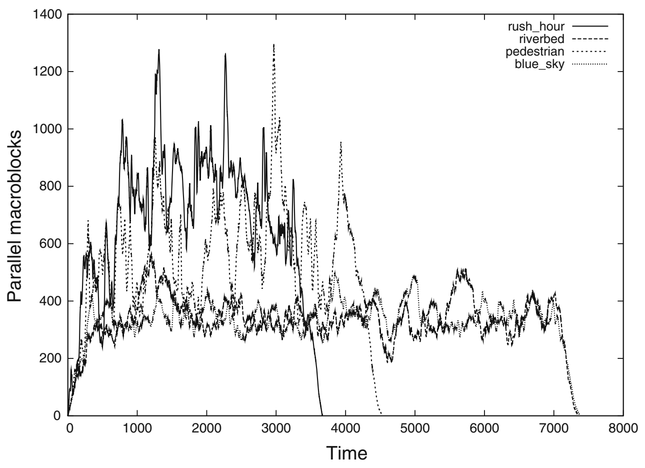

Figure 20 Number of parallel MBs for FHD using esa encoding.

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.

To evaluate the worst case condition the same experiment was performed for movies encoded using the exhaustive search algorithm (esa). Table 5 shows that the average MV length is substantially larger than for normal encoded movies. The exhaustive search decreases the amount of parallelism significantly. The time curve of the MB parallelism in Fig. 20 has a very jagged shape with large peaks and thus the average on the plateau is much lower than the peak. In Table 6 the maximum and the average parallelism for normal (hex) and esa encoded movies is compared. The average is taken over all 400 frames, including ramp up and ramp down. Although the drop in parallelism is significant, the amount of parallelism is still surprisingly large for this worst case scenario.

The analysis above reveals that H.264 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, and NoC bandwidth) will be limited. We now investigate the impact of resource limitations.

We model limited resources as follows. A limited number of cores is modeled by limiting the number of MBs in flight. 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. Limited NoC bandwidth is captured by both modeled restrictions. Both restrictions decrease the throughput, which is directly related to the inter-core communication.

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.

First, the impact of limiting the number of MBs in flight is analyzed. Figure 21 depicts the available MB parallelism for several limits on the number of MBs in flight. As expected, for smaller limits, the height of the plateau is lower and the ramp up and ramp down are shorter. More important is that for smaller limits, the plateau becomes very flat. This translates to a high

Table 5 Maximum MB parallelism, and frames in flight for esa encoded movies.

	SD				HD				FHD			
	MBs	Frames	Max mv	Avg mv	MBs	Frames	Max mv	Avg mv	MBs	Frames	Max mv	Avg mv
Rush_hour	505	63	712.0	6.9	708	84	699.3	9	1277	89	706.4	9.4
Riverbed	150	27	704.9	62.3	251	25	716.2	63.1	563	37	716.2	83.9
Pedestrian	363	50	691.5	23.5	702	70	709.9	27.7	1298	88	708.5	23.6
Blue_sky	138	19	682.0	10.5	257	23	711.3	13	500	24	705.7	15.7

Also the maximum and the average motion vectors (in square pixels) are stated.

Table 6 Comparison of the maximum and average MB parallelism for hex and esa encoded movies (FHD only).

	FHD					
	Max MB			Avg MB		
	hex	esa	Δ	hex	esa	Δ
Rush_hour	5741	1277	−77.8%	4121.2	892.3	−78.3%
Riverbed	7297	563	−92.3%	5181.0	447.7	−91.4%
Pedestrian	4196	1298	−69.1%	3108.6	720.8	−76.8%
Blue_sky	6041	500	−91.7%	4571.4	442.6	−90.3%

utilization rate of the available cores. Furthermore, the figure shows that the decoding time is approximately linear in the limit on MBs in flight. This is especially true for smaller limits, where the core utilization is almost 100%. For larger limits, the core utilization decreases and the linear relation ceases to be true. In real systems, however, also communication overhead has to be taken into account. For smaller limits, this overhead can be reduced using a smart scheduler that assigns groups of adjacent MBs to cores. Thus, in real systems the communication overhead will be less for smaller limits than for larger limits.

Next, we analyze the impact of restricting the number of frames concurrently in flight. The MB parallelism curves are depicted in Fig. 22 and 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.

From this experiment we can conclude that for systems with limited number of cores dynamic scheduling is able to achieve near optimal performance by (almost) fully utilizing the available computational power. On the contrary, for systems where memory is the bottleneck, additional performance losses might occur because of temporal underutilization of the available cores. In Section 8 we perform a case study, indicating that memory will likely not be a bottleneck.

We have shown that using the Dynamic 3D-Wave strategy huge amounts of parallelism are available in H.264. Analysis of real movies revealed that the number of parallel MBs ranges from 4000 to 7000. This amount of MB parallelism might be larger than the number of cores available. Thus, we have evaluated the parallelism for limited resources and found that limiting the number of MBs in flight results in a equivalent larger decoding time, but a near optimal utilization of the cores.

8 Case Study: Mobile Video

So far we mainly focused on high resolution and explored the potential available MB parallelism. The 3D-Wave strategy allows an enormous amount of parallelism, possibly more than what a high performance CMP in the near future could provide. Therefore, in this section we perform a case study to assess the practical value and possibilities of a highly parallelized H.264 decoding application.

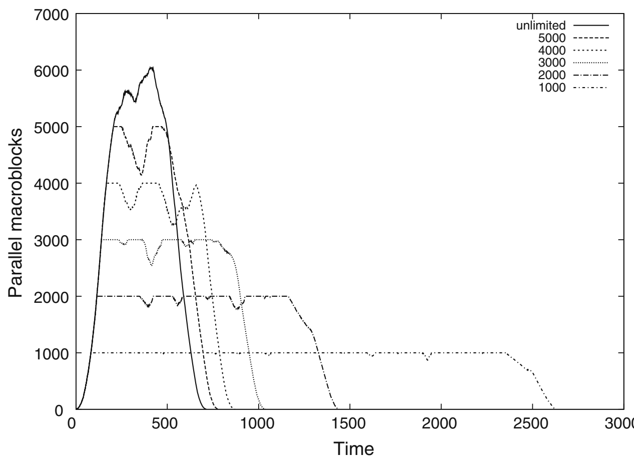

Figure 21 Available MB parallelism in FHD blue_sky, for several limits of the number of MBs in flight.

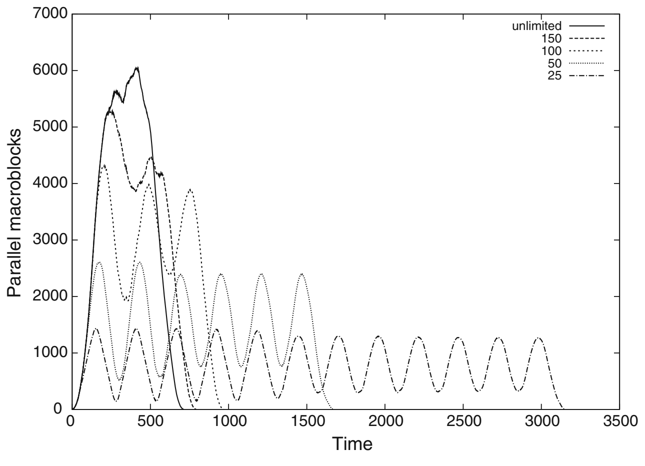

Figure 22 Available MB parallelism in FHD blue_sky for several limits of the number of frames in flight.

For this case study we assume a mobile device such as the iPhone, but in the year 2015. We take a resolution of 480×320 just as the screen of the current iPhone. It is reasonable to expect that the size of mobile devices will not significantly grow, and limitations on the accuracy of the human eye prevent the resolution from increasing.

CMPs are power efficient, and thus we expect mobile devices to adopt them soon and project a 100-core CMP in 2015. Two examples of state-of-the art embedded CMPs are the Tilera Tile64 and the Clearspeed CSX600. The Tile64 [2] processor has 64 identical cores that can run fully autonomously and is well suited to exploit TLP. The CSX600 [1] contains 96 processing elements, which are simple and exploit DLP. Both are good examples of what is already possible today and the eagerness of the embedded market to adopt the CMP paradigm. Some expect a doubling of cores every three years [3]. Even if this growth would be less the assumption of 100 cores in 2015 seems reasonable.

The iPhone is available with 8 GB of memory. Using Moore's law for memory (doubling every two years), in 2015 this mobile device would contain 128 GB. It is unclear how much of the flash drive memory is used as main memory, but let us assume it is only 1%. In 2015 that would mean 1.28 GB and if only half of it is available for video decoding, then still almost 1400 frames of video fit in main memory. It seems that memory is not going to be a bottleneck.

We place a limit of 1 s on the latency of the decoding process. That is a reasonable time to wait for the movie to appear on the screen, but much longer would not be acceptable. The iPhone uses a frame rate of 30 frames/second, thus limiting the frames in flight to 30, causes a latency of 1 s. This can be explained as follows. Let us assume that the clock frequency is scaled down as much as possible to save power. When frame x is started decoding, frame $x - 30$ has just finished and is currently displayed. But, since the framerate is 30, frame x will be displayed 1 second later. Of course this a rough method and does not consider variations in frame decoding time, but it is a good estimate.

Putting all assumptions together, the resulting Dynamic 3D-Wave strategy has a limit of 100 parallel MBs and up to 30 frames can be in flight concurrently. For this case study we use the four normal encoded movies with a length of 200 frames and a resolution of 480×320.

Figure 23 presents the MB parallelism under these assumptions as well as for unlimited resources. The picture shows that even for this small resolution, all 100 cores are utilized nearly all time. The curves for unrestricted resources show that there is much more parallelism available than the hardware of this case study offers. This provides opportunities for scheduling algorithms to reduce communication overhead. The number of frames in flight is on average 12 with peaks up to 19. From this we can conclude that the latency is less then 0.5 s.

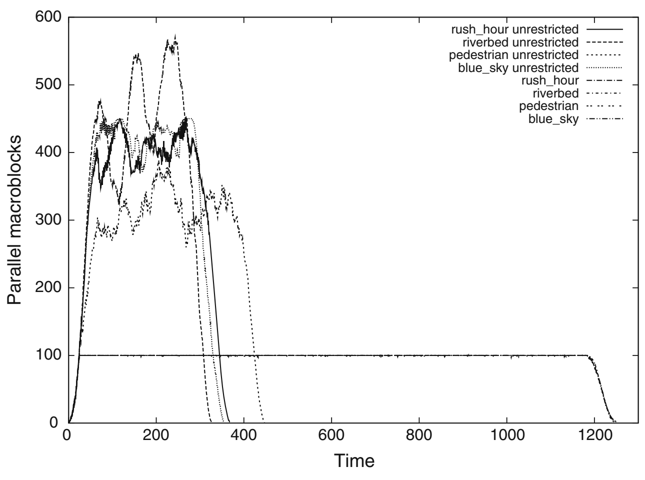

Figure 23 Number of parallel MBs for the mobile video case study. Also depicted is the available MB parallelism for the same resolution but with unrestricted resources.

From this case study we can draw the following conclusions:

- Even a low resolution movie exhibits sufficient parallelism to fully utilize 100 cores efficiently.
- For mobile devices, memory will likely not be a bottleneck.
- For mobile devices with a 100-core CMP, start-up latency will be short (0.5s).

9 Related Work

A lot of work has been done on the parallelization of video Codecs but most of it has been focused on coarse grain parallelization for large scale multiprocessor systems. Shen et al. [33] have presented a parallelization of the MPEG-1 encoder for the Intel Paragon MIMD multiprocessor at the GOP level. Bilas et al. [34] has described an implementation of the MPEG-2 decoder evaluating GOP and slice-level parallelism on a shared memory multiprocessor. Akramullah et al. [35] have reported a parallelization of the MPEG-2 encoder for cluster of workstations and Taylor et al. [36] have implemented a MPEG-1 encoder in a multiprocessor made of 2048 custom DSPs.

There are some works on slice level parallelism for previous video Codecs. Lehtoranta et al [37] 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. Lee et al. [38] have also reported an implementation of the MPEG-2 decoder exploiting slice-level parallelism for HDTV input videos.

Other works have analyzed function-level parallelism. Lin et al. [39] and Cantineau et al. [40] have reported functional parallelization of the H.263 and MPEG-2 encoders respectively. They have used a multiprocessor with one control processor and four DSPs. Using the same technique Oehring et al. [41] have reported the parallelization of the MPEG-2 decoder on a simulator for SMT processors with 4 threads. Yadav et al. [42] 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. Jacobs et al. [43] have analyzed MB-level parallelism for MPEG-2 and MPEG-4.

In the specific case of H.264 there has been previous works in the parallelization at GOP, frame, slice, function and MB levels.

Gulati et al. [44] describe a system for encoding and decoding H.264 on a multiprocessor architecture using a task-level decomposition approach. The decoder is implemented using one control processor and 3 DSPs. Using this mapping, a pipeline for processing MBs is implemented. This system achieves real-time operation for low resolution video inputs. In a similar way Schoffmann et al. [45] propose a MB pipeline model for H.264 decoding. Using an Intel Xeon 4-way SMP 2-SMT architecture their scheme achieves a 2X speed-up. As as mentioned in Section 4.2 MB pipelining does not scale for manycore architectures because it is very complicated to further divide the mapping of tasks for using more processors.

Some works describe GOP, frame and slice-level parallelism and combinations between them. Rodriguez at al. [46] propose an encoder that combines GOP-level and slice-level parallelism for encoding real-time H.264 video using clusters of workstations. 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 only applicable to the encoder, which has the freedom of selecting the GOP size. Chen et al. [47] have pproposed a combination of frame-level and slice-level parallelism. First, they exploit frame level parallelism (they do not allow the use of B-frames as references). When the limit of frame-level parallelism has been reached they explore slice-level parallelism. By using this approach 4.5X speed-up is achieved in a machine with 8 cores and 9 slices. Jacobs et al. [43] proposed a pure slice-level parallelization of H.264 encoder. Roitzsch [48] has proposed a scheme based on slice-level parallelism for the H.264 decoder by modifying the encoder. The main idea is to overcome the load balancing disadvantage by making an encoder that produces slices that are not balanced in the number of MBs, but in their decoding time. The main disadvantages of this approach is that it requires modifications to the encoder in order to exploit parallelism at the decoder. In general, all the proposals based on slice-level parallelism have the problem of the inherent loss of coding efficiency due to having a large number of slices.

MB-level parallelism in the spatial domain has been proposed in several works. Van der Tol et al. [49] describe it using a simulated shared memory WLIW multiprocessor. Chen et al. [50] evaluated an implementation on Pentium machines with SMT and CMP capabilities. In these works they also suggest the combination of MB-level level parallelism in the spatial and temporal domains. Temporal MB-level parallelism is determined statically by the length of the motion vectors and using only two consecutive frames.

MB-level parallelism in the temporal domain has been implemented in the X264 open-source encoder [18] and in the H.264 Codec that is part of the Intel Integrated Performance Primitives [51]. In these cases the encoder limits the vertical motion search range in order to allow a next frame to start the encoding before the current one completes. In these implementations MB-level in the spatial domain is not exploited, and thus only one MB per frame at a time is decoded, resulting in a limited scalability. As mentioned in Section 4.2 this approach can not be exploited efficiently in the decoder because there is not a guarantee that the motion vectors have limited values.

A combination of temporal and spatial MB-level parallelism for H.264 encoding has been discussed by Zhao et al. [28]. 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 MB-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 result in poor load balancing because the decoding time of MBs is not constant. 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 into account the peculiarities of the decoder. Finally, they do not address the scalability issues of their approach in the context of manycore architectures and high definition applications.

Chong et al. [52] has proposed another technique for exploiting MB level parallelism in the H.264 decoder by adding a prepass stage. 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 6 processors simulated system for small resolution input videos. Although they present a dynamic scheduling algorithm it seems to be not able of discovering all the MB level parallelism that is available in a frame. However, the preparsing scheme can be beneficial for the Dynamic 3D-Wave algorithm presented in this paper.

10 Conclusions

In this paper we have investigated if future applications exhibit sufficient parallelism to exploit the large number of cores expected in future CMPs. This is important for performance but maybe even more important for power efficiency. As a case study, we have analyzed the parallel scalability of the H.264 video decoding process, currently the best coding standard.

First, we have discussed the parallelization possibilities and showed that slice-level parallelism has two main limitations. First, using many slices increases the bitrate and, second, not all sequences contain many slices since the encoder determines the number of slices per frame. It was also shown that frame-level parallelism, which exploits the fact that some frames (B frames) are not used as reference frames and can therefore be processed in parallel, is also not very scalable because usually there are no more than three B frames between consecutive P frames and, furthermore, in H.264 B frames can be used as reference frames.

More promising is MB-level parallelism, which can be exploited inside a frame in a diagonal wavefront manner. It was observed that MBs in different frames are only dependent through motion vectors which have a limited range. We have therefore proposed a novel parallelization strategy, called Dynamic 3D-Wave, which combines MB parallelism with frame-level parallelism.

Using this new parallelization strategy we performed a limit study to the amount of MB-level parallelism available in H.264. We modified FFmpeg and analyzed real movies. The results revealed that motion vectors are small on average. As a result a large amount of parallelism is available. For FHD resolution the total number of parallel MBs ranges from 4000 to 7000.

Combining MB and frame-level parallelism was done before for H.264 encoding in a static way. To compare this with our new dynamic scheme we analyzed the scalability of the Static 3D-Wave approach for decoding. Our results show that this approach can deliver a parallelism of 1360 MBs, but only if motion vectors are strictly shorter than 16 pixels. In practice this condition cannot be guaranteed and the worst case motion vector length has to be assumed. This results in a parallelism of 89 MBs for FHD. It is clear that the Dynamic 3D-Wave outperforms the static approach.

We have also performed a case study to assess the practical value and possibilities of the highly parallelized H.264 decoding. In general, the results show that our strategy provides sufficient parallelism to efficiently exploit the capabilities of future manycore CMPs.

The large amount of parallelism available in H.264 indicates that future manycore CMPs can effectively be used. Although we have focused on H.264, other video codecs and multimedia applications in general exhibit similar features and we expect that they can also exploit the capabilities of manycore CMPs. However, the findings of this paper also pose a number of new questions, which we intend to investigate in the future.

First, we will extend the parallelism analysis to include variable MB decoding time. We will investigate its impact on the dynamic scheduling algorithm, the amount of available parallelism, and performance. Second, we will investigate MB clustering and scheduling techniques to reduce communication and synchronization overhead. We are currently working on an implementation of the 3D-Wave strategy for an aggressive multicore architecture and the MB scheduling and clustering algorithm is an important part of that. Finally, we will investigate ways to accelerate entropy coding. More specifically, we plan to design a hardware accelerator tailored for contemporary and future entropy coding schemes.

Acknowledgements This work was partially supported by the European Commission in the context of the SARC integrated project #27648 (FP6), the Ministry of Science of Spain and European Union (FEDER funds) under contract TIC-2004-07739-C02-01, and HiPEAC, European Network of Excellence on High-Performance Embedded Architecture and Compilation. The authors would like to thank Jan Hoogerbrugge from NXP labs for the valuable discussions on parallelizing H.264.

References

1. ClearSpeed (2008) The CSX600 processor. [Online]. http://www.clearspeed.com.
2. Tilera (2007) TILE64(TM) processor family. [Online]. http://www.tilera.com.
3. Stenström, P. (2006). Chip-multiprocessing and beyond. In *Proc. twelfth int. symp. on high-performance computer architecture* (pp. 109–109).
4. Asanovic, K., et al. (2006). The landscape of parallel computing research: A view from Berkeley. EECS Department University of California, Berkeley, Tech. Rep. UCB/EECS-2006-183, December.
5. Liao, H., & Wolfe, A. (1997) Available parallelism in video applications. In *Proc. 30th annual int. symp. on microarchitecture (Micro '97)*.
6. Kissell, K. (2008). MIPS MT: A multithreaded RISC architecture for embedded real-time processing. In *Proc. high performance embedded architectures and compilers (HiPEAC) conference*.
7. Oelbaum, T., Baroncini, V., Tan, T., & Fenimore, C. (2004). Subjective quality assessment of the emerging AVC/H.264 video coding standard. In *Int. broadcast conference (IBC)*.
8. International Standard of Joint Video Specification (ITU-T Rec. H. 264| ISO/IEC 14496-10 AVC) (2005).
9. Flierl, M., & Girod, B. (2003). 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.
10. Wiegand, T., Sullivan, G. J., Bjontegaard, G., & A.Luthra (2003). Overview of the H.264/AVC video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology, 13*(7), 560–576, July.
11. Tamhankar, A., & Rao, K. (2003). An overview of H. 264/MPEG-4 Part 10. In *Proc. 4th EURASIP conference focused on video/image processing and multimedia communications* (p. 1).
12. Malvar, H., Hallapuro, A., Karczewicz, M., & Kerofsky, L. (2003). Low-complexity transform and quantization in H. 264/AVC. *IEEE Transactions on Circuits and Systems for Video Technology, 13*(7), 598–603.
13. Wien, M. (2003). Variable block-size transforms for H. 264/AVC. *IEEE Transactions on Circuits and Systems for Video Technology, 13*(7), 604–613.
14. List, P., Joch, A., Lainema, J., Bjntegaard, G., & Karczewicz, M. (2003). Adaptive deblocking filter. *IEEE Transactions on Circuits and Systems for Video Technology, 13*(7), 614–619.
15. Marpe, D., Schwarz, H., & Wiegand, T. (2003). 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.
16. Sullivan, G., Topiwala, P., & Luthra, A. (2004). The H.264/AVC advanced video coding standard: Overview and introduction to the fidelity range extensions. In *Proc. SPIE conference on applications of digital image processing XXVII* (pp. 454–474).
17. Alvarez, M., Salami, E., Ramirez, A., & Valero, M. (2007). HD-VideoBench: A benchmark for evaluating high definition digital video applications. In *IEEE int. symp. on workload characterization*, [Online]. http://personals.ac.upc.edu/alvarez/hdvideobench/index.html.
18. X264 (2008). A free H.264/AVC encoder. [Online]. http://developers.videolan.org/x264.html.
19. FFmpeg (2008). The FFmpeg Libavcoded [Online]. http://ffmpeg.mplayerhq.hu/.
20. Alvarez, M., Salami, E., Ramirez, A., & Valero, M. (2005). A performance characterization of high definition digital video decoding using H.264/AVC. In *Proc. IEEE int. workload characterization symposium* (pp. 24–33).
21. Ostermann, J., Bormans, J., List, P., Marpe, D., Narroschke, M., Pereira, F., et al. (2004). Video coding with H.264/AVC: Tools, performance, and complexity. *IEEE Circuits and Systems Magazine, 4*(1), 7–28.
22. Lappalainen, V., Hallapuro, A., & Hamalainen, T. D. (2003). Complexity of optimized H.26L video decoder implementation. *IEEE Transactions on Circuits and Systems for Video Technology, 13*(7), 717–725.
23. Horowitz, M., Joch, A., & Kossentini, F. (2003). H.264/AVC baseline profile decoder complexity analyis. *IEEE Transactions on Circuits and Systems for Video Technology, 13*(7), 704–716.
24. Flierl, M., & Girod, B. (2003). 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.
25. Zhou, X., Li, E. Q., & Chen, Y.-K. (2003). Implementation of H.264 decoder on general-purpose processors with media instructions. In *Proc. SPIE conf. on image and video communications and processing*.
26. Shojania, H., Sudharsanan, S., & Wai-Yip, C., (2006). Performance improvement of the h.264/avc deblocking filter using simd instructions. In *Proc. IEEE int. symp. on circuits and systems ISCAS*, May.
27. Lee, J., Moon, S., & Sung, W. (2004). H.264 decoder optimization exploiting SIMD instructions. In *Asia-Pacific conf. on circuits and systems*, Dec.
28. Zhao, Z., & Liang, P. (2006). Data partition for wavefront parallelization of H.264 video encoder. In *IEEE international symposium on circuits and systems, ISCAS 2006*, 21–24 May.
29. Schaumont, P., Lai, B.-C. C., Qin, W., & Verbauwhede, I. (2005). Cooperative multithreading on embedded multiprocessor architectures enables energy-scalable design. In *DAC '05: Proceedings of the 42nd annual conference on design automation* (pp. 27–30).
30. Tian, X., Chen, Y.-K., Girkar, M., Ge, S., Lienhart, R., & Shah, S. (2003). Exploring the use of hyper-threading technology for multimedia applications with intel openmp compiler. In *Proceedings of the international parallel and distributed processing symposium, 2003*, 22–26 April.
31. Ayguadé, E., Copty, N., Duran, A., Hoeflinger, J., Lin, Y., Massaioli, F., et al. (2007). A proposal for task parallelism in OpenMP. In *Proceedings of the 3rd international workshop on OpenMP*, June.
32. X264-devel – Mailing list for ×264 developers (2007). Subject: Out-of-range motion vectors. [Online]. http://mailman.videolan.org/listinfo/×264-devel, July–August.
33. Shen, K., Rowe, L. A., & Delp, E. J. (1995). Parallel implementation of an MPEG-1 encoder: Faster than real time. In *Proc. SPIE, digital video compression: Algorithms and technologies 1995* (Vol. 2419, pp. 407–418).
34. Bilas, A., Fritts, J., & Singh, J. (1997). Real-time parallel mpeg-2 decoding in software. In *Parallel processing sympo-*

sium, 1997. Proceedings., 11th international (pp. 197–203), 1–5 April.
35. Akramullah, S., Ahmad, I., & Liou, M. (1997). 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, August.
36. Taylor, H., Chin, D., & Jessup, A. (1993). A mpeg encoder implementation on the princeton engine video supercomputer. In *Data compression conference, 1993. DCC '93.* (pp. 420–429).
37. Lehtoranta, O., Hamalainen, T., & Saarinen, J. (2001). Parallel implementation of h.263 encoder for cif-sized images on quad dsp system. In *The 2001 IEEE international symposium on circuits and systems, ISCAS 2001* (Vol. 2, pp. 209–212), 6–9 May.
38. Lee, C., Ho, C. S., Tsai, S.-F., Wu, C.-F., Cheng, J.-Y., Wang, L.-W., et al. (1996). Implementation of digital hdtv video decoder by multiple multimedia video processors. In *International conference on consumer electronics, 1996* (pp. 98–), 5–7 June.
39. Lin, W., Goh, K., Tye, B., Powell, G., Ohya, T., & Adachi, S. (1997). Real time h.263 video codec using parallel dsp. In *International Conference on Image Processing, 1997* (Vol. 2, pp. 586–589), 26–29 October.
40. Cantineau, O., & Legat, J.-D. (1998). Efficient parallelisation of an mpeg-2 codec on a tms320c80 video processor. In *1998 international conference on image processing, 1998. ICIP 98* (Vol. 3, pp. 977–980), 4–7 October.
41. Oehring, H., Sigmund, U., & Ungerer, T. (1999). Mpeg-2 video decompression on simultaneous multithreaded multimedia processors. In *International conference on parallel architectures and compilation techniques, 1999* (pp. 11–16).
42. Ganesh Yadav, R. S., & Chaudhary, V. (2004). 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.
43. Jacobs, T., Chouliaras, V., & Mulvaney, D. (2006). 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, February.
44. Gulati, A., & Campbell, G. (2005). Efficient mapping of the H.264 encoding algorithm onto multiprocessor DSPs. In *Proc. embedded processors for multimedia and communications II, 5683*(1), 94–103, March.
45. Klaus Schöffmann, O. L., Fauster, M., & Böszörmenyi, L. (2007). 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.
46. Rodriguez, A., Gonzalez, A., & Malumbres, M. P. (2006). Hierarchical parallelization of an h.264/avc video encoder. In *Proc. int. symp. on parallel computing in electrical engineering* (pp. 363–368).
47. Chen, Y., Tian, X., Ge, S., & Girkar, M. (2004). Towards efficient multi-level threading of h.264 encoder on intel hyper-threading architectures. In *Proc. 18th int. parallel and distributed processing symposium*.
48. Roitzsch, M. (2006). Slice-balancing H.264 video encoding for improved scalability of multicore decoding. In *Work-in-progress proc. 27th IEEE real-time systems symposium*.
49. van der Tol, E., Jaspers, E., & Gelderblom, R. (2003). Mapping of H.264 decoding on a multiprocessor architecture. In *Proc. SPIE conf. on image and video communications and processing*.
50. Chen, Y., Li, E., Zhou, X., & Ge, S. (2006). Implementation of H. 264 encoder and decoder on personal computers. *Journal of Visual Communications and Image Representation, 17*, 509–532.
51. Intel Integrated Performance Primitives (2008). [Online]. http://www.intel.com/cd/software/products/asmo-na/eng/perflib/ipp/302910.htm.
52. Chong, J., Satish, N. R., Catanzaro, B., Ravindran, K., & Keutzer, K. (2007). Efficient parallelization of h.264 decoding with macro block level scheduling. In *2007 IEEE international conference on multimedia and expo* (pp. 1874–1877), July.

Cor Meenderinck received the MSc degree in electrical engineering from Delft University of Technology, the Netherlands. Currently, he is working toward the PhD degree in the Computer Engineering Laboratory of the Faculty of Electrical Engineering, Mathematics and Computer Science of Delft University of Technology, the Netherlands. His research interests include computer architecture, chip multi-processors, media accelerators, design for power efficiency, design for variability, computer arithmetic, nano electronics, and single electron tunneling.

Arnaldo Azevedo received the BSc degree in computer science from the UFRN University, Natal, RN, Brazil, in 2004 and the MSc degree in computer science from UFRGS University, Porto Alegre, RS, Brazil, in 2006. Since 2006, he is a doctoral candidate in the Computer Engineering Laboratory of the Faculty of Electrical Engineering, Mathematics and Computer Science of Delft University of Technology, the Netherlands. He is currently investigating multimedia accelerators architecture for multi-core processors.

Ben Juurlink is an associate professor in the Computer Engineering Laboratory of the Faculty of Electrical Engineering, Mathematics, and Computer Science at Delft University of Technology, the Netherlands. He received the MSc degree in computer science, from Utrecht University, Utrecht, the Netherlands, in 1992, and the Ph.D. degree also in computer science from Leiden University, Leiden, the Netherlands, in 1997. His research interests include instruction-level parallel processors, application-specific ISA extensions, low power techniques, and hierarchical memory systems. He has (co-) authored more than 50 papers in international conferences and journals and is a senior member of the IEEE and a member of the ACM.

Mauricio Alvarez Mesa received the BSc degree in electronic engineering from University of Antioquia, Medellin, Colombia in 2000. From 2000 to 2002 he was a teaching assistant at Department of Electronic Engineering of the this University. In 2002 he joined the High Performance Computing Group at the Computer Architecture Department of the Technical University of Catalonia (UPC) where he is doing his PhD. From 2006 he became teaching assistant at UPC. He was a summer student intern at IBM Haifa Research labs, Israel in 2007. His research interest includes high performance architectures for multimedia applications, vector processors, SIMD extensions, multicore architectures and streaming architectures.

Alex Ramirez is an associate professor in the Computer Architecture Department at the Universitat Politecnica de Catalunya, and leader of the Computer Architecture group at BSC. He has a BSc ('95), MSc ('97) and PhD ('02, awarded the UPC extraordinary award to the best PhD in computer science) in computer science from the Universitat Politecnica de Catalunya (UPC), Barcelona, Spain. He has been a summer student intern with Compaq's WRL in Palo Alto, California for two consecutive years ('99–'00), and with Intel's Microprocessor Research Laboratory in Santa Clara ('01). His research interests include compiler optimizations, high performance fetch architectures, multithreaded architectures, and vector architectures. He has coauthored over 50 papers in international conferences and journals and supervised 3 PhD students.