GPU-BASED OPTIMIZATION FOR SAMPLE ADAPTIVE OFFSET IN HEVC

Yang Wang[1], *Xun Guo*[2], *Yan Lu*[2], *Xiaopeng Fan*[1], *Debin Zhao*[1]
[1]Harbin Institute of Technology, Harbin, China; [2]Microsoft Research, Beijing, China
[1]{wangyang.cs, fxp, dbzhao}@hit.edu.cn, [2]{Xun.Guo, yanlu}@microsoft.com

ABSTRACT

The latest high efficiency video coding (HEVC) standard achieves about 50% bit-rate reduction at equivalent visual quality compared to H.264/AVC. Sample adaptive offset (SAO) is one of the newly adopted tools right after deblocking filter, which can improve both coding efficiency and visual quality. However, for real-time encoding scenarios, the complexity of SAO is usually too high to be enabled. In this paper, a GPU-based optimization algorithm is proposed to reduce the complexity of SAO. Experiments are conducted based on the state-of-the-art open source HEVC encoder, i.e. X265. Results show that the proposed algorithm can reduce about 70% processing time of SAO on average without sacrifice of coding efficiency.

Index Terms—HEVC, SAO, GPU, X265

1. INTRODUCTION

High efficiency video coding (HEVC) [1], which is developed by the Joint Collaborative Team on Video Coding (JCT-VC), adopts several new coding tools compared to H.264/AVC [2]. Sample adaptive offset (SAO) is an important one which aims at compensating quantization distortion by adding offsets to pre-categorized pixel groups [3]. Since SAO contributes to both coding efficiency and subjective quality, reducing its complexity is necessary for practical application, especially for real-time encoding.

Several algorithms have been proposed for that purpose. J. Joo *et al*. proposed a fast SAO encoding algorithm in [4]. The best SAO edge offset class is decided by exploiting intra prediction mode information instead of rate distortion optimization (RDO). Praveen. G. B *et al*. presented two methods to estimate SAO offset at CTU level in [5]. G. Chen combined edge offset types and decreased search time of band offset from 29 to 4 in [6]. Y. Choi *et al*. reduced complexity in SAO parameter estimation and removed inefficiency caused by its implementation in a pipelined manner in [7]. A parallel computing method was also proposed by E. Ryu *et al*., in which an entire frame is equally divided into sub regions [8].

However, all above methods focus on optimization in HEVC reference software, in which SAO doesn't account for a high percentage on overall complexity. But in real-time aimed encoders, e.g. X265 [9], since acceleration strategies are used in other parts, SAO may account for up to 35% of the whole encoding time, which could be a real bottleneck. Fig. 1 shows the percentage of SAO processing time in X265 for several typical screen sequences. In such case, more efficient optimization algorithms need to be developed.

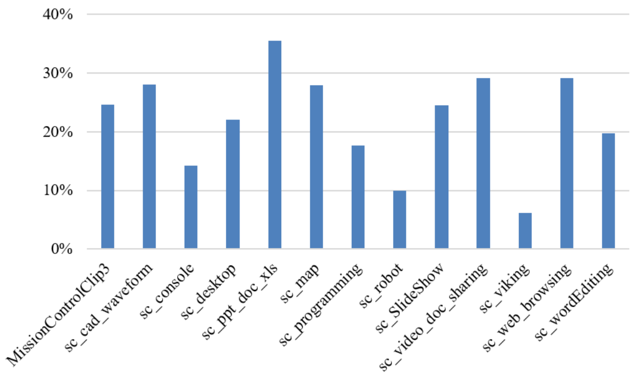


Fig. 1. Percentage of SAO processing time in X265 encoding.

Graphic Processing Unit (GPU) is a good solution for that purpose, because GPU with numerous computing cores and highly parallel structure are available nowadays, so that it has become a powerful assist of CPU to accelerate component technologies in hybrid video coding scheme. Plenty of optimizations based on GPU for motion estimation [10]-[12], motion compensation, and deblocking [13] have been proposed. But there are rarely algorithms that reduce complexity of SAO on GPU. Challenges come from algorithm design and data access overhead. Firstly, pixel-wise constraint exits in type classification and offset computation stage, which will lower the parallelism of GPU. Secondly, data exchange between CPU and GPU is a burden for computation acceleration.

In this paper, we propose a SAO optimization algorithm, which can solve the above problems and accelerate SAO significantly. To achieve this goal, we re-designed the statistical information collection part, which computes offset types and values, to make it well suitable for GPU parallel computing.

The remainder of this paper is organized as follows. Section II gives a brief overview of SAO and GPU. The proposed GPU-based SAO optimization algorithm is described in section III. In section IV, experimental results are shown, followed by the conclusions in section V.

2. BACKGROUND

2.1. SAO in HEVC

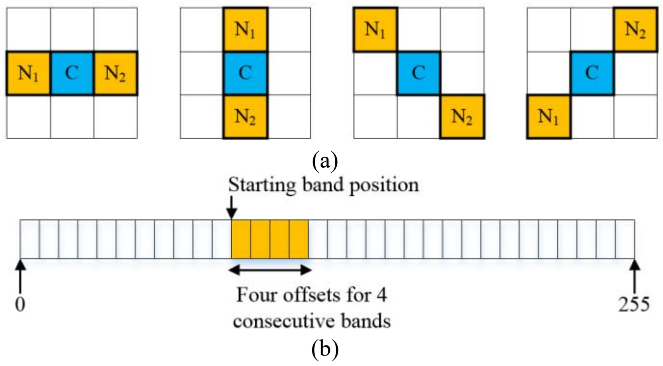


Fig. 2. Edge offset (EO) and band offset (BO) in SAO.

SAO is a post-processing tool in HEVC located right after deblocking filter, which reduces the average distortion and improves visual quality by adding an offset to each pixel conditionally. SAO consists of two kinds of offset: edge offset (EO) and band offset (BO). As shown in Fig. 2 (a), EO is classified into four types: horizontal, vertical, 135 degree diagonal, and 45 degree diagonal. For a certain EO type, the pixel is categorized into five classes by comparing its value with neighboring two pixels: 1 class with zero offset and 4 classes with non-zero offsets. While BO classifies pixels into 32 bands according to bit-depth, as shown in Fig. 2 (b). Once all the pixels are classified, 4 offsets for 4 consecutive bands with minimum RD cost are chosen. After EO and BO are all computed, 4 EO types and 1 BO are sent to RDO process to select the optimal one.

The classification process is very time consuming due to the pixel-wise computation and iterations in finding the best SAO parameters. If it can be performed on GPU in parallel, the speed will be much faster than on CPU. But since there is constraint that different EO types skips different pixel rows and columns, parallelism will be limited.

2.2. Operation on GPU

GPU is a specialized device, which is designed to assist output the graphics to a display. It has tremendous computational capability and high memory bandwidth due to its numerous computing cores and highly parallel structure. Parallel processing on GPU is much more effective than CPU in many cases. To implement SAO algorithm on GPU, we explored two existing tools: Open Computing Language (OpenCL) and CUDA. The former is a framework for writing programs that execute across heterogeneous platforms, while the latter is only applied on graphics cards of NVIDIA. Considering the importance of cross-platforms for different graphics cards, we use OpenCL to implement the proposed SAO optimization algorithm.

3. SAO OPTIMIZATION ALGORITHM ON GPU

As described in Section 2.1, classification for deriving types and offsets is the most time consuming part of SAO. Therefore, in the proposed algorithm, this part is performed on GPU. Specifically, constraint in type classification and offset computation is removed to better support parallel processing on GPU. Meanwhile, the whole CTU row is loaded into GPU memory simultaneously for both original pixels and reconstructed pixels to reduce data access overhead. After all types and offsets are achieved, the RDO with fast distortion estimation is performed on CPU for better logicality judging.

3.1. Parallel processing on GPU

Streaming processor (SP) is the basic processing unit in GPU. Nowadays, GPU in display card usually has many SPs to guarantee good render performance. Since higher parallelism on SP brings better performance, the key point in SAO optimization is to make pixel-wise computation as parallel as possible.

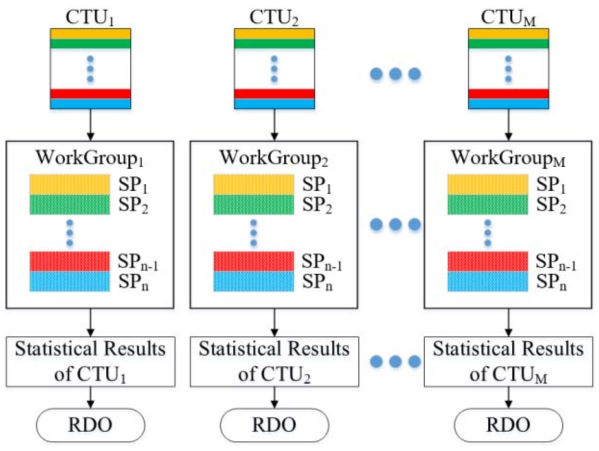

CTU₁
CTU₂
CTU_M
WorkGroup₁
WorkGroup₂
WorkGroup_M
SP₁
SP₂
SPₙ₋₁
SPₙ
Statistical Results of CTU₁
Statistical Results of CTU₂
Statistical Results of CTU_M
RDO

Fig. 3. Parallel processing of SAO on GPU.

The parallel structure of the proposed algorithm is shown in Fig. 3. To fully utilize SPs, each pixel line within a CTU is assigned to a SP for parameter computation. Each WorkGroup contains *n* SPs, where *n* is equal to the CTU height. By this means, the whole CTU can be processed simultaneously by one WorkGroup and share the same local memory. In order to be compatible with wavefront parallel processing (WPP) in HEVC, which aims at supporting parallelism of multiple CPU threads, a row of CTUs will be processed by different WorkGroups at the same time. The advantage of this design is that if WPP is enabled and CPU can afford multiple threads for encoding application besides other possible applications, the encoding can be further accelerated. After all WorkGroups finish, the statistical results of the CTUs including 4 EOs and 1 BO are output to

CPU and determine the optimal one via RDO. To save data exchange overhead, all original pixels and reconstructed pixels of one CTU row are pre-loaded into GPU memory before the computation begins. The total number of SPs used for one CTU row is $N=n*M$, where N denotes the total number of SPs used for one CTU row, and M denotes the number of CTUs in one row, i.e. workgroup number.

As described before, there is constraint on pixels when collecting statistical information of SAO that different SAO types use different number of pixels for type classification and offset computing. Specifically, when deblocking is used, the right 4 columns and the bottom 5 rows of the CTU are excluded in collecting statistical information of luma component. However, when deblocking is not used, the excluded pixels depend on SAO types as shown in TABLE I. please note that the excluded pixels will still be added on corresponding offsets, although they are not used for the offset computation.

TABLE I: Pixels excluded in collecting information for SAO

SAO Type	Right columns	Bottom rows
BO	3	4
EO0(horizontal)	3	5
EO1(vertical)	4	4
EO2(135 degree)	4	5
EO3(45 degree)	4	5

This constraint is a big burden for the parallelism. GPU has to perform logical judgement pixel by pixel to select the correct ones for computation, which also leads to more data access times for one pixel line. This reduces the computation speed significantly. To avoid this, we removed the constraint, which means all the pixels within a CTU are used for information collection for all SAO types no matter deblocking filter is used or not. This change is non-normative that will not change the syntax of SAO. There are two advantages for this change. Firstly, the GPU judgement is removed so that the computation complexity is reduced. Secondly, more pixels are used to calculate the offsets, which increases the accuracy.

One possible mismatch may occur when deblocking filter is enabled and CTU level processing is used. In such case, several pixel lines in the bottom of a CTU will not be deblocked until its lower CTU in next row is in deblocking. This is to avoid adding additional line-buffers in hardware to store the un-deblocked pixels in the upper CTU for the use of the lower CTU. In our algorithm, these un-deblocked pixel lines will be included in the offset computation together with other deblocked pixels. However, no coding loss is observed so that this little mismatch can be ignored.

3.2. Data flow on GPU

Data loading and access is key issue for the proposed algorithm. As described in Section 3.1, before collecting statistical information, original pixels and reconstructed pixels are transferred to GPU. After collecting statistical information, the results, i.e. offset values, are transferred to CPU. Then RDO with fast distortion estimation is performed to determine the best SAO type. Kernel function of GPU consists of three parts: loading data from CPU, collecting statistical information for all the CTUs, and transferring data to CPU. To maintain the data consistency, kernel function must be synchronized after loading data from CPU and after transferring data to CPU. The data flow of kernel function is as follows.

```
Step1. Load data to GPU.
Step2. Sync until all workgroups finish loading data.
Step3. Each SP calculates offsets for one pixel line. All
       SPs in all workgroups are carried out at the same
       time.
Step4. Sync until all workgroups finish calculating.
Step5. Transfer results to CPU.
```

4. EXPERIMENTAL RESULTS

To verify the performance of the proposed algorithm, experiment is conducted based on X265 version 1.4, which is one of the best open source HEVC encoders. X265 has been well optimized on coding structure and encoding modules, e.g. motion estimation, by using fast algorithms and SSE instruction set. The main reason why we use X265 encoder instead of HM encoder is that we aims at achieving real-time HEVC encoder and SAO is a bottleneck for X265. Therefore, optimization of SAO is more important and challenging in X265.

X265 encoder has various pre-defined parameter sets, named presets, that provide different trade-offs between encoding speed and compression efficiency. Encoding time decreases and compression efficiency increases with preset number changes from ultrafast, i.e. preset 0, to placebo, i.e. preset 9. We choose preset 5 in this paper due to its good balance between encoding speed and efficiency.

Video sequences, 10s for each, used by HEVC standardization with both 4:2:0 and 4:4:4 are tested, which include both natural video and screen content video. Four QP values, i.e. 22, 27, 32, 37, and IPPP structure are used. The computer with Intel Xeon E5-1620 3.7GHz quad-core processors with 16GB memory and Microsoft Windows 8.1 Enterprise operating system is used as the test platform. And the GPU used for SAO optimization is AMD R9 290x with 2816 streaming processors. Processing time of SAO is shown in Fig. 4, which is the average of four QPs. The yellow bar denotes processing time of SAO in the proposed algorithm and blue bar denotes that of SAO in X265 with preset 5. According to the results, encoding complexity is reduced significantly. The average processing time of SAO in X265 before and after using the proposed optimization are about 20.08s and 6.22s, respectively, which means the

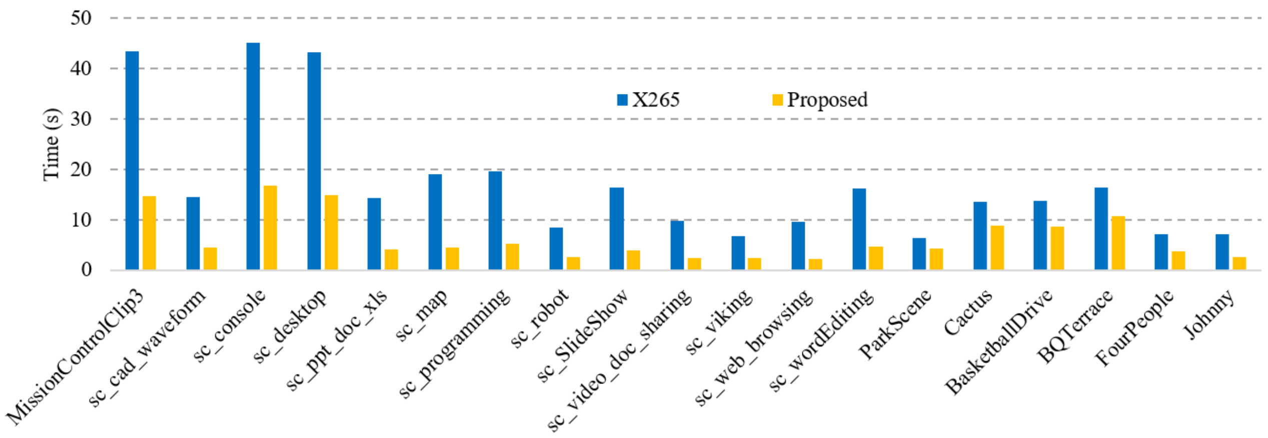


Fig. 4. Processing time of SAO for different sequences.

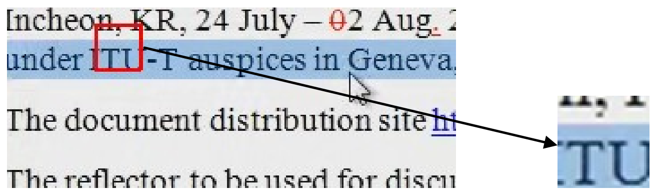
proposed algorithm saves about 70% processing time of SAO. The time saving decreases in natural video, i.e. the right 6 sequences in the figure. That is mainly because the chroma pixels are much less in these sequences.

TABLE II: BD-rate of proposed method compared to X265

Sequence	Y	U	V
MissionControlClip3	-0.4%	-0.17%	-0.3%
sc_cad_waveform	-3.23%	-2.76%	-2.44%
sc_console	-0.59%	-0.09%	-0.17%
sc_desktop	-0.72%	-0.3%	-0.25%
sc_ppt_doc_xls	-1.87%	-0.07%	-0.04%
sc_map	0.04%	0.45%	0.47%
sc_programming	-0.59%	-0.11%	-0.23%
sc_robot	-0.01%	0.18%	-0.15%
sc_SlideShow	0.87%	1.25%	0.77%
sc_video_doc_sharing	-0.51%	0.16%	-0.07%
sc_viking	-0.44%	-0.32%	-0.23%
sc_web_browsing	-0.27%	0.32%	0.31%
sc_wordEditing	0.28%	-0.21%	-0.42%
ParkScene	0.04%	-2.63%	-3.90%
Cactus	-0.21%	-11.59%	-8.65%
BasketballDrive	-0.05%	-0.81%	-1.49%
BQTerrace	-0.08%	-6.96%	-9.55%
FourPeople	-0.03%	-7.35%	-8.25%
Johnny	-0.15%	1.38%	-0.26%
Average	-0.42%	-1.56%	-1.83%

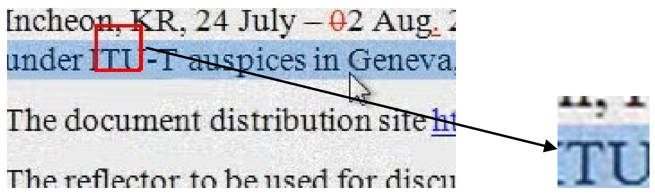
We also compared compression efficiency of the proposed algorithm with that of X265 preset 5. The average bit-rate comparison is shown in TABLE II, in which positive means coding loss and negative means coding gain. From the results, we can see that there is no loss of compression efficiency for the proposed method in almost all sequences for Y component. In most cases, some coding gain can be observed, which is mainly because more pixels are used to collect statistical information in some BO types. On average, the proposed algorithm achieves about 0.42% bit saving.

More screen content videos are used in the experiment than natural video, because we found the benefits of visual quality for screen content video is much larger than natural video. Fig. 5 shows the visual quality of a frame in sc-wordEditing sequence at QP 27. The difference of SAO on and off is very obvious.



Incheon, KR, 24 July – 02 Aug.
under ITU-T auspices in Geneva
The document distribution site ht
The reflector to be used for discu
TU

(a)



Incheon, KR, 24 July – 02 Aug.
under ITU-T auspices in Geneva
The document distribution site ht
The reflector to be used for discu
TU

(b)

Fig. 5. A frame of sc-wordEditing with SAO is on (a) and off (b).

5. CONCLUSIONS

In this paper, we propose a SAO optimization algorithm based on GPU. Specifically, a parallel structure of offset computation is designed for GPU, which overcomes the inherent drawbacks of SAO to parallelism. Experiments are conducted on X265 encoder, which is more challenging for the proposed algorithm. Results show that the proposed algorithm can reduce complexity of SAO significantly, i.e. 70% on average, without loss of compression efficiency.

6. ACKNOWLEDGEMENT

This work was supported in part by the National Science Foundation of China (NSFC) under grants 61472101 and 61390513, the Major State Basic Research Development Program of China (973 Program 2015CB351804), and the National High Technology Research and Development Program of China (863 Program 2015AA015903). This work was done when the first author was an intern at Microsoft Research, Beijing.

7. REFERENCES

[1] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand, “Overview of the high efficiency video coding (HEVC) standard,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, pp. 1648-1667, 2012.
[2] T. Wiegand, G. J. Sullivan, G. Bjntegaard, and A. Luthra, “Overview of the H.264/AVC video coding standard,” *IEEE Trans. Circuits Syst. Video Technol*., vol. 13, pp. 560-576, 2003.
[3] C.-M. Fu, C.-Y. Chen, Y.-W. Huang, and S. Lei, “Sample adaptive offset for HEVC,” in *Proc. IEEE 13th Int. Workshop on Multimedia Signal Processing (MMSP)*, pp. 1-5, 2011.
[4] J. Joo, Y. Choi, and K. Lee, “Fast sample adaptive offset encoding algorithm for HEVC based on intra prediction mode,” in *Proc. IEEE Int. Conf. Consumer Electronics (ICCE)*, pp. 50-53, 2013.
[5] Praveen. G. B, and R. Adireddy, “Analysis and approximation of SAO estimation for CTU-level HEVC encoder,” in *Visual Communications and Image Processing (VCIP)*, pp. 1-5, 2013.
[6] G. Chen, Z. Pei, Z. Liu, and T. Ikenaga, “Low complexity SAO in HEVC base on class combination, pre-decision and merge separation,” in *Digital Signal Processing (DSP)*, pp. 259-262, 2014.
[7] Y. Choi, and J. Joo, “Exploration of practical HEVC/H.265 sample adaptive offset encoding policies,” *Signal Processing Letters (SPL)*, vol. 22, pp. 465-468, 2015.
[8] E.-k. Ryu, J.-h. Nam, S.-o. Lee, H.-h. Jo, and D.-g. Sim, “Sample adaptive offset parallelism in HEVC,” *Multimedia and Ubiquitous Engineering*, pp. 1113-1119, 2013.
[9] http://x265.org/.
[10] X. Wang, L. Song, M. Chen, and J. Yang, “Paralleling variable block size motion estimation of HEVC on multi-core CPU plus GPU platform,” *International Conference on Image Processing (ICIP)*, pp. 1836-1839, 2013.
[11] F. Wang, D. Zhou, and S. Goto, “OpenCL based high-quality HEVC motion estimation on GPU,” *International Conference on Image Processing (ICIP)*, pp. 1263-1267, 2014.
[12] W. Xiao, B. Li, J. Xu, G. Shi, and F. Wu, “HEVC encoding optimization using multi-core CPUs and GPUs,” *IEEE Trans. Circuits Syst. Video Technol*., vol. 25, pp. 1830-1843, 2015.
[13] D. F. de Souza, N. Roma, and L. Sousa, “Cooperative CPU+GPU deblocking filter parallelization for high performance HEVC video codecs,” *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 4993-4997, 2014.