Parallel Scalability of H.264

Cor Meenderinck[1], Arnaldo Azevedo[1], Mauricio Alvarez[2], Ben Juurlink[1], and Alex Ramirez[3]

[1] Delft University of Technology, Delft, the Netherlands
{Cor, Azevedo, Benj}@ce.et.tudelft.nl
[2] Technical University of Catalonia (UPC), Barcelona, Spain
alvarez@ac.upc.edu
[3] Barcelona Supercomputing Center (BSC), Barcelona, Spain
alex.ramirez@bsc.es

Abstract. An important question is whether emerging and future applications exhibit sufficient parallelism, in particular thread-level parallelism (TLP), to exploit the large numbers of cores future CMPs are expected to contain. As a case study we investigate the parallel scalability of the H.264 decoding process. Previously proposed parallelization strategies such as slice-level, frame-level, and intra-frame macroblock (MB) level parallelism, are not sufficiently scalable. We therefore propose a novel strategy, called 3D-Wave, which is mainly based on the observation that inter-frame dependencies have a limited spatial range. Because of this, certain MBs of consecutive frames can be decoded in parallel. The 3D-Wave strategy allows 4000 to 9000 MBs to be processed in parallel, depending on the input sequence. We also perform a case study to assess the practical value and possibilities of the 3D-Wave strategy. The results show that our strategy provides sufficient parallelism to efficiently exploit the capabilities of future manycore CMPs.

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). It seems, however, that these sources of performance gains are exhausted. As a result, industry, including IBM, Sun, Intel, and AMD, turned to CMPs.

It is expected that the number of cores on a CMP will double every three year [1], 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 [2]. 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 suffers. Furthermore, CMPs provide three levels of parallelism, i.e., data-level parallelism (DLP), instruction-level parallelism (ILP), and thread-level parallelism

(TLP). How do they need to be balanced to match the parallelism available in applications?

In this paper we answer these questions for video coding/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 consider an H.264 decoder, analyze the parallelism available, and propose a new parallelization strategy that provides sufficient parallelism to fully utilize the many cores that will be available.

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 propose the 3D-Wave parallelization strategy and show that it can exploit huge amounts of parallelism. Section 6 describes a case study to assess the practical value of the proposed strategy. Section 7 concludes the paper.

2 Overview of the H.264 Standard

Currently, the best video coding standard, in terms of compression and quality is H.264 [3]. 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 [4] 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.

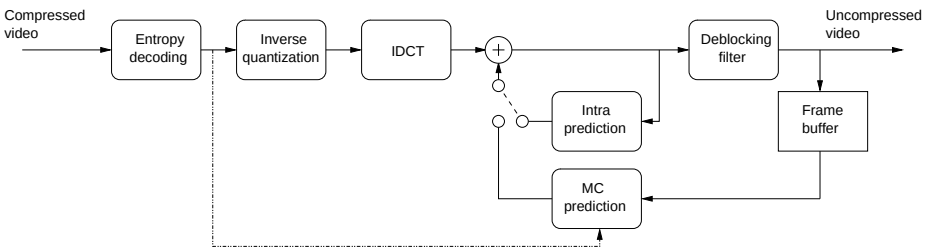


Fig. 1. Block diagram of the decoding process.

Figure 1 depicts the block diagram of the decoding 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 macroblocks (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 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 Macroblock Ordering (FMO) feature allows slice groups to be defined, consisting 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.

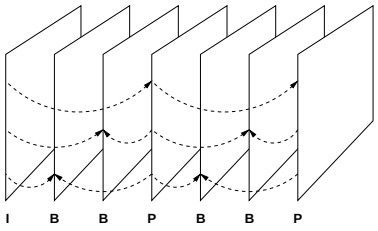


Fig. 2. A typical slice/frame sequence and its dependencies.

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 [5]. Figure 2 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 target mobile and internet streaming applications. For more details on the H.264 standard the interested reader is referred to [6, 4].

3 Benchmark

Throughout this paper we use the HD-VideoBench [7], which provides movie test sequences, an encoder (x264), and a decoder (FFmpeg). 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 were 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 was done with the x264 encoder using the following options: 2 B-frames between I and P frames, maximally 16 reference frames, weighted prediction, hexagonal motion estimation algorithm with maximum search range 24, one slice per frame, and adaptive block size transform.

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 [8, 9]. Thus, it is necessary to exploit parallelism. The H.264 codec can be parallelized either by a task-level or data-level decomposition.

In a *task-level decomposition* the functional partitions of the algorithm are assigned to different processors. For example, Inverse Quantization (IQ) and the IDCT can be done in parallel with the Motion Compensation (MC) stage. The main drawbacks 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. Scalability is a problem because it is limited to the number of tasks, which typically is small.

In a *data-level decomposition* the work (data) is divided into smaller parts and each assigned to a different processor. 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: Group-of-Pictures (GOP) level, frame-level, slice-level, macroblock-level, and block-level. The most important of these will be explained next.

Frame-Level Parallelism As shown in Figure 2 in a sequence of I-B-B-P frames some frames are used as reference for other frames (like I and P frames) but some frames (B frames) are not used as references. That means that B frames can be processed in parallel [10]. In this case, a control processor can assign independent frames to different processors. Frame-level parallelism has good load balancing but scalability problems. This is 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. In such a case, there is no or little parallelism available.

Slice-Level Parallelism In H.264 and in most current hybrid video coding standards each picture is partitioned into one or more slices. As slices are com-

pletely independent from each other they can be processed in parallel without dependency or ordering constraints [10–12]. However, as the number of slices is determined by the encoder, there is a scalability and load balancing problem if there is no control of what the encoder does. More important, increasing the number of slices per frame increases the bitrate for the same quality level. Going from 1 to 64 slices per frame increases the bitrate up to 34%.

Macroblock-Level Parallelism To exploit parallelism between MBs it is necessary to take the dependencies between them into account. In H.264, motion vector prediction, intra prediction, and the deblocking filter use data from neighboring MBs defining a complex set of dependencies (shown as arrows in Figure 3). 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, to distinguish it from the 3D-Wave proposed in this paper.

Figure 3 depicts an example of the 2D-Wave 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 number of independent MBs in each frame depends on the resolution. For a low resolution like QCIF there are only 6 independent MBs during 4 time slots. For High Definition (1920x1088) there are 60 independent MBs during 9 time slots. Figure 4 depicts the available MB parallelism over time for an FHD resolution frame, assuming that the time to decode a MB is constant.

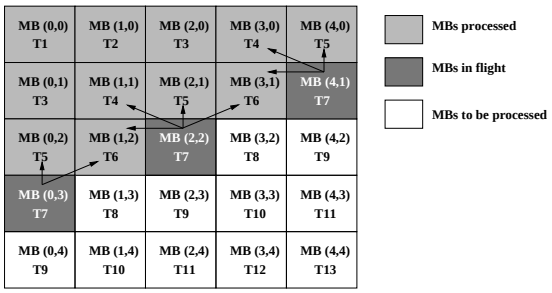


Fig. 3. 2D-Wave approach for exploiting MB parallelism. The arrows indicate dependencies.

MB-level parallelism 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. 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.

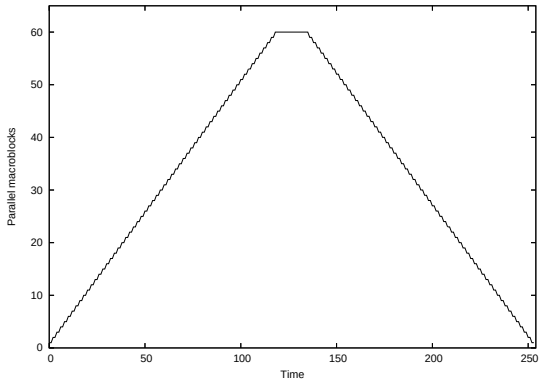


Fig. 4. MB parallelism for a single FHD frame using the 2D-Wave approach.

However, MB-level parallelism has some disadvantages. The first one is that the entropy decoding can not be parallelized using data decomposition, due to the fact that the lowest level of data that can be parsed from the bitstream are slices. Only after entropy decoding has been performed the parallel processing of MBs can start. This disadvantage can be overcome by using special purpose instructions or hardware accelerators for the entropy decoding process. The second disadvantage is that the number of independent MBs does not remain constant during the decoding of a frame (see Figure 4). Therefore, it is not possible to sustain a certain processing rate during the decoding of a frame. Using the parallelization strategy we propose in Section 5, this problem is overcome.

MB-level parallelism has been proposed in previous work. Van der Tol et al. [13] have proposed the exploitation of MB-level parallelism and suggested the combination with frame-level parallelism. Chen et al. [10] have evaluated a similar approach. In the above mentioned works the exploitation of frame-level parallelism is limited to two consecutive frames and the identification of independent MBs is done statically by taking into account the limits of the motion vectors.

5 3D-Wave Strategy

None of the single approaches described in the previous section scales to future many-core architectures containing 100 cores or more. In this work, we propose a parallelization strategy that combines MB-level with frame-level parallelism and which, in contrast to any other work, reveals the large amount of parallelism required to effectively use future many-core CMPs.

In the decoding process the dependency between frames is in the Motion Compensation (MC) module only. Motion Compensation 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). This motion vector in theory can be very long, but in practice the motion estimation algorithm defines a maximum search range of dozens of pixels, because a larger search range is very computationally demanding and provides only a small benefit [14].

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 dynamically be started after the reference areas of the reference frames are decoded. Figure 5 illustrates this way of parallel decoding of frames, called 3D-Wave strategy.

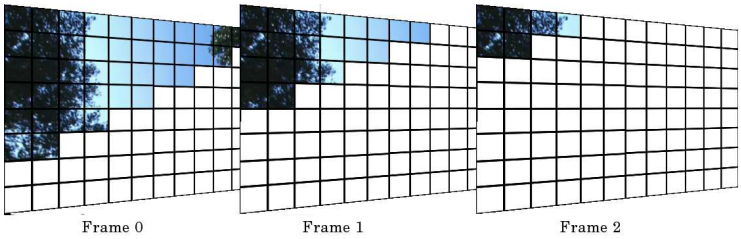


Fig. 5. 3D-Wave strategy: frames can be decoded in parallel because inter frame dependencies have limited spatial range.

To evaluate the 3D-Wave strategy, the FFmpeg H.264 decoder was modified to analyse the dependencies of each MB and assign 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[4], 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.

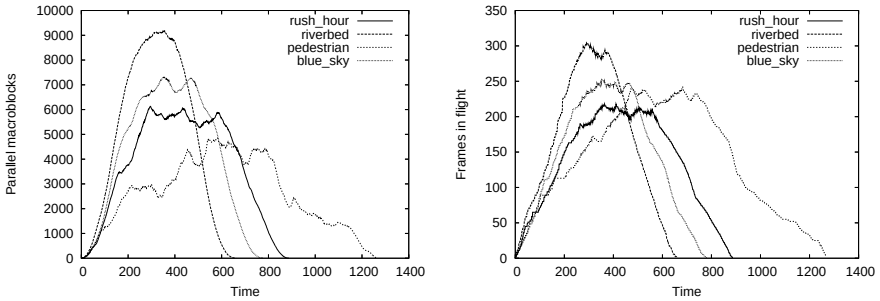
5.1 Maximum Available Parallelism

To start, the 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.

[4] The decoding order of frames is not equal to display order. The sequence I-B-B-P as in Figure 2 is decoded as I-P-B-B sequence.

Table 1. Maximum MB parallelism, and frames in flight for SD, HD, and FHD resolution. Also the average motion vectors (in square pixels) are stated.

	SD			HD			FHD		
	MBs	frames	avg mv	MBs	frames	avg mv	MBs	frames	avg mv
rush_hour	1202	93	1,3	2831	139	1,8	6133	218	2,2
riverbed	1944	150	2	4579	228	2,2	9169	304	2,6
pedestrian	1227	104	9,2	2807	151	11	4851	242	9,9
blue_sky	1298	97	4,4	2873	140	5,1	7327	253	5,6



(a) (b)

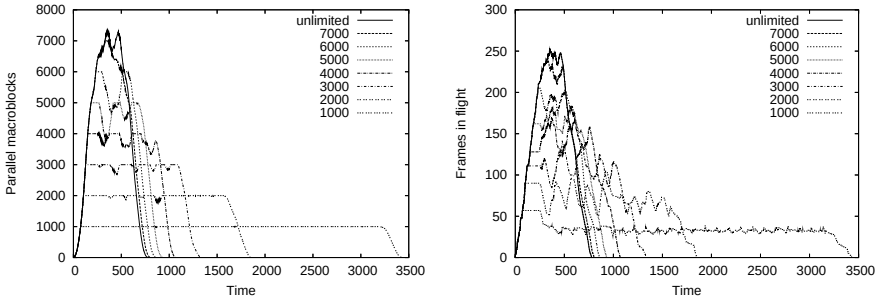
Fig. 6. Number of parallel MBs (a) and frames in flight (b) for FHD resolution using a 400 frames sequence.

Table 1 summarizes the results and shows that the 3D-Wave approach exhibits a huge amount of MB parallelism. For FHD resolution the parallelism ranges from 4851 to 9169 MBs. Figure 6(a) depicts the MB parallelism time curve for FHD, while Figure 6(b) 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 rush_hour the plateau starts at time slot 300 and last until time slot 650. Riverbed exhibits so much MB parallelism that is has a small plateau. Due to the stochastic nature of the movie, the encoder mostly uses intra-coding, resulting in very few dependencies between frames. Pedestrian exhibits the least parallelism. The fast moving objects in the movie result in many large motion vectors. Especially objects moving from right to left on the screen causes large offsets between MBs in consecutive frames.

Limited Resources The analysis above reveals that the 3D-Wave strategy provides 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 many-cores the hardware resources (cores, memory, and NoC bandwidth) will be limited. We now investigate the impact of resource limitations on the 3D-Wave strategy.

We model limited resources as follows. A limited number of cores is modelled by limiting the number of MBs in flight. Memory requirements of the 3D-Wave strategy is mainly related to the number of frames in flight. Thus limited memory is modelled by restricting the number of frames that can be in flight concurrently. Limited NoC bandwidth is captured by both modelled 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 and for all three resolutions. The results are similar, thus only the results for the blue_sky movie at FHD resolution is presented.

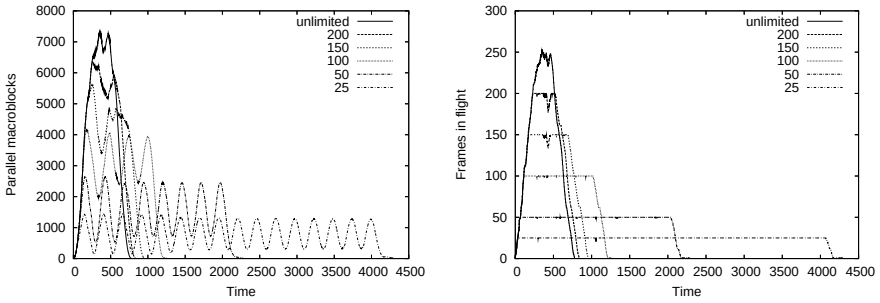


(a) (b)

Fig. 7. Available MB parallelism (a) and number of frames in flight (b) for FHD blue_sky, for several limits of the number of MBs in flight.

First, the impact of limiting the number of MBs in flight is analyzed. Figure 7(a) 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 utilization rate of the available cores. Furthermore, the figure shows that the decoding time is approximately linear in the limit on MBs in flight. Figure 7(b) depicts the number of frames in flight as a function of time when the number of MBs in flight is limited. Not surprisingly, because limiting the number of MBs in flight limits the number of frames that can be in flight, the shape of the curves are similar to those of the available MB parallelism, although with a small periodic fluctuation.

Next, we analyze the impact of restricting the number of frames concurrently in flight. The MB parallelism curves are depicted in Figure 8(a) and shows 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 next frame, however, has to wait until the frame currently being processed



(a) (b)

Fig. 8. Available MB parallelism (a) and number of actual frames in flight (b) for FHD blue_sky for several limits of the number of frames in flight.

is finished. Figure 8(b) shows the actual frames in flight for several limits. The plateaus in the curves translate to full memory utilization.

From this experiment we conclude that for systems with limited number of cores the 3D-Wave approach 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 the next section we perform a case study, indicating that memory will likely not be a bottleneck.

6 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 the 3D-Wave strategy.

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 as it is reasonable to expect that the size of mobile devices will not significantly grow. CMPs are power efficient, and thus we expect mobile devices to adopt them soon and project a 100-core CMP in 2015. Using Moore's law for memory we estimate that 1,28GB of main memory is available. 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.

Figure 9(a) presents the MB parallelism for these assumptions as well as for unlimited number of cores. 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. Figure 9(b) depicts the number of frames in flight. The average is approximately 12 frames with peaks of up to 16 except for riverbed which has a peak in the beginning of 23.

From this case study we conclude that even a low resolution movie exhibits sufficient parallelism to fully utilize 100 cores efficiently. Furthermore, for mobile devices, memory will likely not be a bottleneck.

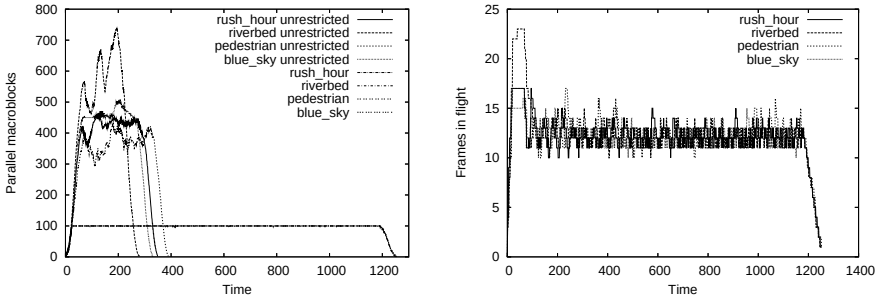


Fig. 9. Number of parallel MBs (a) and frames in flight (b) for the mobile video case study. Also depicted in (a) is the available MB parallelism for the same resolution but with unrestricted resources.

7 Conclusions

In this paper we have investigated if future applications exhibit sufficient parallelism to exploit the large number of cores expected in future CMPs. As a case study, we have analyzed the parallel scalability of the H.264 video decoding process, currently the best coding standard.

Previously proposed parallelization strategies such as slice-level, frame-level, and intra-frame macroblock (MB) level parallelism, are not sufficiently scalable. Therefore, we have proposed a novel parallelization strategy, called 3D-Wave, which combines MB parallelism with frame-level parallelism. Inter-frame dependencies typically have limited spatial range, and thus decoding of frames can be overlapped. Decoding a MB can be started as soon as the MBs upon which it depends have been decoded. Analysis showed that by using this strategy the total number of parallel MBs ranges from 4000 to 9000.

We have also performed a case study to assess the practical value and possibilities of the 3D-Wave strategy. In general, the results show that our strategy provides sufficient parallelism to efficiently exploit the capabilities of future manycore CMPs. Although we have focussed 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 many-core CMPs.

Acknowledgments

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. Stenström, P.: Chip-multiprocessing and Beyond. In: Proc. Twelfth Int. Symp. on High-Performance Computer Architecture. (2006) 109–109
2. Asanovic, K., et al.: The Landscape of Parallel Computing Research: A View from Berkeley. Technical Report UCB/EECS-2006-183, EECS Department University of California, Berkeley (December 2006)
3. Oelbaum, T., et al.: Subjective Quality Assessment of the Emerging AVC/H.264 Video Coding Standard. In: Int. Broadcast Conference (IBC). (2004)
4. : International Standard of Joint Video Specification (ITU-T Rec. H. 264—ISO/IEC 14496-10 AVC) (2005)
5. Flierl, M., Girod, B.: Generalized B Pictures and the Draft H. 264/AVC Video-Compression Standard. IEEE Trans. on Circuits and Systems for Video Technology **13**(7) (2003) 587–597
6. Wiegand, T., Sullivan, G.J., Bjontegaard, G., A.Luthra: Overview of the H.264/AVC Video Coding Standard. IEEE Trans. on Circuits and Systems for Video Technology **13**(7) (July 2003) 560–576
7. Alvarez, M., Salami, E., Ramirez, A., Valero, M.: HD-VideoBench: A Benchmark for Evaluating High Definition Digital Video Applications. In: IEEE Int. Symp. on Workload Characterization. (2007)
8. Alvarez, M., Salami, E., Ramirez, A., Valero, M.: A Performance Characterization of High Definition Digital Video Decoding using H.264/AVC. In: Proc. IEEE Int. Workload Characterization Symposium. (2005) 24–33
9. Ostermann, J., Bormans, J., List, P., Marpe, D., Narroschke, M., Pereira, F., Stockhammer, T., Wedi, T.: Video Coding with H.264/AVC: Tools, Performance, and Complexity. IEEE Circuits and Systems Magazine **4**(1) (2004) 7–28
10. Chen, Y., Li, E., Zhou, X., Ge, S.: Implementation of H. 264 Encoder and Decoder on Personal Computers. Journal of Visual Communications and Image Representation **17** (2006)
11. Roitzsch, M.: Slice-Balancing H.264 Video Encoding for Improved Scalability of Multicore Decoding. In: Work-in-Progress Proc. 27th IEEE Real-Time Systems Symposium. (2006)
12. Rodriguez, A., Gonzalez, A., Malumbres, M.P.: Hierarchical parallelization of an h.264/avc video encoder. In: Proc. Int. Symp. on Parallel Computing in Electrical Engineering. (2006) 363–368
13. van der Tol, E., Jaspers, E., Gelderblom, R.: Mapping of H.264 Decoding on a Multiprocessor Architecture. In: Proc. SPIE Conf. on Image and Video Communications and Processing. (2003)
14. Liu, Y., Oraintara, S.: Complexity Comparison of fast Block-Matching Motion Estimation Algorithms. In: Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing. (2004)