High Efficiency Video Coding (HEVC)

Kiana Calagari
School of Computing Science
Simon Fraser University
Surrey, BC, Canada
kcalagar@cs.sfu.ca

I. Introduction

The High Efficiency Video Coding (HEVC) standard (unofficially known as H.265) is a new video coding standard which its first edition is expected to be finalized in January 2013. It is a successor to H.264/AVC and it doesn't have any of the Scalable Video Coding or Multiview extensions yet, but the draft standard does contain provisions for future extensions similar to these. For now it only has a Main profile which is kind of similar to the High profile in H.264. The HEVC test model HM software is the reference software for this standard and HM-8.0 is the latest version of it. HEVC has its focus on particularly two key issues: Doubling the coding efficiency so it would be suitable for applications with increased video resolutions (but not in a way that would cause an extreme increase in complexity) and also making it suitable for parallel processing architectures. I read the wiki page for HEVC [1] and had a look at two papers about it, for more details in some specific parts [2], [3]. Compered to H.264/AVC it is kind of improved in almost every part but I have mainly focused in some particular parts.

II. Features and Improvements

Coding Tree Structure:

One of the most beneficial improvements that has a big effect in coding efficiency, is the use of CTUs (Coding Tree Units) in HEVC, instead of the MBs (Macro Blocks) in H.264. The core of the coding layer in H.264 was a MB containing a 16x16 block, but in HEVC it's a CTU which its size is selected by the encoder according to its architectural characteristics and the needs of its application environment and can be larger than a traditional MB.

In HEVC, a picture is partitioned into coding tree blocks (CTBs). A luma CTB covers a rectangular picture area of NxN samples of the luma component and the corresponding chroma CTBs cover each

(N/2)x(N/2) samples of each of the two chroma components. The value of N is signaled inside the bitstream, and can be 16, 32, or 64. larger sizes of N typically result a better compression. The luma CTB and the two chroma CTBs, together with the associated syntax, form a coding tree unit (CTU), which as mentioned before is the basic processing unit of the standard to specify the decoding process. Furthermore, CTBs can be partitioned into multiple coding blocks (CBs). The quadtree syntax of CTU specifies the size and position of its CBs. The size of the CB can range from the same size as the CTB to a minimum size of 8x8 luma samples. The luma CB and the chroma CBs, together with the associated syntax, form a coding unit (CU). So a CTU may contain only one CU or may split to multiple CUs.
The decision whether to code a picture area using inter or intra prediction is made at the CU level.
Depending on the basic prediction type decision the CBs can be also splited to form the Prediction Blocks (PBs). HEVC supports variable PB sizes from 64x64 to 4x4 samples. Again, the luma and chroma PBs, together with the associated syntax, form a prediction unit (PU). A PU has its root at the CU level. Each PU contains one motion vectors for uni-predictive or two motion vectors for bi-predictive coding. All PBs of a CB can have the same size, or, when asymmetric motion partitioning (AMP) is used, a luma CB of size NxN can also be split into two luma PBs, where one of the luma PBs covers Nx(N/4) or (N/4)xN samples and the other luma PB covers the remaining Nx(3N/4) or (3N/4)xN area of the CB.
It has been shown that when the encoder was forced to use smaller CTB sizes, for all test videos, the HEVC bitrate increased by 2.2% when forced to use a 32x32 CTB size and increased by 11.0% when forced to use a 16x16 CTB size, compared to a 64x64 CTB size. Also when the resolution of the video was 2560x1600, it was shown that the HEVC bitrate increased by 5.7% when forced to use a 32x32 CTB size and increased by 28.2% when forced to use a 16x16 CTB size, again compared to a 64x64 CTB size. According to these tests it was shown that large CTB sizes are even more important for coding efficiency when higher resolution video are used. Also it was shown that it takes a longer time to decode a video encoded at small CTB sizes than at a 64x64 CTB size. So large CTB sizes increase coding efficiency while also reducing decoding time.
For transforming the prediction residuals, HEVC uses Transform Blocks (TBs) and Transform Units (TUs). A TU also has its root at the CU level. TBs can have square sizes of 4x4, 8x8,16x16 and 32x32.

Intra prediction:

HEVC supports 33 directional modes for intra prediction (compared to 8 such modes in H.264/AVC) plus planar and DC prediction. So there are 35 spatial intra prediction modes for a CB. When the luma CB has reached its smallest allowable size, it is also possible to signal one intra prediction mode for

each of its four square sub-blocks.

Motion Vector coding:

The coding of motion parameters has faced a lot of improved compared to H.264. There are two methods for MV prediction:

Merge Mode: HEVC supports a so called merge mode, in which no motion parameters are coded. Instead, a candidate list of motion parameters is made for the corresponding PU and the index information for selecting one of the candidates is transmitted. In general, the candidate list includes the motion parameters of spatially neighboring candidates as well as a temporally candidate and generated candidates. The usage of large block sizes for motion compensation and the merge mode, allow a very efficient coding for large consistently displaced picture areas.

Advanced Motion Vector Prediction (AMVP): If a inter coded CB is not coded using the merge mode, the motion vector is differentially coded using a motion vector predictor. Prediction is done using the advanced motion vector prediction (AMVP) algorithm. Similar to the merge mode, HEVC allows the encoder to choose the predictor among multiple candidates. In AMVP, for each motion vector, a candidate list (consisting of two candidates) is constructed, and the difference between the chosen predictor and the actual motion vector is transmitted along with the index of the chosen candidate.

Parallel Processing Tools :

Other than slices, three new features are introduced in HEVC for enhancing parallel processing capability.

Tiles:

In addition to slices, HEVC also defines tiles, which are self-contained and independently decodable rectangular regions of the picture. The main purpose of tiles is to enable the use of parallel processing architectures for encoding and decoding. Tiles provide parallelism at a more coarse level (picture/sub-picture) of granularity.

Multiple tiles may share header information by being contained in the same slice. Alternatively, a single tile may contain multiple slices. A tile consists of a rectangular arranged group of CTUs (typically, but not necessarily, with all of them containing about the same number of CTUs).

Wavefront parallel processing (WPP):

When Wavefront Parallel Processing (WPP) is enabled, a slice is divided into rows of CTUs. The decoding

of each row can be begun as soon a few decisions that are needed for prediction and adaptation of the entropy coder have been made in the preceding row. This supports parallel processing of rows of CTUs by using several processing threads in the encoder or decoder. For design simplicity, WPP is not allowed to be used in combination with tiles.
WPP provides a form of processing parallelism at a rather fine level of granularity, i.e. within a slice. WPP may often provide better compression performance than tiles.

Dependent Slices:
Divides each frame to slices that are each packetized in a separate NAL unit but each slice is dependent and can only be decoded after at least part of the decoding process of the previous slice has been performed. Dependent slices are mainly useful for ultra low delay applications such as remote surgery and it can reduce the delay by starting the transmission of a coded data earlier than if we wanted to wait for the whole frame to get encoded (it sends a slice as soon as it gets encoded, and moves to the next slice). So error resiliency gets worst but it still has high coding efficiency (unlike normal slices)[4]. So I guess it's kind of the same as letting the frame get fragmented in lower network layers (which has a high coding efficiency and an awful error resiliency), but with low delay because it doesn't wait for the whole frame to be encoded and sent to the lower network layers.
Also dependent slices are especially relevant to WPP.

After reading these I actually didn't get the difference between tiles and normal slices so I searched the web for the answer and here is what I found:
Tiles vs Slices:
Slices break the bitstream up into packets, with overhead from slice headers and if going over a network, other packet headers. If you are in a tight bitrate situation you can't afford the extra bits. Also in these environments you often want to carefully control the slice size to fit within the network packets. That is harder if you are trying to use slices for parallelism. So in HEVC it is better to use slices for controlling packet sizes and one of the other tools for parallelism.
For the same degree of parallelism, tiles have fewer boundaries where prediction is broken, and fewer wasted bytes in headers. Tiles are kind of a zero overhead slices which need to send tile information typically once for sequence, based on the resolution. Slice header needs to be sent at every slice and it can constitute to sufficient overhead to the bitrate.
Tiles also allow for the creation of meaningful rectangular regions of interest unlike slices, which only

partition an image in coding unit raster scan order. Since FMO is no longer in the HEVC standard! because depending on which slice group map you use, FMO breaks prediction all over the place so reduces efficiency a lot. It was intended for error resiliency in earlier standard where prediction was less useful, e.g. Intra just predicting dc terms from the last mb in coding order. (so now in case of having ROI tile can take the place of FMO).

Tails are tailored for multiple cores architecture. So for example in dual-cores frames are divided into two tiles (more than two tiles for dual-cores are not coding efficient due to break of dependencies across tile borders). And since in some applications it's required to divide frames into slices either to match to MTU size or to enhance error resilience, in such scenarios each tile should be divided into a number of slices.[5]

Tiles vs WPP:

Tiles, break up the picture into isolated processing regions, WPP, interleaves the processing units, but requires frequent communication between processing units.

Its hard to match MTU sizes using WPP. And since some use cases do have MTU matching constrains, tiles are excellent for them since you can encode in parrallel first, then group tiles according to the MTU sizes.

While WPP does look good for many use cases, it struggles to get full utilization of cores when you have a high number of cores e.g. 15 for 1080p where you can't start the last row until you've finished the first. But if you have a relatively small number of cores with good inter core communication, no need to match MTU sizes, and a big enough shared cache then WPP is a good solution.[6]

Other features:

There are also some other features and improvments in HEVC. For example speeding up its CABAC algorithm in the Entropy part, Adding a loop filter named SAO in addition to the deblocking filter,...

Similar to H.264/MPEG-4 AVC, HEVC supports quarter-sample precision motion vectors. HEVC also supports multiple reference pictures, and the concepts of I, P, and B slices are basically unchanged from H.264/MPEG-4 AVC. Weighted prediction is also supported in a similar manner. Similar concepts of network abstraction layer (NAL) and high-level syntax (such as frame buffer management, sequence and picture parameter sets) are used as in the AVC standard and a new video parameter set (VPS) has also been added.

REFERENCES

[1] "High efficiency video coding," http://en.wikipedia.org/wiki/High-Efficiency-Video-Coding.

[2] J.-R. Ohm, G. J. Sullivan, H. Schwarz, and T. K. T. T. Wiegand, "Comparison of the coding efficiency of video coding standards–including high efficiency video coding (hevc)," *IEEE Trans. on Circuits and Systems for Video Technology*, December 2012.

[3] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand, "Overview of the high efficiency video coding (hevc) standard," *IEEE Trans. on Circuits and Systems for Video Technology*, December 2012.

[4] http://www.linkedin.com/groups/Feedback-from-9-th-HEVC-3724292.S.111535682.

[5] http://www.linkedin.com/groups/I-am-littile-confused-about-3724292.S.113293985.

[6] http://www.linkedin.com/groups/If-you-had-choice-between-3724292.S.109622180.