COMPARISON OF DIFFERENT PARALLEL IMPLEMENTATIONS FOR DEBLOCKING FILTER OF HEVC

Anand Meher Kotra, Mickaël Raulet, Olivier Deforges

IETR-INSA, UEB, UMR 6164
Rennes, 35708, France
{vkotra, mraulet, odeforge}@insa-rennes.fr

ABSTRACT

Unlike deblocking filter of H.264/AVC, deblocking filter of HEVC is computationally less complex and offers more parallelization possibilities. In this paper we present comparison of three different parallelization implementations of deblocking filter which operate on picture by picture basis. The first method filters the vertical edges and horizontal edges in separate passes. The other two methods combine the vertical and horizontal edge filtering in a single pass. On a 6 core machine with 6 threads running concurrently, experimental results showed an average accelerating factor of 4.5, 5 and 5 for each of the implementation methods on 1080p, 1600p and 2160p sequences respectively.

Index Terms— HEVC, Deblocking Filter, Parallelization

1. INTRODUCTION

High Efficiency Video Coding standard (HEVC) is the latest video coding standard under development by the Joint Collaborative Team On Video Coding (JCT-VC) [5][6][10]. Similar to H.264/AVC [17], HEVC uses block based prediction. The resulting block based prediction error is quantized and then coded. The blocking artifacts which result due to quantization are typically filtered using deblocking filters [7]. In addition to deblocking filter (DBF), HEVC introduces a new in loop filtering stage called as SAO (Sample Adaptive Offset) [8]. SAO filter is located after the deblocking filter stage and is used to filter the ringing artifacts. Typically HEVC decoding can be categorized into three stages. The first stage is the entropy decoding and picture reconstruction stage (CTB (Coding Tree Block) or LCU (Largest Coding Unit) decode stage), where the bit stream is parsed and slice data is decoded. The second stage is the DBF. The third stage is SAO filtering.

Compared to the DBF of AVC, DBF of HEVC is computationally less complex and offers more parallelization possibilities. This is mainly due to the fact that DBF in HEVC is performed over an grid of 8x8 samples whereas in AVC, DBF is performed over each grid of 4x4 samples. Similar to AVC, DBF in HEVC also modifies a maximum of three samples on either side of the edge. By exploiting the new parallelization possibilities, we propose three different parallel implementations of DBF in HEVC. All the three methods operate with the whole reconstructed picture as input. The first method parallelizes the vertical edge filtering and horizontal edge filtering in separate passes. The other two methods combine the vertical edge filtering and horizontal edge filtering in a single pass.

The parallel DBF methods are implemented by modifying the DBF of HM-8.0 reference software [9]. On a 6 core machine with 6 threads running concurrently, experimental results showed an average accelerating factor (AF) or speedup factor close to the number of cores for all three methods. A summary of the rest of the paper is as follows : Section 2 presents the related work. Section 3 explains the different parallel DBF implementation methods. Section 4 gives the evaluation and results. Section 5 concludes the paper.

2. RELATED WORK

[1] proposes parallel HEVC decoders using Wavefront Parallel Processing (WPP) and Tiles [10]. WPP and Tiles are two new tools introduced in HEVC standard to facilitate efficient parallelization of the LCU decode stage. The WPP decoder of [1] and the decoder presented in [2] parallelize all the three stages (LCU decode, DBF and SAO) in a single LCU decode loop. The Tiles decoder of [1] parallelizes all the three stages in a separate pass. In Tiles decoder of [1], DBF is parallelized by performing the vertical edge filtering and horizontal edge filtering in separate passes.

In our paper we propose three parallelization methods for DBF. The first method is similar to the DBF parallelization used in Tiles decoder of [1], but the way LCUs are assigned to threads, to perform DBF parallely differs. The second and the third method combine the vertical and horizontal edge filtering in a single pass. The paper then compares the accelerating factors achieved by each of the implementation methods.

This work is done as part of 4EVER, a French national project with support from Europe (FEDER), French Ministry of Industry, French Regions of Brittany, Ile-de-France and Provence-Alpes-Cote-d'Azur, Competitivity clusters Images&Reseaux (Brittany), Cap Digital (Ile-de-France) and Solutions Communicantes Sécurisées (Provence-Alpes-Cote-d'Azur).

3. EXPLANATION OF DIFFERENT PARALLELIZATION APPROACHES

From the perspective of the decoder the steps for deblocking are as follows:

- Perform horizontal filtering of all vertical edges in the picture. The different steps in horizontal filtering are:
  - calculate the boundary strength (bs) of all the prediction units edges and the transform unit edges lying on the 8x8 grid.
  - if (bs>0), then for each four-sample part of the 8x8 block, perform the filtering decisions (weak filter, strong filter or no filter) and apply the filtering on the samples.
- Perform vertical filtering of all horizontal edges in the picture. The filtering steps are similar to the horizontal filtering. However the modified samples after horizontal filtering are used as input to the vertical filtering.

Profiling of HM-8.0 decoder is conducted to derive the $\%timespent$ in the DBF stage. Table 1 shows the sequences used for profiling and break down of execution times for the DBF stage. For 1080p and 1600p we used the test sequences described in the HEVC common conditions [15]. Sequences under [16] are used for 2160p sequences.

Res	Seq	QP	Config	BR(Mbps)	Deblocking%
1080p50	BasketBallDrive	22	Intra	71.333	12.26
		37	Intra	8.543	14.76
		22	RandomAccess	17.453	11.27
		37	RandomAccess	1.527	11.54
1080p60	BQTerrace	22	Intra	180.205	10.08
		37	Intra	21.843	15.30
		22	RandomAccess	39.708	10.82
		37	RandomAccess	1.020	9.79
1080p50	Cactus	22	Intra	105.393	11.63
		37	Intra	14.328	15.61
		22	RandomAccess	18.297	12.64
		37	RandomAccess	1.423	12.87
1080p24	Kimono	22	Intra	22.251	12.10
		37	Intra	3.842	14.01
		22	RandomAccess	4.820	11.52
		37	RandomAccess	0.565	11.23
1080p24	ParkScene	22	Intra	52.712	11.59
		37	Intra	7.315	15.44
		22	RandomAccess	7.685	11.69
		37	RandomAccess	.740	11.24
1600p60	NebutaFestival	22	Intra	402.598	6.50
		37	Intra	81.510	13.50
		22	RandomAccess	216.496	5.97
		37	RandomAccess	7.148	10.92
1600p60	SteamLocomotive	22	Intra	100.196	13.22
		37	Intra	14.485	14.42
		22	RandomAccess	23.687	12.56
		37	RandomAccess	1.246	11.60
1600p30	PeopleOnStreet	22	Intra	104.706	13.10
		37	Intra	20.455	17.53
		22	RandomAccess	32.850	15.08
		37	RandomAccess	4.666	16.75
1600p30	Traffic	22	Intra	101.828	13.08
		37	Intra	18.516	16.37
		22	RandomAccess	13.189	13.10
		37	RandomAccess	1.351	12.09
2160p50	ParkJoy	22	Intra	635.207	10.18
		37	Intra	92.469	14.64
		22	RandomAccess	191.823	11.10
		37	RandomAccess	13.602	12.53
2160p30	PeopleOnStreet4K	22	Intra	479.112	10.78
		37	Intra	149.474	14.42
		22	RandomAccess	344.332	11.79
		37	RandomAccess	84.695	13.70

Table 1: Profiling of sequences using HM-8.0 decoder.

The DBF stage consumes around 10-17% of the whole decoding time. Therefore efficient parallelization of the DBF stage can reduce the overall decoding time of the picture by a good factor. Explanation of different parallel DBF implementation methods follows:

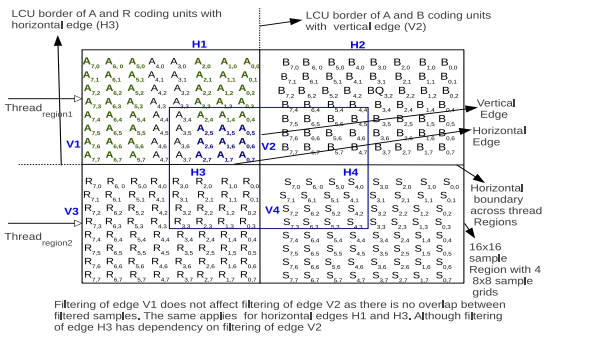

Fig. 1: Deblocking filter sample dependencies across LCU borders and thread regions

For all the parallel DBF methods, consecutive LCU rows are equally divided among threads to perform DBF parallely, as depicted in subfigure A of Figure 2 and Figure 3, Figure 4 respectively. The number of LCU rows assigned to each thread is:
$(Totalnumber\ of\ LCURows/Number\ of\ Threads)$. If the remainder of the division is an odd number, the first thread is assigned an extra LCU row. LCU rows 1, 2 together named as Thread$_{region1}$ are filtered by thread1, LCU rows 3, 4 (Thread$_{region2}$) are filtered by thread2 and so on. LCUs are processed in raster scan order inside each Thread$_{region}$.

The dependencies for performing DBF parallely are depicted in Figure 1. The figure depicts one 16x16 sample grid across four LCU borders, spanning two Thread$_{regions}$. The DBF modifies a maximum of three samples on either side of the edge as depicted for the samples of LCU A in Figure 1. Therefore vertical edge filtering of one edge does not affect the filtering of other vertical edges and therefore can be carried out in parallel. The same applies for filtering of horizontal edges. Although as depicted in Figure 1, during the filtering of vertical edge V2, the samples $A_{2,5}$, $A_{1,5}$, $A_{0,5}$, $A_{2,6}$, $A_{1,6}$, $A_{0,6}$, $A_{2,7}$, $A_{1,7}$, $A_{0,7}$ may be modified if strong filtering is chosen. The resulting samples after horizontal filtering of V2 are used for vertical filtering of H3: Therefore, the filtering of horizontal edge H3 cannot be initiated by $thread2$ until $thread1$ finishes the filtering of vertical edge V2. DBF of H.264/AVC operates over an grid of 4x4 samples and therefore results in overlap between filtered samples. Hence unlike DBF of AVC, vertical and horizontal edge filtering phases are implicitly parallelizable in DBF of HEVC.

3.1. Separate filtering (SF)

This method performs the parallel vertical edge filtering and parallel horizontal edge filtering in separate passes to fulfill the dependencies of DBF. Algorithm 1 explains the horizontal DBF of vertical edges. The output picture after horizontal DBF is used as input for vertical DBF of horizontal edges. For vertical DBF, consecutive LCU columns are equally divided among threads as depicted in subfigure B of Figure 2.

Algorithm 2 explains the vertical DBF.

Algorithm 1 Parallel horizontal DBF of vertical edges (Pass1)

```
1:  procedure HOR_FILTER(Num_Threads, Picture)
2:      Start ← 0
3:      Num_Rows ← (Total_LCUrows/Num_Threads)
4:      for i = 1 → Num_Threads do
5:          for j = Start → Num_Rows do
6:              Perform filtering of all vertical edges
7:          end for
8:          Start ← Num_Rows
9:          Num_Rows ← Num_Rows + Num_Rows
10:     end for
11: end procedure
```

Algorithm 2 Parallel vertical DBF of horizontal edges (Pass2)

```
1:  procedure VER_FILTER(Num_Threads, Picture)
2:      Start ← 0
3:      Num_Col ← (Total_LCUCol/Num_Threads)
4:      for i = 1 → Num_Threads do
5:          for j = Start → Num_Col do
6:              Perform filtering of all horizontal edges
7:          end for
8:          Start ← Num_Col
9:          Num_Col ← Num_Col + Num_Col
10:     end for
11: end procedure
```

3.2. Combined Filtering1 (CF1) and Combined Filtering2 (CF2)

In CF1 and CF2 methods, as a first step, the DBF of vertical edges of each Thread$_{region}$ is performed in a concurrent fashion. In Combined filtering1 (CF1), to fulfill DBF dependencies, after $threadN$ finishes the filtering of all the vertical edges in its thread region, it consequently waits for $threadN-1$ to finish its horizontal filtering of vertical edges, before triggering the vertical filtering of all horizontal edges which belong to its own region. This situation is depicted in Figure 3. Although, if more than one LCU row belong to a Thread$_{region}$, then the thread can initiate the vertical filtering of the horizontal edges from its 2nd LCU row onwards. This is due to the fact that DBF operates over each 8x8 sample grid and only modifies a maximum of three samples on either side of the edge. Hence only the first LCU row has dependencies across Thread$_{regions}$.

Therefore in Combined filtering2 (CF2), the filtering of horizontal edges of just the first LCU row of each Thread$_{region}$ is postponed until all the vertical edges of the Thread$_{region}$ above it are filtered. As depicted in Figure 4 , $thread2$ initiates the filtering of horizontal edges in LCU row 4 as soon as all the vertical edges which belong to its own Thread$_{region2}$ are filtered. Hence only the vertical filtering of all horizontal edges of LCUs in LCU row 3 need to wait for Thread$_{region1}$ to finish its horizontal filtering of vertical edges.

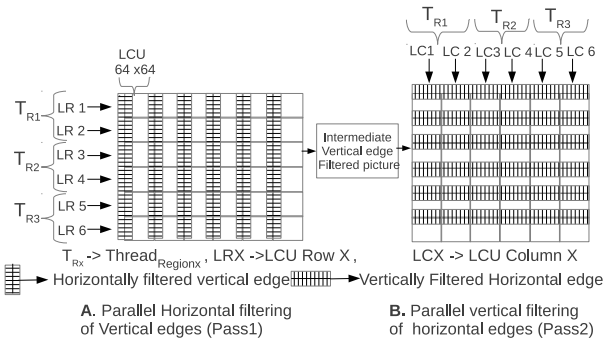

Fig. 2: Parallel separate filtering of horizontal and vertical edges for each picture.

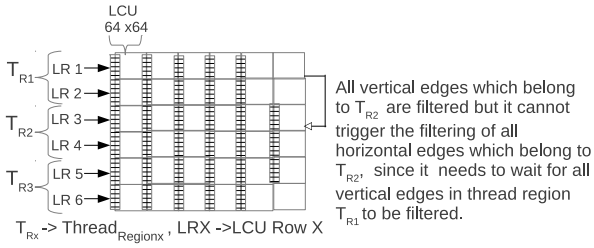

Fig. 3: CombinedFiltering1 (Vertical edges in all Thread$_{regions}$ are processed concurrently. For horizontal edge filtering, Thread$_{regionN}$ has to wait for Thread$_{regionN-1}$ to finish filtering of vertical edges).

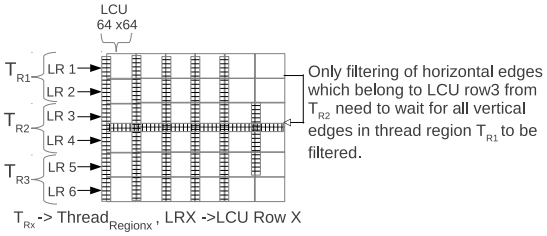

Fig. 4: CombinedFiltering2 (Vertical edges in all Thread$_{regions}$ are processed concurrently. For horizontal edge filtering, only filtering of horizontal edges which belong to first LCU row from Thread$_{regionN}$ need to wait for Thread$_{regionN-1}$ to finish filtering of vertical edges).

4. RESULTS

The parallel deblocking filter methods are implemented on top of the DBF present in HM-8.0 decoder. The sequences in

Table 1 are used to evaluate the decoder. The configuration of the test machine is as follows:

- Linux 64 bits, gcc version:4.6.3, Intel(R) Xeon(R) CPU W3670 @ 3.20GHz, 6 cores on single die (Turbo boost off), cache size: 12288 KB.
- Pthread library is used for multi threading implementations.

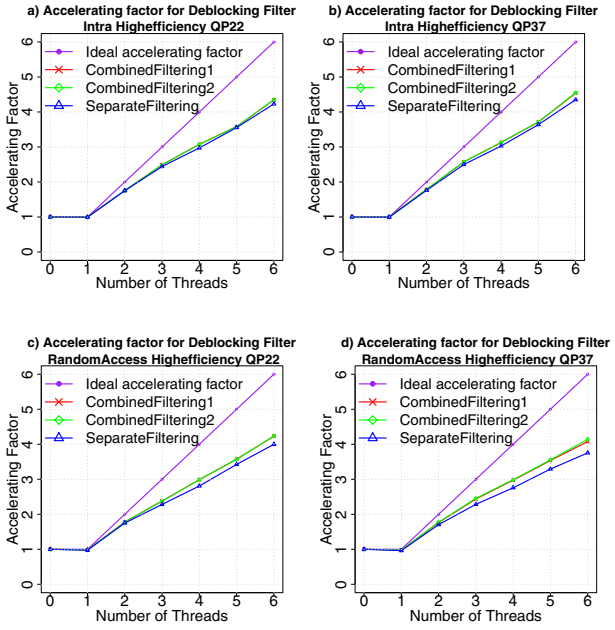

Fig. 5: Accelerating factors of different parallel DBF implementations for HD (1080p) sequences.

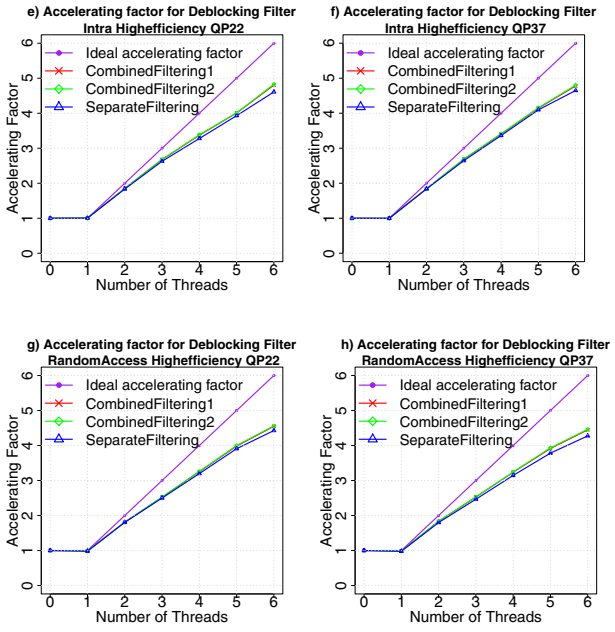

Fig. 6: Accelerating factors of different parallel DBF implementations for 1600p sequences.

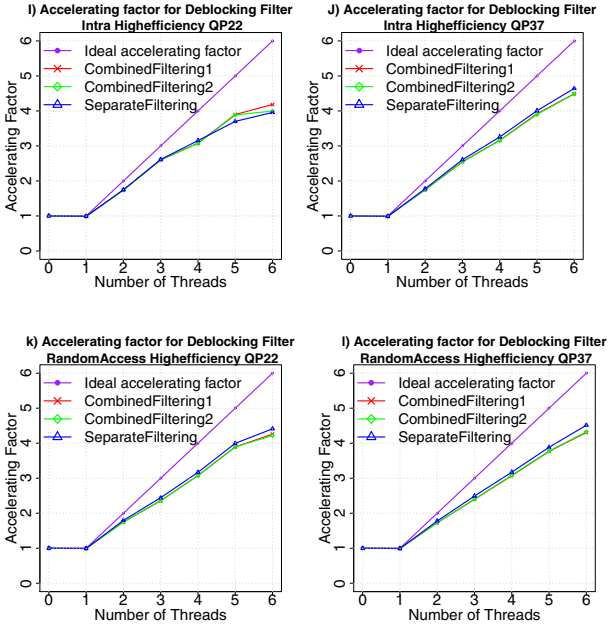

Fig. 7: Accelerating factors of different parallel DBF implementations for 2160p sequences.

The average accelerating factors (AF) for each of the methods are ascertained. The AFs are measured with reference to the serial DBF of HM-8.0 decoder. (value 0 on x-axis). The AFs for 1080p sequences are depicted in subfigures a, b, c, d of Figure 5. Combined Filtering1 (CF1) and Combined Filtering2 (CF2) methods perform slightly better when compared to Separate filtering (SF) method which is similar to the parallel DBF approach proposed in Tiles decoder of [1]. AFs of CF2 are slightly better when compared to CF1 for randomaccess QP37 sequences (subfigure d). AFs for 1600p and 2160p sequences are depicted in the Figure 6 and Figure 7 respectively. The results for 1600p and 2160p sequences follow similar pattern as that of 1080p sequences. To summarize the results : On a 6 core machine with 6 threads running concurrently, experimental results show an AF of 4.5, 5 and 5 for each of the implementation methods on 1080p, 1600p and 2160p sequences respectively.

5. CONCLUSION

In this paper we presented comparison of three different parallelization implementations of deblocking filtering in HEVC which operate with whole picture as input. One of the methods performs the vertical filtering and horizontal filtering in separate passes. The other two proposed methods perform the vertical filtering and horizontal filtering in a single pass. On a 6 core machine with 6 threads running concurrently all the three methods achieve accelerating factors close to the number of cores.

6. REFERENCES

[1] C.C. Chi, M.A. Mesa, B. Juurlink, G. Clare, F. Henry, S. Pateux, T. Schierl, "Parallel Scalability and Efficiency of HEVC Parallelization Approaches", IEEE Transactions on Circuits and Systems for Video Technology, IEEE TCSVT, Dec 2012.

[2] M.A. Mesa, C.C. Chi, B. Juurlink, V. George, T. Schierl, "Parallel Video Decoding In The Emerging HEVC Standard", Proceedings of International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2012), Kyoto, Japan, March 2012.

[3] C. C. Chi, M. Alvarez-Mesa, V. George, B. Juurlink, T. Schierl, "Parallel Scalability and Efficiency of WPP and Tiles", Tech. Rep. JCTVC-I0520, April 2012

[4] G. Clare, F. Henry, S. Pateux "Wavefront Parallel Processing for HEVC Encoding and Decoding", Tech. Rep. JCTVC-F274, July 2011.

[5] G.J. Sullivan, J.R Ohm, W.J. Han, T. Wiegand, "Overview of the High Efficiency Video Coding (HEVC) Standard", IEEE Transactions on Circuits and Systems for Video Technology, IEEE TCSVT, Dec 2012.

[6] J.R. Ohm, G.J. Sullivan, H. Schwarz, T.K. Tan, T. Wiegand, "Comparison of the Coding Efficiency of Video Coding Standards Including High Efficiency Video Coding (HEVC)", IEEE TCSVT, Dec 2012.

[7] A. Norkin, G. Bjøntegaard, A. Fuldseth, M. Narroschke, M. Ikeda, K. Andersson, M. Zhou, G.V. Auwera "HEVC Deblocking Filter", IEEE TCSVT, Dec 2012.

[8] C.M. Fu, E. Alshina, A. Alshin, Y.W. Huang, C.Y. Chen, C.Y. Tsai, C.W. Hsu, S.M. Lei, J.H. Park, W.J. Han, "Sample Adaptive Offset in the HEVC Standard", IEEE TCSVT, Dec 2012.

[9] HEVC-Reference-Software version 8 (HM-8.0) "https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware /tags/HM-8.0/"

[10] "High Efficiency Video Coding (HEVC) text specification draft 8 - version 8", JCTVC-J1003, Stockholm, September 2012

[11] A. Kotra, M. Narroschke, T. Wedi,"Deblocking boundary strength and filtering process simplifications", Tech. Rep. JCTVC-G638, November 2011

[12] P.Bordes, G.Clare, F.Henry, M. Raulet, J. Viéron,"An overview of the emerging HEVC standard", Proceedings of ISIVC, July 2012.

[13] F. Henry, G. Clare, et al others "On WPP support in the Main profile", Tech. Rep. JCTVC-J0575, Stockholm, September 2012

[14] G. Clare, F. Henry, "An HEVC transcoder converting non-parallel bitstreams to/from WPP", Tech. Rep. JCTVC-J0032, 9th JCTVC meeting, Geneva, May 2012.

[15] F. Bossen, "Common Test Conditions and Software Reference Configurations", Tech. Rep. JCTVC-J1100, July. 2012.

[16] Scalable_sequences"ftp://ftp.tnt.uni-hannover.de/scalable/ sequences_under_consideration"

[17] ITU-T Recommendation H.264 and ISO/IEC 14496-10 (MPEG-4) AVC,Advanced Video Coding for Generic Audiovisual Services, 2005.

[18] Anand Meher Kotra, Mickaël Raulet, Olivier Deforges, "Efficient Parallelization Of Different HEVC Decoding Stages", Proceedings of IEEE Data Compression Conference (DCC 2013), Snowbird, UT, USA, March 20-22 2013.