HIGHLY PARALLEL HEVC MOTION ESTIMATION BASED ON MULTIPLE TEMPORAL PREDICTORS AND NESTED DIAMOND SEARCH

Esmaeil Hojati, Jean-François Franche, Stéphane Coulombe, Carlos Vázquez

vantrix
Vantrix Industrial Research Chair in Video Optimization

NSERC CRSNG

compute canada

Calcul Québec

ÉTS
Le génie pour l'industrie
Department of Software and IT Engineering

1. INTRODUCTION

- Fast search motion estimation (ME) algorithms are extremely important to reduce the complexity of high efficiency video coding (HEVC) encoding.
- Massively parallel architectures, such as GPUs, provide a promising computing platform to calculate the best motion vector (MV) of several blocks in parallel in order to achieve fast encoding.

❖ Problem

- In rate-constrained ME (RCME), the best MV depends on motion vector predictors (MVPs):

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

These MVPs are derived from neighboring blocks already processed. A parallel ME algorithm must be designed by taking into account these dependencies.

- Fast search ME algorithms are iterative methods which select different execution paths based on the result of cost evaluations at each iteration.
- However, conditional algorithms are not executed efficiently in GPUs because of their hardware design.
- Thus, a specifically designed algorithm for GPU is required to achieve higher performance.

❖ Prior Art

- In [1], MVP is assumed to be always (0,0).
- In [2], the MVP is estimated by averaging MVs collocated in the reference frame.
- [1] and [2] ➔ reduced the rate-distortion (RD) performance.
- In [2] and [3], ME algorithms for GPU have been targeting exhaustive search algorithms such as full search, but ignored fast search methods.

2. MULTIPLE TEMPORAL PREDICTORS (MTP)

- Perform RCME in two steps:

1- Use a list of MVP candidates, RCME data is calculated in parallel in the GPU:

$$mv_i = \arg\min_{mv \in MV_{search}} \{D(mv) + \lambda \cdot R(mvp_i - mv)\}$$

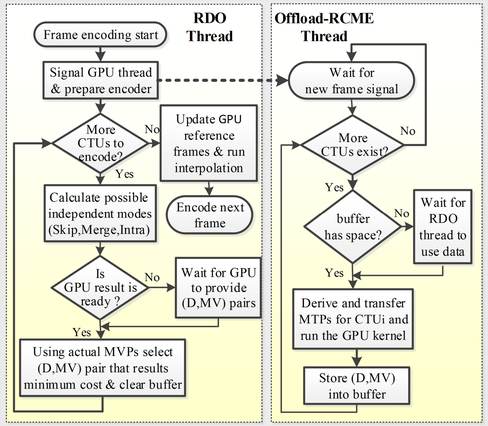
$$P_{ME}(mvp_i) = (D(mv_i), mv_i)$$

$$mvp_i \in \{mvp_1, \dots, mvp_N\}$$

2- Best $\boldsymbol{mv_i}$ and $\boldsymbol{mvp}$ is selected when actual mvp_A and mvp_B are available in RDO thread of CPU:

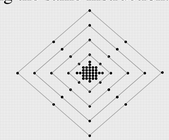
$$P_{ME} = (mv^*, mvp^*) = \arg\min_{\substack{mv_i,\ with\ i \in 1\dots N, \\ mvp \in \{mvp_A, mvp_B\}}} \{D(mv_i) + \lambda \cdot R(mvp - mv_i)\}$$

- Using probable MVP candidate list, dependencies are completely eliminated while RD is only slightly affected.
- All MVs from previously coded frame in co-located CTU are selected as MVP candidates for the current PU.
- For a CTU of size 64x64, there are 16 temporal MVs already stored in the encoder's picture buffer and there is no significant overhead in terms of memory.



3. NESTED DIAMOND SEARCH (NDS)

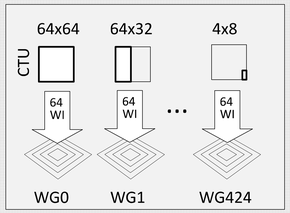
- Smallest execution unit in GPU is a wavefront/warp that contains 64 parallel thread executing the same instruction.
- Designed for GPU architecture, we define a nested fixed diamond pattern consisting of 64 positions.
- RCME is performed in several iteration by finding best MVs of our fixed pattern.


Search pattern

Algorithm 1. Proposed nested diamond search method kernel

```
1:  WG ← get_group_id()          ▶ PU index (idx)
2:  WI ← get_work_id()           ▶ Position idx in search pattern
3:  PUjob ← PUArray[WG]          ▶ PU size and position
4:  SearchPos ← PosArray[WI]     ▶ Search position
5:  BestMVI ← MVPi , iter ← 0
6:  do
7:      Center ← BestMVI
8:      JME[WI] ← SAD(PUjob, Center + SearchPos)
9:      BestMVI ← argmin(JME[0…63])  ▶ After barrier
10:     Iter ← iter + 1
11: while (iter < 4 and abs(Center - BestMVI) ≥ 2)
12: BestMV = fractionalRefinement(BestMVI)
```

- RCME for all possible PUs of a CTU (425 possible PUs) are scheduled at the same time into the execution queue.
- Threads of a warp are being executed efficiently since there is no diverged execution path and all of 64 threads of each PU follow the same execution path.



- NDS for each PU is performed by one workgroup and benefits from memory locality and cache performance.
- Sub-pel interpolation for the whole frame is performed in the GPU.

4. EXPRIMENTAL RESULTS

❖ Methodology

- **Software:** Implementation in the HEVC test model HM15.0.
- **Hardware:** Intel(R) Xeon(R) CPU E5-2670 @ 2.60GHz, equipped with an AMD Radeon R9-270 GPU.
- Encoder is set to "Low-delay P" configuration and quantization parameters (QPs) of 22, 27, 32, and 37.
- Comparison is performed according to:

Name / Param	TZS	Zero-FS [1]	AVG-FS [2]	Zero-NDS	AVG-NDS	MTP-NDS
MVP	Actual MVP derivation	(0,0)	Collocated Average	(0,0)	Collocated Average	Multiple Temporal
RCME method	TZS	FullSearch	FullSearch	**NDS**	**NDS**	**NDS**

❖ Results

Video	BD-Rate (%) compared to HM full search						Time Reduction (%) compared to HM TZS				
	TZS	Zero-FS	AVG-FS	Zero-NDS	AVG-NDS	**MTP-NDS**	Zero-FS	AVG-FS	Zero-NDS	AVG-NDS	**MTP-NDS**
BasketballDrill (832×480)	*1.11*	2.16	1.73	3.14	2.59	**1.60**	42.1	41.7	41.9	41.3	**41.1**
Flowervase (832×480)	*0.31*	2.11	1.52	2.08	1.49	**0.64**	37.9	39.0	37.7	38.6	**38.4**
RaceHorses (832×480)	*1.08*	2.97	2.28	3.80	3.39	**2.56**	38.9	37.8	38.5	37.5	**37.6**
FourPeople (1280×720)	*0.81*	2.15	1.67	3.02	2.39	**1.43**	43.7	43.4	43.2	43.6	**43.1**
Johnny (1280×720)	*0.82*	1.76	1.55	2.64	2.24	**1.57**	44.0	43.7	44.8	44.5	**45.8**
Cactus (1920×1080)	*0.47*	2.63	1.99	2.66	2.01	**1.29**	43.7	42.7	43.5	42.9	**42.9**
Kimono (1920×1080)	*0.59*	2.25	1.72	3.28	2.49	**1.85**	41.3	40.9	41.3	41.4	**41.7**
ParkScene (1920×1080)	*0.48*	2.61	1.93	3.18	2.62	**1.68**	42.6	41.5	42.8	42.1	**42.5**
PeopleOnStreet (2560×1600)	*0.62*	2.98	2.49	3.30	2.62	**1.83**	41.5	42.6	42.3	42.1	**42.3**
Average	***0.70***	**2.40**	**1.88**	**3.01**	**2.43**	**1.61**	**41.74**	**41.48**	**41.78**	**41.56**	**41.71**

5. CONCLUSIONS

- Proposed method achieves **41%** time saving with **0.9%** BD-Rate increase compared to fast search method (TZS).
- GPU load is reduced from 86% for full search to 52% for our proposed NDS method.

[1] 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.
[2] W. Chen and H. Hang, "H.264/AVC motion estimation implementation on compute unified device architecture (CUDA)," IEEE Int. Conf. Multim. Expo, pp. 697–700, 2008.
[3] 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.