International Journal of Innovative Research and Advanced Studies (IJIRAS)
Volume 3 Issue 4, April 2016

ISSN: 2394-4404

An Efficient Motion Estimation Technique For High Efficiency Video Coding (HEVC/H.265)

Borra Venkata Chandra Swaroop

M.Tech Scholar,
School of Electronics and Communication Engineering,
REVA University, Bengaluru, Karnataka, India

Dr. V. Siva Reddy

Professor,
School of Electronics and Communication Engineering,
REVA University, Bengaluru, Karnataka, India

Raveendra Gudodagi

Assistant Professor,
School of Electronics and Communication Engineering,
REVA University, Bengaluru, Karnataka, India

Dr.Vijay Prakash A M

Professor, Department of Electronics and Communication Engineering, Bangalore Institute of Technology,
Bengaluru, Karnataka, India

Abstract: ***High Efficiency Video Coding (HEVC), also called as H.265 which is the present compression standard, which play a crucial role in many video transmission applications in the coming future. Its best compression performance enables H.265 to be specifically suitable for ultra high definition videos in multimedia environments, however, it comes with the high price and high computational complexity. The H.265 encoding process, involves the Motion Estimation (ME), which is very time consuming process, which makes H.265 unrealistic for real-time applications at this present moment. In this work, an efficient motion estimation algorithm is proposed, to perform bi-predictive ME in a highly efficient manner. The PSNR, Rate-Distortion (RD) and Time comparison performances are evaluated in this work.***

Keywords: High Efficiency Video Coding (HEVC), Bi-Predictive Motion Estimation, Sum of Absolute Differences (SAD).

I. INTRODUCTION

The High Efficiency Video Coding (HEVC/H.265), the newest standard became finalized in 2013, will made to improve the efficiency in transmission and storage of video content for video transmission applications. H.265 provides improved performance gain when compared with its previous standard, the H.264 Advanced Video Coding standard (H.264/AVC), as it reduces the required data volume or bandwidth by up to fifty percent, while keeping the exact statistical visual quality. So, that is why H.265 is well suited for 4k and 8k Ultra High-Definition (UHD) multimedia applications and will most likely be the state of the art video coding standard for future.

Among the advanced designs highlighted by Ohm et al [1], HEVC employs a highly flexible coding tree structure. Pictures are partitioned into Coding Tree Units (CTUs) with a maximum size of 64x64, which are further subdivided into Prediction Units (PU's), Coding Units (CU's), and Transform Units (TU's). In inter-prediction, PU's are broken down into Prediction Blocks (PBs) using eight different modes, four of which are asymmetric. This feature is referred to as Asymmetric Motion Partitioning (AMP) and is one of the primary innovations of HEVC's inter-prediction scheme.

Meanwhile, the complexity of HEVC is high and particularly the encoder's intricacy is the major drawback as analyzed by Bossen *et al* [2]. Recent studies attempted to reduce HEVC's complexity. For example, Correa *et al* [3] proposed an algorithm that dynamically limits the coding

ISSN: 2394-4404

structures' depth to solve a predefined complexity target for battery-powered mobile devices. In very less resource-constrained environments, particular hardware can be utilized more efficiently in addition to reducing the encoder's complexity at an algorithmic level. For instance, Miyazawa *et al* [4] presented custom-build hardware HEVC encoder based on Field Programmable Gate Arrays (FPGAs) towards enabling real-time performance.

However, such customized hardware setups are not widely available. Most end users will instead expect to use HEVC for day to day real time applications such as videoconferencing or live streaming. Therefore, sufficiently fast encoding speed should be achieved with regular retail computers. Optimized algorithms, such as the fast Motion Estimation (ME) scheme by Kibeya et al [5], can give a partial solution. Such approaches, however, are often heavily data-dependent and are inevitably a compromise between speed and compression performance. Therefore, adequately and reliably in order to meet the real-time demands, HEVC encoders need to exploit the inherent parallel processing capabilities of consumer-level hardware.

In addition, the software has been ported to the HEVC test model HM13, and the encoding with 10 bit internal accuracy is now supported, but here the prototype software used is MATLAB with version 14b Finally, a much larger amount of experimental results are presented to comprehensively analyze the new features and the power consumption, of course when compared to HM13 encoder, MATLAB takes more time in executing the time.

The rest of the paper is structured as follows, the existing motion estimation method (uni-prediction), the proposed motion estimation algorithm (bi-prediction), experimental results, and conclusion of the paper.

II. PRESENT UNI-PREDICTIVE SEARCH METHOD

To find the best MVs, a block matching algorithm is explored. The search is a two-step process, where integer MVs are found first, and then they are refined with quarter-pixel accuracy using an 8-tap Interpolation Filter (IF). The cost is calculated by using Sum of Absolute Differences (SAD) for integer Motion Vectors, and the Sum of Absolute Transformed Differences (SATD) for fractional Motion Vectors, respectively. The implementation follows these two steps and employs the same cost functions.

INTEGER MOTION VECTOR SEARCH

The presented algorithm moves entire CTU's around in the search area instead of searching the MV's for every single PB individually. At every search position, the 64x64 pixel CTU is subdivided into 256 sub-blocks with a size of 4x4, for which the respective SADs are calculated. At this point, sub-sampling is used to speed up the process. Afterwards, the SADs for every single PB inside the CTU are derived in a recursive manner based on the SADs of their individual sub-blocks. The AMPs are calculated using the same principle. Their computation is just slightly more complex since sub-blocks from two different depth layers of the CTU must be accessed. This search method leads to exactly the same results as if the search was performed using individual PBs, although it is much more efficient on a GPU. The described algorithm is referred to as Recursive Sum of Absolute Differences (RSAD) and was inspired by the structured motion tree approach [6]. The principle of Recursive Sum of Absolute Differences is illustrated in Figure.1. It can be seen how the SADs of all PBs are calculated in a recursive manner. For example, the SAD of a 16x8 PB is calculated by adding up the cost of its two 8x8 sub-blocks. Asymmetric PU partitions are formed by adding either two or three SADs. A 24x32 PB, e.g., is formed out of three sub-blocks with sizes of 8x16, 8x16 and 16x32, respectively.

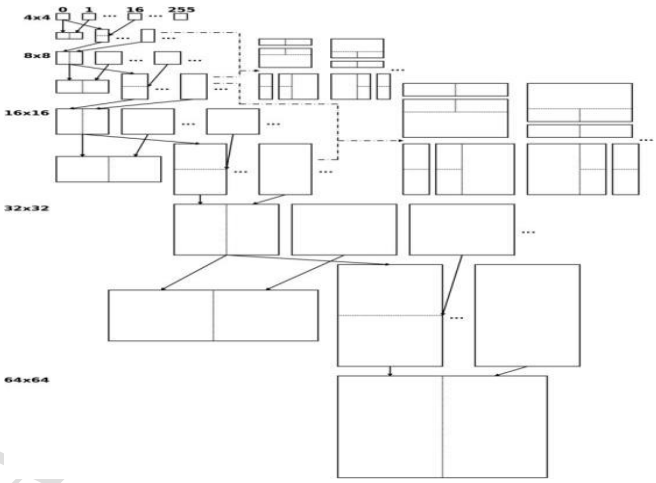

Figure 1: Schematic display of the proposed RSAD algorithm

The RSAD is configured to use a thread block size of 16x16 with one block per CTU. Consequently, every 4x4 SAD is processed by one thread. Parallel reduction is then used to recursively calculate the SADs of all PBs. Hereby, the number of threads is halved after every step until eventually the last thread computes the topmost 64x64 PB's cost. All the data buffers needed for this process is kept in the shared memory in order to minimize access latency. The shared memory is essentially a very fast on-chip cache which can be accessed cooperatively by all threads of the same block. The global memory only needs to be read once at every search position when the 4x4 sub-blocks are calculated. Nonetheless, the amount of shared memory needed per thread block is only 9.43kB. This is important, as it indicates how many blocks can occupy the GPU concurrently. The used graphics adapter consists of 14 so called multiprocessors, each with 192 cores, 48kB shared memory, and 65536 registers. Theoretically, five RSAD thread blocks can reside on each multiprocessor, which means that in total 70 CTUs can be processed simultaneously. This results in a reasonable occupancy and allows the GPU-hardware to effectively hide memory latency. For further optimization, a lookup table is employed which contains pre-calculated costs for all MVs within a {-64..+64} range.

The proposed method is by nature a full-search algorithm where the complexity is independent of the processed data. A search pattern, which can be best described as a Frayed Diamond (FD) as shown in Fig.2, is used to reduce the number of tested positions. The FD pattern features a high search density in the middle which gets progressively lower towards

ISSN: 2394-4404

the edges. With a regular full-search method 16641 positions would be tested within a range of {-64….+64}. The FD approach reduces this number to 3521 and is thus much faster. Meanwhile, the impact on the coding efficiency is negligible.

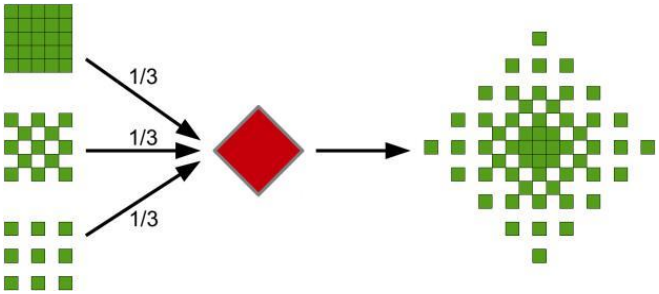

Figure 2: Frayed Diamond search pattern

MOTION VECTOR REFINEMENT

The integer MVs are subsequently refined with first half- and then quarter-pixel accuracy. An eight-tap IF is used in order to generate the needed sub-pixels. One kernel is launched for every PB size and every resultant pixel is calculated by one thread. The thread configuration is hence directly dependent on the number of PBs and their dimensions. The used hardware, however, can only handle a maximum amount of 1024 threads per block. For large PB sizes, it is thus necessary to calculate multiple resultant pixels per thread to stay within this limitation. Horizontal and vertical filtering is performed in separate operations within the same kernel. The intermediate pixels are stored in the shared memory and the final sub-pixel blocks in the global memory. The reasons for such arrangements are two-fold. Firstly, a total of 20 buffers are needed to store all possible half- and quarter-pixel blocks, which are too much data for the very limited available shared memory. Secondly and more importantly, the data must be accessible in the kernel to be subsequently launched to calculate the MVs' cost.

As mentioned, the Sum of Absolute differences is used to estimate the cost of fractional MV's. The PB's are subdivided into blocks of 8x8, 4x4, or 2x2 pixels and then a Hadamard transformation is applied to these blocks prior to the cost computation. In the respective kernel, one thread calculates one transformation and afterwards parallel reduction is applied to calculate the total cost of the respective MV. The needed data is once again stored in the shared memory for maximum efficiency. However, this configuration may lead to a thread block size that is smaller than the warp size, which is 32. This is undesirable because warps are the units in which threads are scheduled by the GPU hardware. Therefore, multiple PB's are eventually processed by one thread block. For example, 16 SATDs with the size of 8x8 are used for a 32x32 PB. To match the warp size of 32, two PB's are assigned to every thread block in that case instead of just one. This concludes the uni-prediction.

III. PROPOSED BI-PREDICTIVE SEARCH ALGORITHM

In contrast to the above uni-prediction, bi-prediction selects one picture from each of the two reference lists. Consequently, two MV's per PB are signaled. The algorithm aims to find two predictions that result in the minimum error between an original block (O) and a prediction block (P = P0 + P1), whereby (P0, P1) are defined by two MV's.

MOTION COMPENSATION

One kernel is launched for every individual PB size which is used to build the prediction (P0) using Motion Compensation (MC). However, the first thing the kernel actually needs to do is to decide which uni-predictive MV should be utilized to construct (P0) for every distinct PB. To make this possible, access to all available reference pictures and all MVs with their respective cost is needed. For every PB, the cost of all found MVs is compared and the one with the lowest cost is selected. The decisions are stored as bit-flags in a dedicated buffer. One bit represents the list index and another bit represents the reference picture index. At this point it becomes clear why bi-prediction is so difficult to implement on heterogeneous platforms, because from here on every single PB potentially depends on distinct reference pictures and operates on different search result buffers.

After the decision process, the kernel uses the IF to create the prediction (P0) for every single PB. In this context, the respectively selected MV in combination with its reference picture is used. The resulting prediction signals are stored in a buffer in the global memory. Similarly, it also employs the shared memory to store the intermediate filter results.

MOTION VECTOR SEARCH

As mentioned, an iterative uni-predictive search is utilized and therefore first integer MV's are found which are afterwards refined with quarter-pixel accuracy. For every PB size, one kernel per available reference picture is invoked to find the individual best integer MV. Which list is used corresponds to the decision made in the MC kernel. If the best MV for the respective PB points to list 0, then list 1 is used and vice versa. This means that if the predictor (P0) was produced using a MV of list 0, the corresponding vector for (P1) is searched in list 1, or the other way around. Therefore, the kernel also needs access to all reference pictures and search result buffers, which is thus rather complex. It should be noted that no reference pictures are being ignored at this point, even if they are redundant. Before the search starts, the reference samples are calculated as (2O – P0). The result is stored in the shared memory and the global memory since it needs to be available in the subsequent refinement step as well. The version residing in the shared memory is sub-sampled and the version in the global memory actually contains all pixels. This is necessary because the SATDs are later used for the fractional search. Afterwards, a regular full-search algorithm with a search range of {-9..+9} is applied to find the best MV. The same cost metrics are utilized as for the uni-predictive search. The thread blocks are configured to consist of 128 threads at most. In the x dimension, each thread processes between 1 and 16 pixels, whereas in the y dimension

ISSN: 2394-4404

always height/2 threads are used due to the sub-sampling. These relatively low thread numbers result in significantly better performance here because the reduced occupancy largely facilitates concurrent kernel execution on the device. The motion vector refinement works almost the same way as described in previous section. The only two differences are that the prediction (2O – P0) is used as reference samples and that the reference picture is selected based on the decision made in the MC kernel. Again, this means that the respective refinement kernels also need access to all pictures and result buffers.

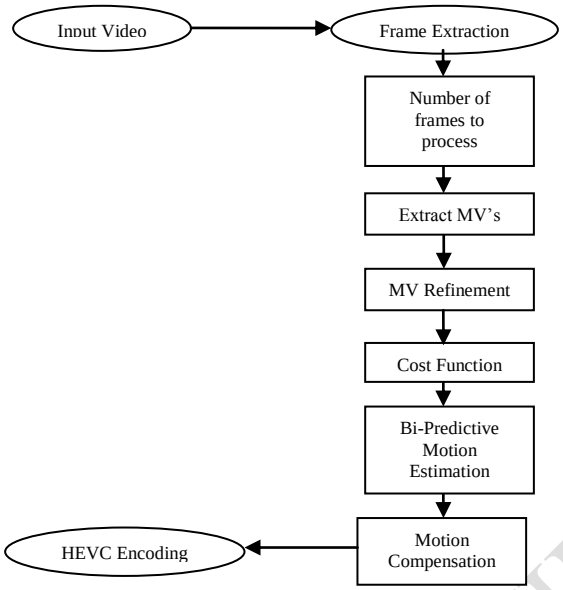

Figure 3: Flow diagram of the proposed bi-predictive search algorithm

EXPERIMENTAL RESULTS

The following development environment was employed to evaluate the performance of the proposed architecture and algorithms: 6-core CPU with 12 logical threads running at 3.20 GHz, 32 GB memory, solid-state drive, programmable GPU with a unified device architecture, and 64-bit Operating System (OS). The machine only consists of regular consumer electronics components and represents a common end-user computer configuration.

The four HEVC test sequences were encoded in Full High-Definition (HD) or beyond: *(1) Kimono 1920x1080* and *(2) Traffic 2560x1600*. The first have 240 frames and run at 24 Frames Per Second (FPS), and the other have 150 frames and 30 FPS. All the sequences have a color sub-sampling of 4:2:0.

The following figures show the original video sequence and Motion estimated video of Kimono *1920x1080* and *Traffic 2560x1600* for the uni-prediction.

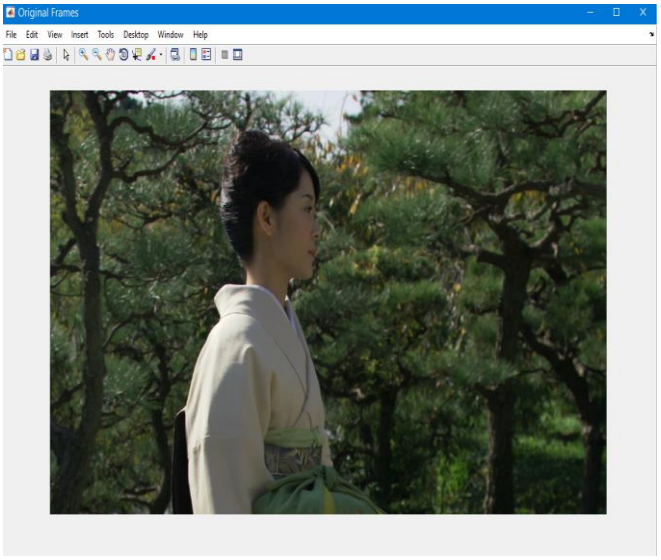

Figure 4: Original Frame from Video Sequence (Kimono)

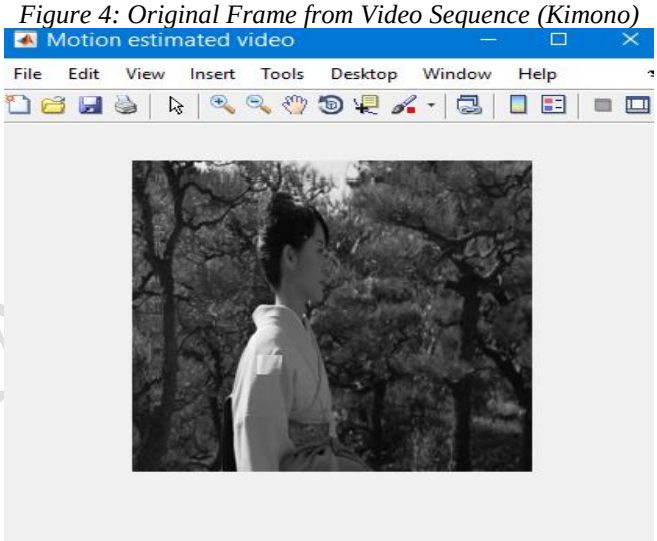

Figure 5: Motion Compensated Video Frame (Kimono)

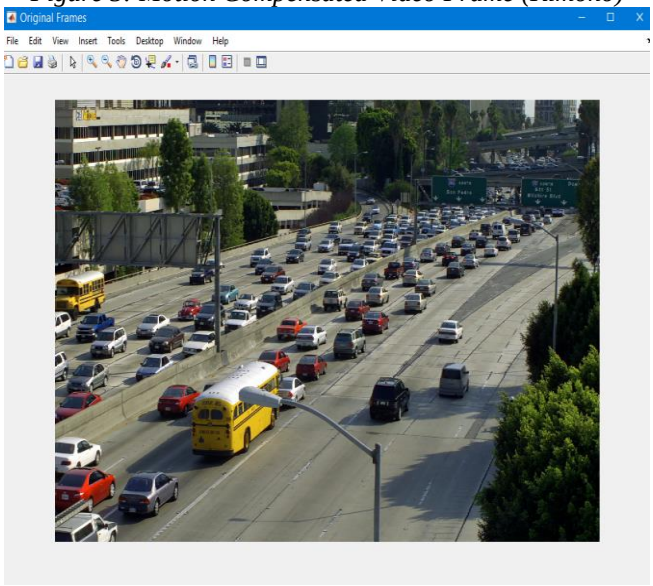

Figure 6: Original Frame from Video Sequence (Traffic)

ISSN: 2394-4404

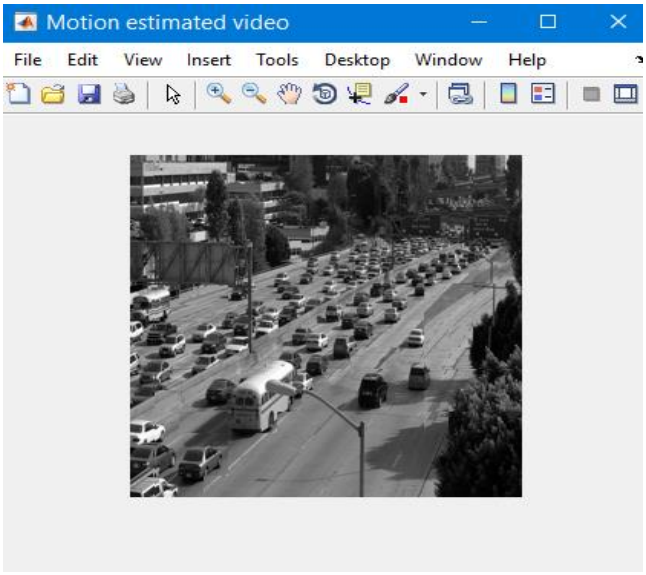

Figure 7: Motion Compensated Video Frame (Traffic)

The following figures show the original video sequence and Motion estimated video of the *Kimono 1920x1080* and *Traffic 2560x1600*for bi-prediction.

Figure 8: Original Frame from Video Sequence (Kimono)

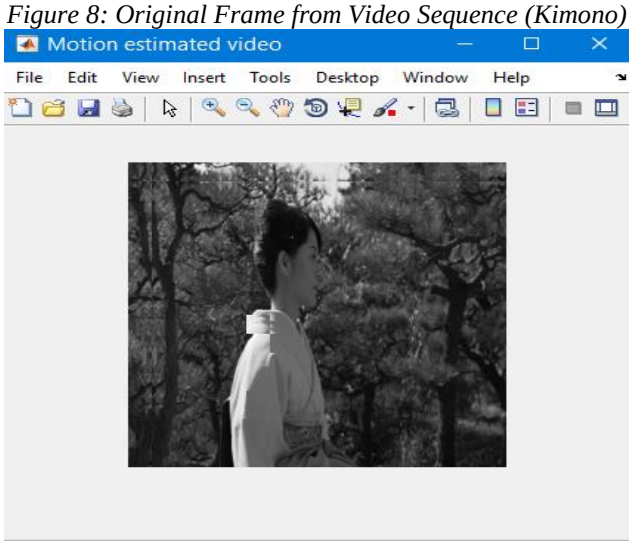

Figure 9: Motion Compensated Video Frame (Kimono)

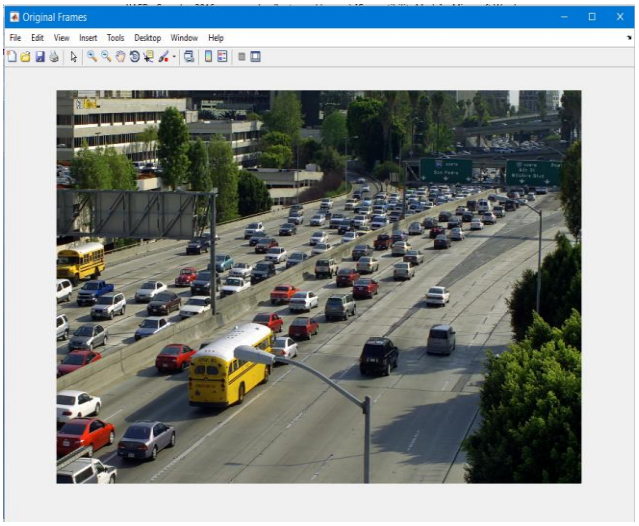

Figure 10: Original Frame from Video Sequence (Traffic)

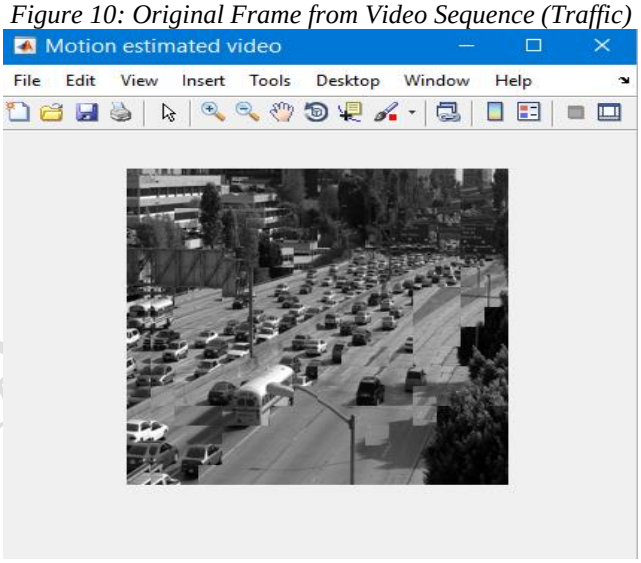

Figure 11: Motion Compensated Video Frame (Traffic)

The following graphs show the comparison of Rate Distortion, PSNR and time taken for Kimono *1920x1080* and *Traffic 2560x1600.*

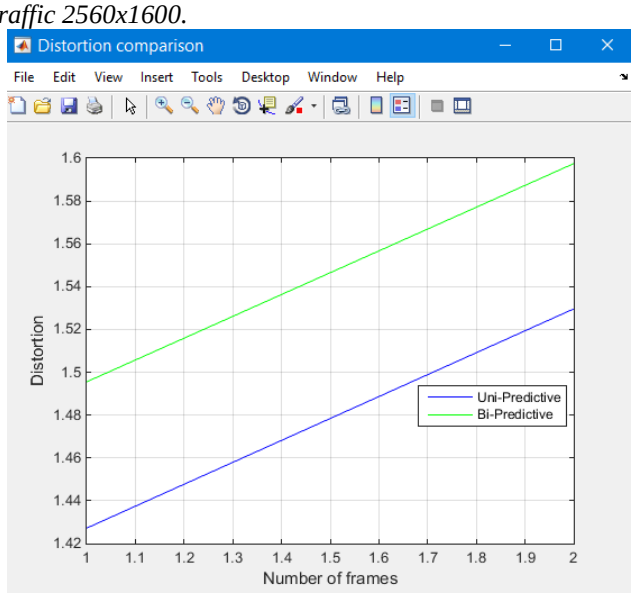

Figure 12: Rate Distortion comparison (Kimono)

ISSN: 2394-4404

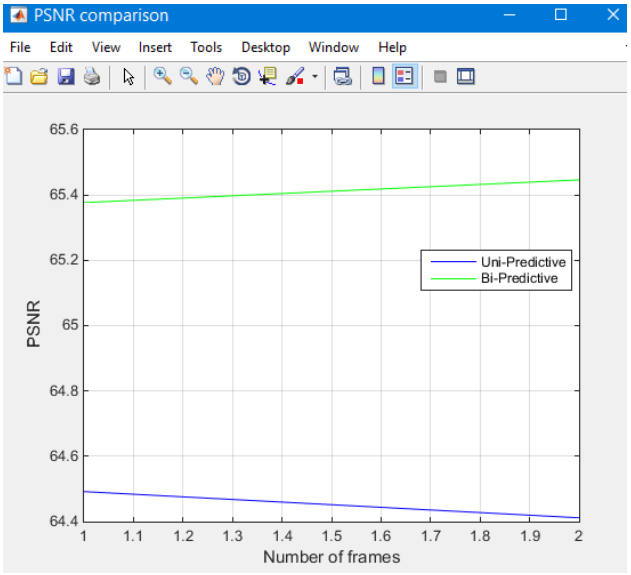

Figure 13: PSNR comparison (Kimono)

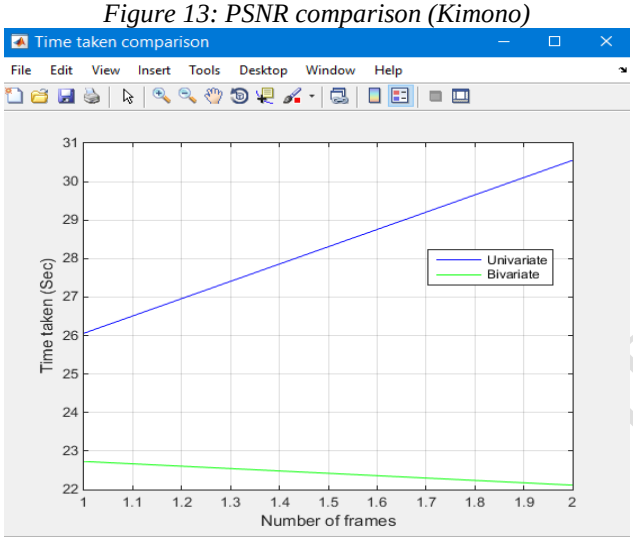

Figure 14: Time comparison (Kimono)

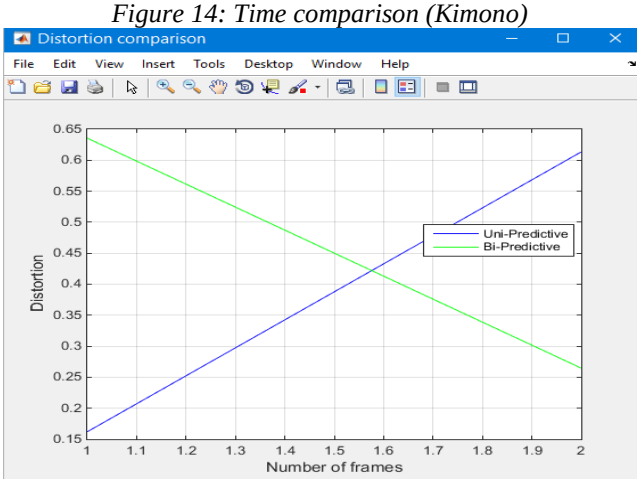

Figure 15: Rate Distortion comparison (Traffic)

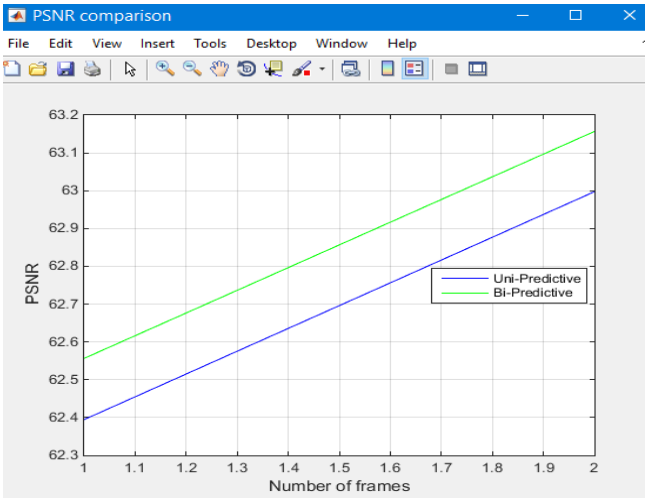

Figure 16: PSNR comparison (Traffic)

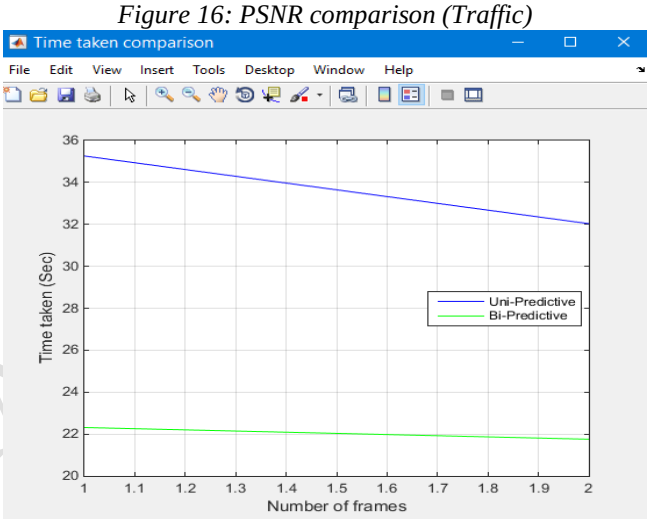

Figure 17: Time comparison (Traffic)

IV. CONCLUSION

A novel architecture with a range of new algorithm has been presented to process bi-predictive Motion Estimation (ME) for High Efficiency Video Coding (HEVC). It can be concluded that bi-predictive Motion Estimation can be computed very efficiently with large speedups and minimal coding losses. From the results we can say that presented prototype implementation has shown that bi-prediction for HEVC can be processed effectively in a high parallel manner when compared to the uni-prediction for the HEVC since, the parameters Time Comparison, Rate Distortion and PSNR will give the better result when compared to the Uni-Predictive search algorithm.

REFERENCES

[1] J.-R. Ohm and G. Sullivan, "High efficiency video coding: the next frontier in video compression," IEEE Signal Process. Mag., pp. 152-158, Jan. 2013.
[2] F. Bossen, B. Bross, K. Sühring, and D. Flynn, "HEVC complexity and implementation analysis," IEEE Trans.

ISSN: 2394-4404

Circuits Syst. Video Technol., vol. 22, no. 12, pp. 1685-1696, Dec. 2012.

[3] G. Correa, P. Assuncao, L. Agostini, and L. da Silva Cruz, "Complexity control of high efficiency video encoders for power-constrained devices," IEEE Trans. Consumer Electron., vol. 57, no. 4, pp. 1866-1874, Nov. 2011.

[4] K. Miyazawa, H. Sakate, S. Sekiguchi, N. Motoyama, Y. Sugito, K. Iguchi, A. Ichigaya, and S. Sakaida, "Real-time hardware implementation of HEVC video encoder for 1080p HD video," in Proc. IEEE International Picture Coding Symposium, San Jose, USA, pp. 225-228, Dec. 2013.

[5] H. Kibeya, F. Belghith, H. Loukil, M. Ben Ayed, and N. Masmoudi, "TZSearch pattern search improvement for HEVC motion estimation modules," in Proc. IEEE International Conference on Advanced Technologies for Signal and Image Processing, Sousse, Tunisia, pp. 95-99, Mar. 2014.

[6] R. Rodríguez-Sánchez, J. Martínez, G. Fernández-Escribano, J. Claver, and J. Sánchez, "Reducing complexity in H.264/AVC motion estimation by using a GPU," in Proc. IEEE International Workshop on Multimedia Signal Processing, Hangzhou, China, pp. 1-6, Oct. 2011.