Accepted in the 2017 IEEE International Conference on Multimedia and Expo (ICME 2017), 2017
Hong Kong, China, 10-14 July 2017 https://doi.org/10.1109/ICME.2017.8019449

MASSIVELY PARALLEL RATE-CONSTRAINED MOTION ESTIMATION USING MULTIPLE TEMPORAL PREDICTORS IN HEVC

Esmaeil Hojati, Jean-François Franche, Stéphane Coulombe, Carlos Vázquez,
École de technologie supérieure, Montréal, Canada

esmaeil.hojati-najafabadi.1@ens.etsmtl.ca, jean-françois.franche.1@etsmtl.ca,
stephane.coulombe@etsmtl.ca, carlos.vazquez@etsmtl.ca

ABSTRACT

Rate-constrained motion estimation (RCME) is considered to be the most time-consuming process of H.265/HEVC encoding. Massively parallel architectures, such as graphics processing units (GPUs), used in combination with a multi-core central processing unit (CPU), provide a promising computing platform to achieve fast encoding. However, the inherent dependencies in the process for deriving motion vector predictors (MVPs) prevent the parallelization of prediction units (PUs) processing. In this paper, we present a framework for performing a two-stage parallel RCME, in which the RCME of all the PUs of a frame can be calculated in parallel. A novel method is introduced to overcome the dependencies inherent to the derivation of MVPs. Multiple temporal predictors (MTPs) within the two-stage parallel RCME framework provide fine-grained parallelism encoding without significant BD-Rate penalty, compared to serial encoding. Experimental results show that our proposed approach achieves a BD-Rate improvement of over 1% as compared to state-of-the-art parallel methods providing similar time reductions.

***Index Terms*—** HEVC, GPU, rate-constrained motion estimation, massively parallel architecture, motion vector predictor

1. INTRODUCTION

The latest hybrid video compression standard, H.265/HEVC, was developed by the Joint Collaborative Team on Video Coding (JCT-VC) established by ISO/IEC MPEG and ITU-T VCEG [1]. Although HEVC doubles the compression ratio of H.264/AVC at the same video quality, its computational complexity is considerably higher [2]. Most of its coding complexity is due to rate-constrained motion estimation (RCME) [3][4]. During the last few years, highly parallel processing devices, such as graphics processing units (GPUs) or many-core central processing units (CPUs), have been developed and utilized to accelerate such complex tasks. The number of cores has also increased significantly. For instance, an NVIDIA K40 GPU has 2880 cores. However, to exploit the high processing capacity of such hardware, an algorithm must exhibit a high degree of parallelization. This raises the question whether HEVC encoding can be scaled to such a large number of cores.
High-level parallelization tools in HEVC, such as wavefront parallel processing (WPP) and tiles, allow several Coding Tree Units (CTUs) to be coded in parallel. For example, the maximum number of concurrent processes is equal to the number of CTU rows when WPP is used to encode one frame. This number increases significantly when a variant of WPP, called overlapped wavefront (OWF), is used to encode several frames simultaneously, as proposed by C. Chi et al. [5]. At the cost of a lower coding efficiency, the degree of parallelism can be increased by using tiles in addition to WPP/OWF [6]. Hence, the parallel encoding of CTUs is usually sufficient to maintain a multi-core CPU fully occupied most of the time, especially for high resolutions. However, this cannot provide enough parallelization for a many-core CPU or a heterogeneous architecture consisting of CPU and GPU.

In order to increase the degree of parallelization, many methods process RCME in parallel on several prediction units (PUs). The main challenge with these methods is determining the best motion vector for a PU without knowing its MVPs. Most of them estimate these MVPs by using MVs from already encoded CTUs. Yu et al. [7] and Yan et al. [8] proposed methods estimating the MVPs from neighboring CTUs using spatial information. These methods permit the parallel processing of all the prediction units (PUs) within a CTU. However, to provide parallel RCME for all the CTUs in a frame, the spatial MVP dependency must be removed completely. Subsequently, Chen et al. proposed an algorithm to perform parallel motion estimation (ME) on heterogeneous architectures for a whole frame [9]. This algorithm calculates the motion vectors of entire frame blocks in parallel. However, the MVP is ignored, resulting in poor rate-distortion (RD) performance. An improvement on this idea uses a temporally predicted motion vector. Shahid et al. proposed a method that uses an already available motion vector from a previous frame [10]. Gao et al. proposed a similar method using the collocated motion vectors of the previous frame and a method derived from it that extrapolates the motion vectors into the encoding frame [11]. Although these methods achieve fine-grained parallelism suitable for a GPU, the prediction of MVPs can induce extra overhead for the CPU, without significantly improving the RD performance.

In another type of parallel RCME implementation, Radicke et al. [12] and Jiang et al. [13] used the GPU as pre-processor by calculating the sum of absolute differences (SADs) for the whole search region, and then transferred the results back to the CPU. Compared to [9], their method achieves better RD performance because it preserves MVP dependencies. However, due to the high bandwidth usage when transferring an excessive amount of data, the time reduction is smaller than with other methods.

Considering the above-mentioned methods, it can be observed that, on the one hand, transferring the distortion values (SADs) for the whole search region back to the CPU requires very high bandwidth, leading to a reduced speedup. On the other hand, methods that attempt to predict the MVPs sacrifice RD performance because the MVPs are unknown, and mispredicting them can significantly reduce the RD performance.

In this paper, we propose a novel RCME method using multiple predictors. The method, targeted at GPU/CPU heterogeneous architectures, performs RCME in two stages, and uses multiple temporal motion vector predictors. It provides a high degree of parallelization, which is well-suited for massively parallel architectures, while the RD performance is significantly improved, as compared to previous state-of-the-art methods, and with similar time reduction. Moreover, our approach can be combined with high-level parallel tools such as WPP, tiles and slices to reach a higher degree of parallelization and speed.

The paper is organized as follows. Section 2 presents the RCME process in HEVC and its dependencies. The parallel encoding framework for the GPU/CPU heterogeneous architecture under consideration is presented in section 3. Section 4 provides the experimental results. Finally, Section 5 concludes this paper.

2. RATE-CONSTRAINED MOTION ESTIMATION IN HEVC

HEVC uses a quadtree structure called CTU to partition each frame. This structure consists of blocks and units having a maximum size of 64×64 pixels. A block includes a rectangular area of picture samples with related syntax information. A CTU can be recursively divided into units called Coding Units (CUs). The information associated with the prediction process of a CU is stored in the PUs. Depending on encoding configuration and encoder decision, each PU mode is selected from a set of modes. Partitioning and selection of best modes are done using the rate distortion optimization (RDO) process.

RCME is a process which consists in estimating the best temporal prediction parameters based jointly on the rate and distortion for each PU. The SAD is used as a distortion measure (D) for integer precision motion vectors, while the sum of absolute transformed differences (SATD) is used for fractional motion vectors. Moreover, the rate cost is a function of the motion vector difference with the MVP. The prediction parameters that result in the minimum cost are obtained as follows:

$$\mathrm{P_{ME}} = (mv^{*}, mvp^{*}) = \underbrace{\arg\min}_{\substack{\forall mv \in MV_{search},\\ mvp \in \{mvp_A, mvp_B\}}} \{\mathrm{D}(mv) + \lambda \cdot \mathrm{R}(mvp - mv)\} \tag{1}$$

where the two derived motion vector predictor candidates are denoted by mvp_A and mvp_B. These predictors are selected from neighboring PUs using the MVP derivation process determined by the HEVC standard. The constant λ is a Lagrange multiplier. MV_{search} is the search region composed of the set of integer motion vector coordinates over which the cost minimization process is performed. For the full-search algorithm, MV_{search} covers a square area determined by a search range (SR) variable as:

$$MV_{search} = \{(x, y)\},\ \ |x| \le SR\ , |y| \le SR \tag{2}$$

Because of the interpolation required by fractional pel motion estimation, performing it for the whole search range would impose a huge amount of calculations. Therefore, to overcome this problem, first, RCME is performed for the integer motion vector, and then the fractional motion vector is determined around the best integer motion vector. Consequently, Equation 1 can be calculated by integer motion estimation, followed by fractional motion estimation, using the following equations:

$$\mathrm{P_{IME}} = (imv^{*}, mvp^{*}) = \underbrace{\arg\min}_{\substack{imv \in MV_{search},\\ mvp \in \{mvp_A, mvp_B\}}} \{\mathrm{SAD}(imv) + \lambda \cdot \mathrm{R}(mvp - imv)\} \tag{3}$$

$$\mathrm{P_{FME}} = fmv^{*} = \underbrace{\arg\min}_{\substack{\forall fmv \in \{(imv+x, imv+y)\},\\ x,y \in \left\{0, \pm\frac{1}{4}, \pm\frac{1}{2}, \pm\frac{3}{4}\right\}}} \{\mathrm{SATD}(fmv) + \lambda \cdot \mathrm{R}(mvp^{*} - fmv) \tag{4}$$

where imv^{*} is the optimal integer motion vector, mvp^{*} is the optimal motion vector predictor, and fmv^{*} is the optimal fractional motion vector. Note that in many HEVC implementations, this step is performed by successively considering half-pel, and then quarter-pel precision, and not all fractional positions are tested. From Equation 4, it can be observed that to calculate the RCME of PUs in parallel, the only unknown parameter is mvp because it is derived from neighbors. As a result, the MVP is the main dependency in a framework for a parallel RCME process. In this paper, we assume a video encoder that processes one frame at a time.

However, the proposed method can easily be extended to a case such as OWF, where several frames are processed in parallel by only processing CTUs once their search region refers to an already processed reference region (from a reference frame).

3. PROPOSED MULTI-PREDICTOR RCME METHOD

In this section, we propose an efficient parallel framework for HEVC consisting of a novel multi-predictor RCME (MP-RCME) using multiple temporal predictors (MTPs). The method permits a very high degree of parallelism in GPU/CPU heterogeneous architectures.

3.1. Multi-predictor rate-constrained motion estimation

As can be seen in Section 2, the derivation of the MVP from neighboring PUs prevents a high degree of parallelism. On the other hand, using an improper MVP in the RCME process will produce an incorrect rate cost that will in turn lead to incorrect optimal motion vector (MV) selection. To achieve a high degree of parallelism while preserving a high coding efficiency, we propose a method that evaluates the RCME of Equation 1 on a list of probable MVPs composed of MTPs. Compared to a spatial predictor, the MTPs eliminate dependencies between all the CTUs composing the current frame. Hence, all PUs of this frame can be processed in parallel. Moreover, the RD performance loss is limited by using an appropriate list of MVPs, instead of a single predictor. This proposed method is called multi-predictor RCME (MP-RCME), and is depicted in Figure 1. Using this approach allows us to develop a highly parallel RCME framework. We divide the RDO mode decision procedure into two stages: 1) GPU-RCME, and 2) CPU-RDO. In the first stage, we propose to perform the RCME using multiple predictors (MPs) to remove the dependencies associated with this operation. This permits the parallel RCME processing of all PUs/CUs/CTUs of a frame. The results of this stage will be used in the second stage performed by the CPU. In the second stage, the actual MVP is available, and, using the prior GPU calculated results, the best decision is made.

Furthermore, the frame encoding is executed by two parallel threads on the CPU, one for the RDO stage processing, and the other for offloading the workload to the GPU. Using two separate threads provides asynchronous CPU and GPU execution without stalls. We perform an MP-RCME for all PUs in the GPU, and the CPU performs the RDO process. Furthermore, we use a First In, First Out (FIFO) queue implemented as a circular array of buffers to communicate with the GPU in order to eliminate any possible processing time variations occurring in the GPU.

In the first step of offloading, reference images, original image and MTP list are prepared and transferred to the GPU for RCME. The final corresponding results of each PU are the optimal MV and corresponding distortion (SAD), for each considered MVP in the list. To avoid dependencies,

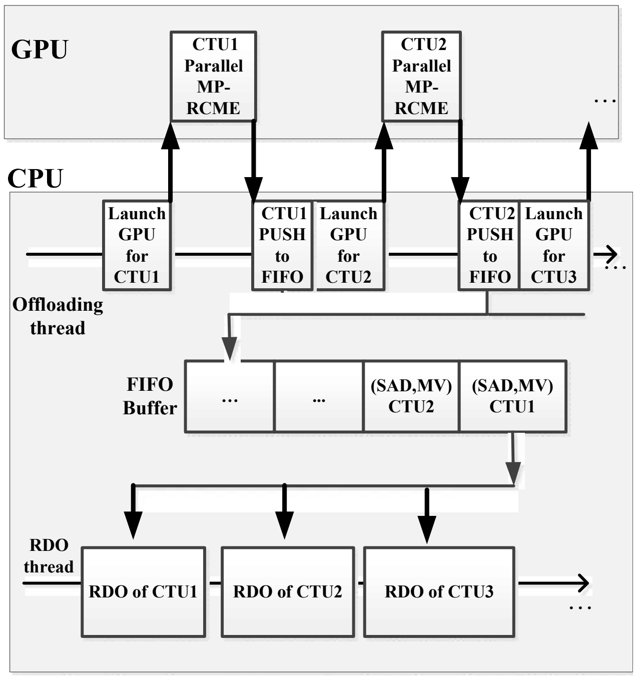

Fig. 1. Multi-predictor RCME architecture

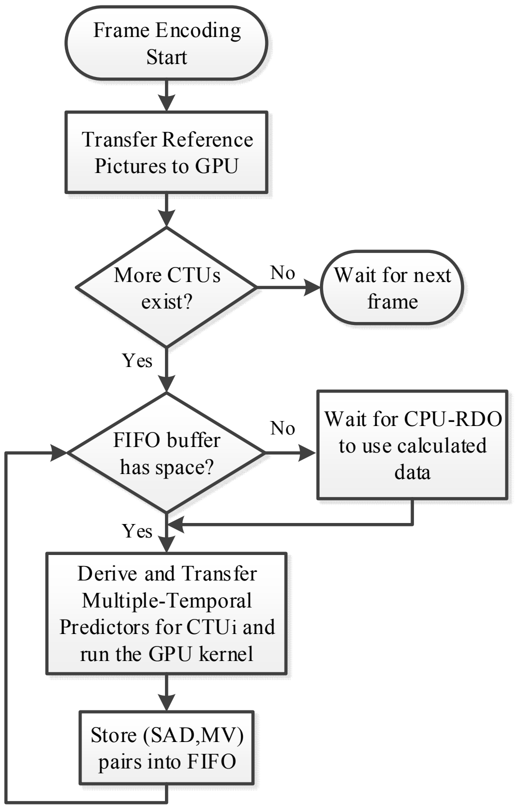
Frame Encoding Start
Transfer Reference Pictures to GPU
More CTUs exist?
No
Wait for next frame
Yes
FIFO buffer has space?
No
Wait for CPU-RDO to use calculated data
Yes
Derive and Transfer Multiple-Temporal Predictors for CTUi and run the GPU kernel
Store (SAD,MV) pairs into FIFO

Fig. 2. Offloading thread flowchart

the MVP list contains MTPs obtained from past frames, as presented in Section 3.2. The execution flow of RCME offloading to GPU is depicted in Figure 2.

Next, in the proposed MP-RCME method, Equation 1 is modified as follows:

$$mv_i = \underset{mv \in MV_{search}}{\arg\min} \; \{\mathrm{D}(mv) + \lambda \cdot \mathrm{R}(mvp_i - mv)\}$$
$$\mathrm{P_{ME}}(mvp_i) = (\mathrm{D}(mv_i), mv_i) \qquad (5)$$

where mvp_i is the i^{th} candidate from the MVP candidate list:

$$mvp_i \in \{mvp_1, \dots, mvp_N\} \qquad (6)$$

where N is the number of probable candidates.

The resulting parameters from Equation 5 are the best rate-constrained motion vector and the corresponding distortion when RCME is performed for mvp_i.

In our proposed framework, we perform integer ME for the candidate list in order to reduce the complexity. The fractional pel refinement will be performed in the CPU, when the actual MVP is available. The best pair in terms of RD is determined in the CPU by the following formula:

$$(mv^*, mvp^*) = \underset{\substack{mv_i,\, with\; i \in 1 \dots N, \\ mvp \in \{mvp_A, mvp_B\}}}{\arg\min} \{\mathrm{D}(mv_i) + \lambda \cdot \mathrm{R}(mvp - mv_i)\} \qquad (7)$$

Equation 7 shows that after the actual MVP is determined, the best-assumed candidate, and consequently the best integer motion vector, are determined with significantly fewer computations. Furthermore, the fractional refinement is performed only for the best integer motion vector, and compared to conventional ME, would not increase the complexity. The RCME process in the CPU is depicted in Algorithm 1.

Moreover, the full-search RCME will be executed efficiently on the GPU because of its simple data structures and equal execution paths. The motion estimation is performed by distortion calculation (SAD) of 4x4 blocks. Similarly to [9], the SAD of bigger blocks is generated by the summation of smaller SAD blocks. For each PU and each motion vector predictor, the best motion vector and SAD will be determined according to Equation 5.

Algorithm 1. PU motion information selection in RDO thread

```
1:  for each PU
2:    BestCost = infinity
3:    for each $mvp = \{mvp_A, mvp_B\}$
4:      for 1 ≤ i ≤ N do
5:        Cost=D[mv[i]]+ λ* $\mathrm{R}(mvp - mv[i])$
6:        if (Cost < BestCost) then
7:          BestCost = Cost
8:          BestIndex = i
9:          BestMvp=mvp
10:       end if
11:     end for
12:   end for
13:   BestIntegerMV=mv[BestIndex]
14:   fmv=FractionalRefinement(BestIntegerMV, BestMvp)
15:   RCME_prediction(PU)= [fmv, BestMvp]
16: end for
```

3.2. Multiple temporal predictors

In HEVC, the MVPs are derived from neighboring and collocated blocks. As a result, using a function of the MVs in the previous frames as the MVP for the current frame can improve the RD performance to some extent [10][11]. However, in some cases, it can reduce quality. For instance, predicting the MVP using the average of MVs can result in a zero MV, while the derivation of the actual MVP using the HEVC standard is more likely to be one of the motion vectors, but not the zero MV. The same analogy is applied in H.264, showing that using an actual motion vector (median motion vector predictor), will result in might be more efficient [14]. Consequently, using the most probable MVs as predicted MVPs will result in less misprediction.

The proposed MP-RCME allows the use of multiple predictors, and we propose using multiple temporal predictors to achieve better RD performance. The MTP list consists of the set of MVs that are the exact MVs of the collocated CTU in the past frame, defined as:

$$mvp_i = mv_{ci}$$
$$MTP = \{mvp_1, \dots, mvp_N\}, N = 16 \qquad (8)$$

where mvp_i is a candidate MVP for the current block, and is equal to the i^{th} motion vector in the collocated CTU in the previous frame.

In HEVC, regardless of the CTU structure, the encoder must maintain a temporal MV field of the frame. To reduce the amount of memory required, MVs are stored in a grid with each cell covering a region of 16x16 pixels [3]. For a CTU of size 64x64, there are 16 temporal MVs, which we use as MTPs. Using the proposed MTP method, there is no overhead for the derivation process since the MVs of the previous frame are already stored. In addition, all of the possible predictors are taken into account, and therefore, any RD performance loss is reduced.

4. PERFORMANCE EVALUATION

To validate the proposed method, we implemented our method in the HEVC Test Model (HM 15.0) [15] by replacing the RCME module with our own implementation, leaving all the rest of the encoder intact. Furthermore, Open Computing Language (OpenCL) [16] was used as the parallel programing framework in order to implement GPU MP-RCME parallelization, and thereby take advantage of its compatibility with different hardware.

In addition, the results were obtained by encoding standard video sequences from the *common HM test conditions* [17]. As well, we used quantization parameters (QPs) of 22, 27, 32, and 37. To measure RD performance, the Bjøntegaard delta rate (BD-Rate) [18] was used. This metric is an extensively used RD performance measure of the encoder with respect to an anchor (the HM), considering both rate and distortion. Positive BD-Rate values imply a decrease in compression performance with respect to the anchor (the HM), and as a result, a decrease in the BD-Rate shows an improvement in the RD performance of the algorithm.

The execution speed was measured using the time reduction (TR) metric. TR represents the average encoding time savings provided by an algorithm as compared to the HM reference encoder, and is calculated as follows:

$$TR = \frac{T_{HM} - T_{Proposed}}{T_{HM}} \times 100 \quad (9)$$

We compared our results with two highly parallel state-of-the-art methods have the same degree of parallelization as our method. The first method was one using a fixed MVP with a value of (0,0) [19][9]. We re-implemented their method to ensure that the differences between the configuration and the hardware would not affect our comparisons.

The second method is involved an MVP derivation by averaging four collocated MVs [20]. However, their method used a maximum CTU size of 32x32. For a fair comparison, we implemented their method, but using a maximum CTU size of 64x64, which resulted in a better RD performance compared to using a 32x32 size. All the methods used the "*Low-delay P*" configuration, with the same encoding parameters. The hardware used consisted of an Intel(R) Xeon(R) CPU E5-2670 running at 2.60GHz, and fitted with an NVIDIA Tesla K20 GPU.

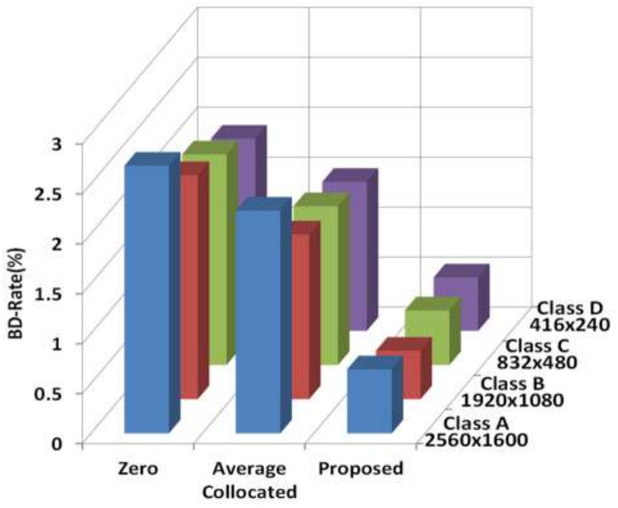

Figure 3: BD-Rate Comparison for different video classes

In Table 1, the performances of these methods are compared. It shows that our algorithm achieves a 1.5% and 1% lower BD-Rate as compared to zero MVP and average collocated methods, respectively. In addition, the time reduction is very similar among all the methods. In Figure 3, we present the results as a function of video resolution. The resolution of the encoded video has no impact on the performance of our method, and for all the video classes, our algorithm provides significant BD-Rate improvements. Compared to [12], which transfers SAD values back to the CPU, our method reduces the BD-Rate by 0.8% and reduces the TR by 20%.

Table 1. Rate distortion and time reduction comparison between proposed method and prior art methods

Video	Zero [19][9]		Average Collocated [20]		Proposed	
	BD-Rate (%)	TR (%)	BD-Rate (%)	TR (%)	BD-Rate (%)	TR (%)
BQSquare (416×240)	1.54	70.3	1.23	71.4	0.298	69.6
BasketballPass (416×240)	1.77	70.7	1.23	72.1	0.362	70.4
BlowingBubbles (416×240)	1.61	73.8	1.35	68.7	0.386	66.5
RaceHorses (416×240)	2.37	70.2	1.81	68.2	0.745	68.6
BQMall (832×480)	1.74	72	1.49	72.1	0.189	70.2
BasketballDrill (832×480)	1.92	72.1	1.53	71.2	0.504	70.4
RaceHorses (832×480)	2.72	71.3	1.99	71.4	1.13	69.7
FourPeople (1280×720)	1.91	72.7	1.41	73.2	0.529	71.3
Johnny (1280×720)	1.59	72.7	1.38	72.2	0.196	71.5
Cactus (1920×1080)	2.36	72.9	1.71	72.8	0.542	70.9
Kimono (1920×1080)	1.99	72.1	1.54	72.1	0.425	70.8
ParkScene (1920×1080)	2.38	73.1	1.7	73	0.489	71
Peopleonstreet (2560×1600)	2.68	73.8	2.23	74.4	0.641	72.4
Average	**2.04**	**72.13**	**1.58**	**71.75**	**0.49**	**70.25**

5. CONCLUSION

In this paper, we presented a multi-predictor RCME framework based on temporal motion vectors. The method provides the high degree of parallelization needed to efficiently exploit massively parallel architectures. It achieves a 70% time reduction with respect to HM, with negligible RD performance reduction. It offers improved RD performance as compared to previous highly parallel methods estimating the motion vector predictor in RCME. Lastly, it can be combined with high-level parallel tools such as WPP and tiles to further increase the speedup.

ACKNOWLEDGEMENT

This work was funded by Vantrix Corporation and by the Natural Sciences and Engineering Research Council of Canada under the Collaborative Research and Development Program (NSERC-CRD 428942-11). Computations were made on the “Guillimin” supercomputer at École de technologie supérieure, which is managed by Calcul Québec and Compute Canada. The operation of this supercomputer is funded by the Canada Foundation for Innovation (CFI), the ministère de l'Économie, de la science et de l'innovation du Québec (MESI) and the Fonds de recherche du Québec-Nature et technologies (FRQ-NT).

6. REFERENCES

[1] B. Bross, W. J. Han, J. R. Ohm, G. J. Sullivan, Y. K. Wang, and T. Wiegand, “High Efficiency Video Coding (HEVC) text specification draft 10,” document JCTVC-L1003, ITU-T/ISO/IEC Joint Collaborative Team on Video Coding (JCT-VC), Jan. 2013.

[2] D. Grois, D. Marpe, A. Mulayoff, B. Itzhaky, and O. Hadar, “Performance comparison of H.265/MPEG-HEVC, VP9, and H.264/MPEG-AVC encoders,” *2013 Picture Coding Symposium (PCS)*. IEEE, pp. 394–397, 2013.

[3] G. J. Sullivan, J. Ohm, W. Han, and T. Wiegand, “Overview of the High Efficiency Video Coding,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1649–1668, 2012.

[4] F. Bossen, B. Bross, S. Karsten, and D. Flynn, “HEVC Complexity and Implementation Analysis,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1685–1696, 2013.

[5] C. C. Chi *et al.*, “Parallel Scalability and Efficiency of HEVC Parallelization Approaches,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1827–1838, 2012.

[6] K. Misra, A. Segall, M. Horowitz, S. Xu, A. Fuldseth, and M. Zhou, “An Overview of Tiles in HEVC,” *IEEE J. Sel. Top. Signal Process.*, vol. 7, no. 6, pp. 969–977, Dec. 2013.

[7] Q. Yu, L. Zhao, and S. Ma, “Parallel AMVP candidate list construction for HEVC,” *Vis. Commun. Image Process.*, 2012.

[8] C. Yan *et al.*, “Efficient Parallel Framework for HEVC Motion Estimation on Many-Core Processors,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 24, no. 12, pp. 2077–2089, 2014.

[9] W. Chen and H. Hang, “H.264/AVC motion estimation implementation on compute unified device architecture (CUDA),” *IEEE Int. Conf. Multimed. Expo*, pp. 697–700, 2008.

[10] M. U. Shahid, A. Ahmed, and E. Magli, “Parallel rate-distortion optimised fast motion estimation algorithm for H.264/AVC using GPU,” *2013 Picture Coding Symposium (PCS)*. IEEE, pp. 221–224, 2013.

[11] Y. Gao and J. Zhou, “Motion vector extrapolation for parallel motion estimation on GPU,” *Multimed. Tools Appl.*, vol. 68, no. 3, pp. 701–715, Mar. 2014.

[12] S. Radicke, J. Hahn, C. Grecos, and Q. Wang, “A highly-parallel approach on motion estimation for high efficiency video coding (HEVC),” *IEEE Int. Conf. Consum. Electron.*, pp. 187–188, Jan. 2014.

[13] X. Jiang, T. Song, T. Shimamoto, and L. Wang, “High efficiency video coding (HEVC) motion estimation parallel algorithms on GPU,” *IEEE Int. Conf. Consum. Electron. - Taiwan*, pp. 115–116, May 2014.

[14] Z. Chen, J. Xu, Y. He, and J. Zheng, “Fast integer-pel and fractional-pel motion estimation for H. 264/AVC,” *J. Vis. Commun. Image Represent.*, vol. 17, no. 2, pp. 264–290, 2006.

[15] “Joint Collaborative Team on Video Coding Reference Software, ver. HM 15.0.” [Online]. Available: http://hevc.hhi.fraunhofer.de/.

[16] “The open standard for parallel programming of heterogeneous systems,” *Khronos Group*. [Online]. Available: https://www.khronos.org/opencl/.

[17] F. Bossen, “JCTVC-L1100: Common HM test conditions and software reference configurations. JCT-VC Document Management System (April 2013).” 2013.

[18] G. Bjøntegaard, “Improvements of the BD-PSNR model,” ITU-T SG16/Q6 Video Coding Experts Group (VCEG), Document VCEG-AI11, Berlin, Germany, Jul. 2008.

[19] S. Momcilovic and L. Sousa, “Development and evaluation of scalable video motion estimators on GPU,” *Signal Processing Systems, 2009. SiPS 2009. IEEE Workshop on*. IEEE, pp. 291–296, 2009.

[20] J. Ma, F. Luo, S. Wang, and S. Ma, “Flexible CTU-level parallel motion estimation by CPU and GPU pipeline for HEVC,” *Visual Communications and Image Processing Conference, 2014 IEEE*. IEEE, pp. 282–285, 2014.