ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 3, Issue 8, February 2014

Open Multiprocessing Aided Overlapped Motion Compensated Temporal Interpolation

Madiha Sher, Nasru Minallah, Fahad Khan, Muhammad Asif Manzoor, Munaza Sher
Department of Computer Systems Engineering, University of Engineering and Technology, Peshawar, Pakistan
School of Electrical Engineering and Computer Science, National University of Science and Technology, Islamabad, Pakistan
Department of Computer Engineering, Umm Al-Qura University, Makkah, Saudi Arabia

Abstract*— Many today's multimedia applications demand low bit rate transmission of the video sequences due to the limited bandwidth of transmission channels. Video compression is particularly required for these applications for the reception of an acceptable video quality at receiver. An important part of many video compression techniques is motion compensation. Overlapped Motion Compensated Temporal Interpolation (OMCTI) is a block based search approach for the temporal interpolation of skipped frames. It generates interpolated frames with considerably improved video quality at the receiver. Motion compensation is computationally complex and data intensive operation. The multi-core processor has captured major portion of the market due to its enhanced computational capabilities. Increase in single-core microprocessors' performance is limited by semiconductor scaling, associated power and thermal challenges. Currently multi-core CPUs have turned out to be the mechanism for the enhancement of the processor's performance to overcome these limitations. Parallel processing is a very cost-effective solution for the computationally large and data intensive problems. Parallel processing changes the whole way we live. In this work, we speed up the motion compensation by leveraging the multicore processors and an Open MP based multithreaded approach is established to reduce the computational complexity of the OMCTI. The performance of the proposed multi-core processor technique is evaluated with reference to the bench marker as single-core processor in order to analyze the performance tradeoffs. The paper is concluded with a discussion about the generated experimental results. Multi-core processors achieve performance enhancement of 30% - 50% in different scenario while the single-core processors, the bench marker, performance is improved by 5% at the most.

Index Terms—**Block based search, Motion Compensated Temporal Interpolation (MCTI), Multithreading, Open MP, Overlapped Motion Compensated Temporal Interpolation (OMCTI)**

I. INTRODUCTION

The advancement in multimedia application has caused increase in the video display formats and video processing techniques. This increase has resulted in the demand for efficient conversion of one video format to other video formats. Even with the current advancements in the communication infrastructure, bandwidth is still a valuable commodity and many modern multimedia applications still require low bit rates for the video transmission. Video compression is one of the very important processes required for the effective utilization of the channel bandwidth. Every digitized video is comprised of considerable amount of redundant data and compression can be accomplished by exploring these redundancies of videos. These redundancies are generally classified into subjective redundancy and statistical redundancy. The goal of video compression techniques is to reduce the amount of both temporal and spatial redundancies in videos. Motion compensation is very popular technique for video compression. It can take advantage of high temporal correlation between the successive frames. Temporal subsampling is very simple approach for achieving low bit rate requirement. Other compression techniques can be combined with the temporal subsampling to achieve better compression rate. In this approach, the frame rate is reduce at sender side by skipping the frames after a specific interval like transmitting one frame and skipping the next frame and repeats this pattern for the complete video. The dropped frames are required to be reconstructed at the decoder side with the intention of achieving the original frame rate at the receiver end. A simple frame reconstruction method like frame repetition can be used but the output videos are not of good quality and may results in jerkiness. Chi-kong Wong et al in [1] has proposed a technique called Motion Compensated Temporal Interpolation (MCTI) for reconstruction of skipped frames. MCTI is a block based motion compensation algorithm. In MCTI, motion vector for each block is computed by following the blocks between consecutive frames. The displacement between the matched blocks of successive frames is used to place the blocks at appropriate locations in the reconstructed frame and then reconstructed frame is added in between the successive frames to increase the frame rate. MCTI is a block based algorithm so the inserted frames tend to be blocky. Similarly Chi-Kong Wong et al in [2] proposed a technique called Overlapped Motion Compensated Temporal Interpolation (OMCTI) technique to reduce this blocky effect in the reconstructed frame. Main drawback of MCTI and OMCTI algorithms is its massive computational requirements. In order to increase the processors speed, currently the processors manufacturers use to add more processors instead of increasing clock frequency to increase the processor speed. Hence the applications must have to be

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 3, Issue 8, February 2014

modified to exploit multiprocessing efficiently. Parallel programming paradigm can be used to manage this multi-core architecture. These systems can have one or multiple cores. Multi-core architecture can overcome the large computational requirements of motion compensation process. An efficient method for parallel programming is the use of multithreading for the improvement of the computing capability of underlying system. Open Multiprocessing (OpenMP) is a parallel programming API which is used to develop multithreaded applications where the underlying system is shared memory architecture systems. In this article, an OpenMP based multithreaded implementation of Overlapped Motion Compensated Temporal Interpolation is discussed. This implementation is tested on single-core and multi-core processors to determine the impact of multithreading. The remainder of this article is structured as follow: In section-II we provide some discussion about the state-of-the-art literature review. Furthermore, an overview of some motion compensated algorithms and some OpenMP based applications are provided in this section. Moreover MCTI, OMCTI and the parallel implementation is discussed in Section-III. Finally the discussion about our generated experimental results is provided in Section-IV before concluding the article in Section-V.

II. LITERATURE REVIEW

The frame rate up-conversion is performed on video sequence for each pair of temporally adjacent frames. Both forward and backward Motion Estimation (ME) is performed between two frames and exploited the fact that spatially adjacent pixels exhibit high correlation for adaptive refinement. In [3], Motion Compensated Frame Interpolation is performed using an algorithm which is highly efficient and not very complex. Motion vector embedded in bit stream is first examined. Upon blocks whose embedded motion vector is not accurate enough, overlapped block bi-direction ME is performed. Blocking artifact is further reduced by utilizing motion vector post-processing and overlapped blocked motion compensation. Another algorithm, which gives better performance than existing algorithms, is multiple objective frame rate up conversion which produces high quality interpolated frame [4]. This algorithm has two models, the constant velocity model and spatial correlation model which results in much more accurate estimation of motion trajectory. Thus producing high quality interpolated frames. Nowadays there is trend of enhancing performance of processor's performance by using multi-core, but it should be in the form of parallel processing. But the issue is to automatically convert the serial code to parallel by using Open MP or CUDA [5]. The analysis shows that in case of general computers, performance of Intel and PGI is best. While in case of embedded system ROSE, PAR4ALL and Cetus performed best. Zonal algorithm requires less computation and produce efficient result [6]. In this algorithm the predicted frame is constructed using the current frame and previous frame by tracking the blocks of pixels from the previous frame in the current frame. Performance of the algorithm can be improved by using Multihy pothesis Motion Compensated Temporal Interpolation. Zonal algorithm identifies whether the examined frame can be predicted or not; which further improves the performance. Wavelet based compression can be used for video compression. In this approach 2D motion vector is computed per block by splitting frame into blocks. Then regions with the motion are computed for motion estimation. On the basis of these segments, the frames adjacent to central frames are predicted. Hence it doesn't produce blocking artifacts and makes this algorithm better then block transformation coding [7]. To improve the accuracy of motion compensation for video coding background frame can be used as reference frame [8]. Background frames are constructed from the frames that are kept unchanged among certain number of continuous frames. Using both the background frame and previous frame helps in better prediction since the moving objects do not influence the background frames. This algorithm also increases the coding efficiency. In [9], the performance of Gauss-Jordan algorithm is analyzed using different parallel algorithms. Its pipelined implementation of is done using OpenMP programming. In this implementation, if n is the number of cores and p is the number threads; each core is assigned [n/p] contiguous rows of Matrices. N successive steps of the algorithm are executed by each thread. Results show that pipelined implementation is better as compare to other parallel versions of Gauss-Jordon like Rowblock and RowCyclic.

III. METHODOLOGY

A. Motion Compensated Temporal Interpolation

Motion Compensated Temporal Interpolation is used for the reconstruction of those skipped frames at the decoder side which were removed from the video sequence in order to reduce the video size. It is very simple compression technique and doesn't require any sort of complex computation at the encoder side. The reconstructed frames are placed at the appropriate position in order to increase the video frame rate. This increase in frame rate results in the smooth object motion in the video. MCTI algorithm makes some assumptions regarding the motion of the objects. It assumes that the nature of object motion is translational and the motion of objects is relatively small so that the motion is nearly linear with respect to time in the successively received frames. For any frame in the received video sequence, the goal of the temporal interpolation is to reconstruct a frame that can be placed between the current t and previous t-1 received frames. In Fig 1 the object from frame t-1 is moved to a new location in frame t. MCTI calculates the displacement of the object and then place the object at the mid-point between the previous location of the object and new location in previous t-1 and present t frame respectively. MCTI is block based algorithm. It divides all the frames in blocks. The present t frame and the

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 3, Issue 8, February 2014

previous t-1 frame are divided into blocks and the block size is fixed N x N. after this the displacement of each block is calculated. The inserted frame is divided into blocks. Normally the block size is 16 x 16 pixels. Now the displacement of each block from previous frame to present frame is determined and this displacement is known as motion vector. Each block in one frame is search in the other frame to find the displacement of each block. Forward motion vector (from past t-1 frame) and backward motion (from present t frame) are used. The Mean Absolute Difference (MAD) is chosen to calculate the motion vector for each block. The selection of Mean Absolute Difference is triggered by its simple nature. For every block in the one frame, a search area is defined in the other frame of size (2W + 1) x (2W + 1). Theoretically we have to search the complete frame in order to find the block of one frame in the other frame. But we reduce the search area because of the assumption that the object motion is very less in the successive received frames. This reduction in search area reduces the computational requirement of MCTI. The Mean Absolute Difference between the block at position (x,y) of present t frame and the block at position (x+m, y+n) of previous t-1 frame is defined as [1]

$$MAD_{(x,y)}(m,n) = \frac{1}{N^2}\sum_{i=0}^{N-1}\sum_{i=0}^{N-1}|f_t(x+i,y+j) - f_{t-1}(x+m+i,y+n+j)| \quad (1)$$

The block having the minimum of MAD(x,y)(m,n) for all location (m, n) with in the defined (2W + 1) x (2W + 1) search area is consider as best matched block. In this way the motion vector is computed for each block. According to these motion vectors the blocks are placed into the inserted frames.

B. Block Based Search for Motion Vector Estimation

Motion compensation using block based search algorithms are widely used in different video compression techniques. The purpose of block searching is to find the match of a block of present frame in the previous frame. After finding the matched block, the displacement is determined between the matched blocks. Different criterion can be used to find the Similarity between two blocks. Mean Absolute Difference (MAD) is used to calculate the similarity in the case of MCTI and MCTIO.

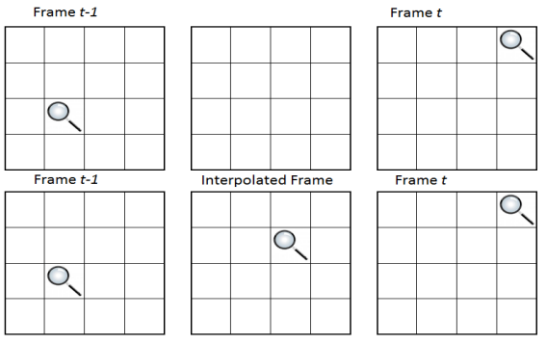

Fig 1: Motion Estimation and Interpolation

Full search is very preliminary block search algorithm for motion compensation. In full search, each block of present t frame is compared with every possible block in the previous t-1 frame with in the fixed size widow and similarity is calculated using MAD. The block having best match is selected as the desired block and its displacement from the original block is considered as the motion vector for this particular block. Similarly this search process is repeated for all the blocks of the frame. Although, the full search algorithm is the simplest algorithm of its kind and it provides with the best Peak-Signal-to-Noise-Ratio among the entire range of block based search algorithms but full search algorithm requires the most computational resources and is poorest algorithm in terms of high computing power requirements. Several other algorithms are available that can replace full search algorithm. These algorithms have reduced computational requirements as compared to full search algorithm at the expense of degraded search results. These algorithms are sub-optimal block based search algorithms. Diamond Search algorithm [10] is one of these sub-optimal search algorithm. The diamond shape is used as central search pattern in this algorithm. The number of steps required to come across the solution by Diamond Search algorithm can be unlimited as opposed to full search algorithm. Diamond search works on two fixed patterns; Large Diamond Search Pattern (LDSP) and Small Diamond Search Pattern (SDSP). Fig. 2 [11] shows both patterns and searching method. The Diamond Search algorithm is composed of the following steps [11]:

Step 1: At first LDSP is centered at the origin of fixed size search window, and nine points of LDSP are tested. If the Minimum Block Distortion MBD evaluated is located at the center location, go to Step 3; otherwise, go to Step 2.

Step 2: The MBD point found in the previous search step is re-positioned as the center point to form a new LDSP and new MBD point is calculated. If the new MBD point obtained is

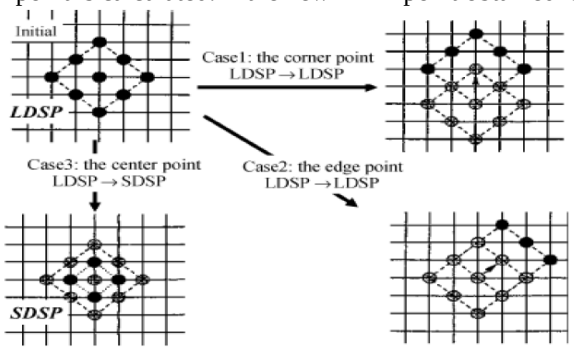

Fig 2: Diamond Search Algorithm

Located at the center position, go to Step 3; otherwise, recursively repeat step 2.

Step 3: Now switch the search pattern from LDSP to SDSP. The MBD point evaluated at this step is the concluding result of the motion vector which refers to the best Matched block.

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 3, Issue 8, February 2014

C. Overlapped Motion Compensated Temporal Interpolation

As discussed in the previous section; MCTI scheme causes an artifact in the temporally interpolated frame. The inserted frames produced by MCTI have blocky effect. There is no smooth transition between different frames. MCTI is a block based temporal interpolated scheme which cause this blocky effect. For matching of every block MCTI only depends upon single motion vector related to the currently under consideration block and as mentioned in the previous section this motion vector is calculated by using MAD between the best matched blocks of previously received frame and currently received frame. Hence location of every block in the inserted frame is estimated with single matched frame and without consideration of the neighboring blocks. The blocky effect in the newly constructed frame, which is produce for the frame rate up conversion, is caused by this single motion vector dependent motion compensation. Motion compensated temporal interpolation with overlapping was introduced to overcome this blocky effect. OMCTI depends not only on its own motion vector for the interpolation of frame but it also uses the motion vector of its neighboring blocks. A block A is accompanied by 4 neighboring blocks B, C, D and E as shown in fig. 3. In this overlapped scheme, every block is divided into four sub blocks. The size of one block is 16 x 16; so the size of these sub blocks is 8 x 8. Let assume that the motion vectors of each block (A, B, C, D and E) is determined using MCTI and they are denoted by V_a, V_b, V_c, V_d and V_e respectively. Block A is divided into A_1, A_2, A_3 and A_4. Every sub block will go through the process of interpolation individually instead of as a single block. The motion vectors are calculated on the block wise basis while interpolation is done on sub block wise basis. For the interpolation of a sub block in inserted frame, the motion vector of the under Consideration block and motion vectors of two blocks adjacent to sub blocks are used.

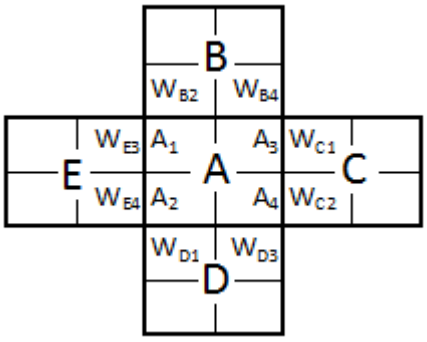

Fig 3: Overlapped MCTI sub blocks

Now the motion vector of under consideration block is more important than the adjacent two blocks in the process of interpolation so the weighted mean is used for this process. Weighted masks are established in a way that the weight of its own motion vector is highest these three blocks and sum of weights of these blocks must be unity. These weights are also defined for every sub block separately. V_A, V_B and V_E motion vectors along with the corresponding weights W_{A1}, W_{B2} and W_{E3} will be used for the prediction of sub block A_1. Similarly V_A, V_D and V_E motion vectors along with their respective weights W_{A2}, W_{D1} and W_{E4} are used for the interpolation of sub block A_2. This overlapping scheme reduces the blocky effect in the resulting video.

D. Parallel Implementation

The increase in the clock frequency of CPU is limited by the laws of physics which forces chip makers to end up with increase in CPU's clock frequency and instead focus on increase in number of cores on a single chip in order to enhance the processor performance. A lot of parallel hardware architectures have created to cope with the limitation imposed by the laws of physics. Some of these diverse parallel hardware architectures are multi-core CPUs, Sun UltraSparc [12], Tillera 64 [13], the Cell Broadband Engine [14], and Graphics Processing Unit (GPU) e.g. NVIDIA [15] and ATI [16]. Currently multi-core CPUs has turned out to be the mechanism for the enhancement of the processor's performance. Parallel processing has increased its usability in every domain of life where the computation requirements are high. Parallel processing changes the whole way we live by providing extra flexibility to implement computationally complex processes. Parallel processing is a very cost-effective solution for the computationally large and data intensive problems. Data intensive applications such as multimedia services, data mining, computational biology, nanotechnology, and information retrieval can be processed by different parallel processing methods (using software or hardware approach) and results in the overall performance improvement of the processing system. Video coding is very important in terms of removing the redundant information from the video in order to make it feasible for transmission and storage. Motion compensation is a core part of video codecs and these motion compensation algorithms utilize different searching algorithm to find the match between the blocks of two consecutive frames received at the destination and construct the intermediate frame in order to increase the frame rate. These searching algorithms used for block matching is the part of motion compensation that requires a lot of computational resources and hence these computationally intense operation make it difficult to implement [1], [17]. In this article, a parallel implementation of search algorithms using the OpenMP is proposed which reduces the computational cost of the overall motion compensation part. OpenMP (OMP) is a parallel programming API which is used to create parallel applications by using the multithreading approach for shared memory architecture systems. An OpenMP application starts with one thread, known as the master thread. With the flow of execution, there can be some parallel regions in which the master thread creates multiple threads; the programmer can set this number of threads. After the execution of parallel region, created threads are stopped and the master thread executes the rest of the serial region. Multithreading is a

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 3, Issue 8, February 2014

method of parallelizing the application where a master thread creates several number of child threads and the task is divided among these child threads. Several threads are created for searching algorithm. Each thread simultaneously perform search and tries to find the best match using the previous frame for a given block in the current frame. Once all the threads have completed their tasks, the results are combined and the final results are generated. As a result the overall computational time of the search algorithm is reduced while all the threads execute simultaneous. This OpenMP based approach is tested on single core as well as multicore processors.

IV. EXPERIMENTAL RESULTS

A. System Platform and Experimental Process

The process is tested on both single-core and multi-core processors for the experimental evaluation. We have used Intel ® Core i3-2310M processor for performance evaluation on multi-core processor. Intel ® Pentium 4 and Intel ® Core ™ 2 Duo (only one core enabled) processors are used for single-core processor evaluation. All programs are coded in C++ using Open MP API. The Intel (R) Core i3-2310M CPU had two processor cores, a 2.10 GHz clock speed and 3 GB of memory. The system ran Windows 7 and Microsoft Visual Studio 2010 Ultimate. The Clock speed of Intel ® Pentium 4 is 3.0 GHz with 2 GB of memory. Windows XP and Microsoft Visual Studio 2008 Professional edition were installed on Pentium 4 system. Intel ® Core ™ 2 Duo had two cores with 2.1 GHz Clock speed and 4 GB of memory. Only one core was enabled at the time of experiments. For single-core results, the parallel algorithm is executed on two different processors; their respective results differ by 0.2% to 0.4% for both processors. Several sets of videos sequences were used to evaluate the performance of the parallel algorithm. The execution time of the sequential algorithm is compared with that of parallel algorithm to find out the performance enhancement. The execution time was measured as an average of 10 runs of the algorithm in each case to reduce the random variation.

B. Evaluation and Analysis of Performance Model

The Open MP based implementation is applied on several test sequences [18] to portray the performance improvement. Both diamond search algorithm and full search algorithm are used for the assessment of the proposed implementation. The experimental results of the suggested implementation are provided in the table 1-4; where each table represents the results for specific type of processer and search algorithm. The rows of each table represent the number of threads and columns represent the different format. Multiple videos of each format have been tested for the validation of the idea. The performance is measured as the comparison of execution time of multithreaded application for specific number of threads with that of a sequential execution of the application for the same test video. Diamond search algorithm is an optimize alternate of full search algorithm at the cost of degraded quality. The number of steps required to reach the output are different for different cases. Description of each table is given below:

Table 1 shows the performance enhancement of parallel algorithm against the sequential algorithm for different numbers of threads and different video formats. Diamond search algorithm is used to find the best match on the multi-core processor. The maximum performance improvement is about 30% in all cases.

Table 2 presents the results for the OMCTI using full search algorithm on multi-core processor. Full search is more computationally expensive process as compared to diamond search; hence processors have to do more computation and the whole process can be easily divided between threads which results in the better performance. These test results shows approximately 50% performance improvement.

Table 3 and Table 4 shows the results for diamond search algorithm and full search algorithm respectively executed on single-core processor. The whole process is again divided into multiple threads but in this case, all the threads are executed on the same processor and threads have to wait for their turn. Due to this reason, performance is not too much increased as compared to multi-core execution. The performance of the OpneMP based application is 5% at the most on single-core processor quite less than that of multi-core processor.

Different video format like QCIF (176 x 144), CIF (352 x 288), 4CIF (704 x 576) and 1080i (1920 x 1080) are used for testing and validation of the idea. The block based search algorithm is implemented as multithreaded parallel task and OpenMP API is used for the multithreading. Numbers of threads are changed while examining the performance in order to analyze the impact of degree of multithreading upon performance. Some computation is also required at the time of thread creation which brings extra overhead in terms of computation. But this overhead is compensated by the performance improvement caused by the parallel execution. All we need is to balance the number of threads to achieve the maximum throughput. According to these results, the maximum performance is achieved with the three or four threads in most of the cases. Multi-core processors have multiple independent processing units; so the threads can be executed on different units independently. This independent execution increases the efficiency of the multithreaded application. In our case, two cores are available for independent processing of the threads. Diamond search algorithm is computationally less expensive algorithm as compare to full search at the cost of degraded quality. The multithreaded implementation has evident with approximately 30% improvement over the sequential version in case of diamond search. The performance is improved by 50% for full search as compared to the sequential version. In case of full search, the performance is more improved as compared to diamond search. The reason for this difference is

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 3, Issue 8, February 2014

more computational resource requirement for the full search. All threads are executed on the same processor while testing the application for the single-core processor. So it does not improve the performance very much. Every thread has to wait for the same processor to get its turn but in case of multi-core, if one processor is not free then the thread can be executed on the other processor. Still approximately 5% performance improvement in time is visible for the single-core processor.

V. CONCLUSION

We attempt to deal with the performance issues of temporal interpolation in this article; temporal interpolation is the foundation of video compression techniques. We have designed a multithreaded architecture of Overlapped Motion Compensated Temporal Interpolation based on OpenMP API. A blocky effect is normally added to the reconstructed video at the receiver side while using Motion Compensated Temporal Interpolation. An overlapped approach is used to overcome this blocky effect. MCTI is computationally expensive process; computation time is reduced by including the multithreading mechanism. The performance improved by the multithreading is quite enough in order to compensate the burden incur by the threads creation process. The number of threads cannot be increased too much otherwise it will negatively affect the performance. Various video sequences of different formats are tested for the proposed scheme. This OpenMP based multithreaded architecture is tested on single-core as well as multi-core architectures. The improvement in each case is presented in the article. It is evident from the results that OpenMP based multithreading exploit the multi-core architecture very well and enhance the overall performance of the process. Comparing multithreaded application with sequential version, performance is improved in every case. Multi-core execution attains more performance boost as compared to single-core implementation.

REFERENCES

[1] Chi-kong Wong, and Oscar C. Au, "Fast Motion Compensated Temporal Interpolation for Video", Proc. SPIE: Visual Communication and Imgae Processing, Vol. 4, pp. 1108-1118, May 95

[2] C. K. Wong, O. C. Au, and C. W. Tang, "Motion Compensated Temporal Interpolation for video", Proc. of IEEE ISCAS, Vol. 2, pp. 608-611, May 96

[3] Tao Chen; "Adaptive Temporal Interpolation using Bidirectional Motion Estimation and Compensation" IEEE ICIP, 2002.

[4] Tak-Song Chong, Oscar C. Au, Wing-San Chau, Tai-Wai Chan; "Multiple Objective Frame Rate Up Conversion", IEEE, 2005.

[5] Chao-Tung Yang, Tzu-Chieh Chang, Hsien-Yi Wang, William C.C. Chu & Chih-Hung Chang; "Performance Comparison with OpenMP Parallelization for Multi-core Systems", Ninth IEEE International Symposium on Parallel and Distributed Processing with Applications, 2011.

[6] Alexis M. Tourapis, Hye-Yeon Cheongf, Ming L. Liouf, Oscar C. Au; "Temporal Interpolation Of Video Sequences Using Zonal Based-Algorithms", Proc. of Conference on Image Processing, pp 895-898, 2001.

[7] Danny Lazar, Amir Averbuch: "Wavelet Video Compression Using Region Based Motion Estimation And Compensation", Proc. of IEEE International conference on Acoustics, Speech and Video Processing, pp 1597-1600, 2001.

[8] Rong Ding, Qionghai Dai, Wenli Xu, Dongdong Zhu and Hao Yin; "Background-frame Based Motion Compensation for Video Compression", Proc. of IEEE International Conference on Multimedia and Expo, pp1487-1490, 2004.

[9] Panagiotis D. Michailidis, Konstantinos G. Margaritis; "Open Multi Processing (OpenMP) of Gauss-Jordan Method for Solving System of Linear Equations", IEEE 11th International Conference on Computer and Information Technology, pp 314-319, 2011.

[10] Shan Zhu and Kai-Kuang Ma, "A New Diamond Search Algorithm for Fast Block-Matching Motion Estimation", IEEE Transactions on Image Processing, Vol. 9, No. 2, pp 287-290, February 2000

[11] Sherief M. Hashimaa, Imbaby I. Mahmoud and Atef A. Elazm, "Hardware Implementation of Diamond Search Algorithm for Motion Estimation and Object Tracking", Proc. of the 7th Conference on Nuclear and Particle Physics, pp 353-365. 11-15 Nov. 2009

[12] UltraSPARC® III Cu Processor Datasheet: http://datasheets.chipdb.org/Sun/UltraSparc-III.pdf

[13] The TILE64™ family of multi-core processors: http://www.tilera.com/products/processors/TILE64

[14] T.Chen, R.Raghavan, J.Dale and E.Iwata. "Cell Broadband Engine Architecture and its first implementation": http://www.ibm.com/developerworks/power/library/pa-cellperf/

[15] What is GPU Computing: http://www.nvidia.com/object/GPU_Computing.html

[16] ATI Stream Technology "GPU and CPU Technology for Accelerated Computing": http://www.amd.com/US/PRODUCTS/TECHNOLOGIES/STREAMTECHNOLOGY/Pages/stream-technology.aspx

[17] C. W. Tang and O. C. Au, " Unidirectional Motion Compensated Temporal Interpolation", IEEE International Symposim on Circuits and Systems, pp 1444-1447 , June 97

[18] Test sequences: http://media.xiph.org/video/derf/

AUTHOR'S PROFILE

Madiha Sher obtained her B.Sc and M.Sc degrees from University of Engineering and Technology, Peshawar, Pakistan in 2010 and 2014 respectively and working as lecturer in same institute. Her research interest includes parallel processing, operating systems and computer vision.

Nasru Minallah is working as assistant professor in UET, Peshawar, Pakistan. He received B.Sc degree from UET, Peshawar, Pakistan 2004 and M.Sc Degree from LUMS in 2006. In 2010 he was awarded Ph.D degree by University of Southampton, UK. His research interest includes low bit-rate video coding, turbo coding and detection, and peer-to-peer video streaming.

Fahad Khan obtained his B.Sc degrees from National University of Science and Technology, Islamabad, Pakistan in 2013. His research interest includes parallel processing, operating systems and programming.

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 3, Issue 8, February 2014

Muhammad Asif Manzoor is a lecturer at the Computer Engineering Department at Umm Al-Qura University, Saudi Arabia. He obtained his B.Sc and M.Sc degrees from the Department of Computer Systems Engineering at the University of Engineering and Technology, Peshawar, Pakistan in 2007 and 2010 respectively. Currently he is working on Visual Sensor Networks, Embedded Systems and Image Processing projects.

Munaza Sher obtained her B.Sc and M.Sc degrees from University of Engineering and Technology, Peshawar, Pakistan in 2009 and 2013 respectively. Her research interest includes Digital Image Processing, Digital Signal Processing and Artificial Intelligence.

TABLE I. DIAMOND SEARCH ON MULTI-CORE PROCESSER

Number of Threads	QCIF			CIF			4CIF			1080i		
	Forman	City	Coast Guard	Foreman	City	Coast Guard	Foreman	City	Coast Guard	Mob Cal 30F	Mob Cal 50F	Park Run 30F
2	27%	27%	26%	26%	30%	25%	28%	28%	28%	21%	21%	20%
3	33%	26%	31%	29%	31%	26%	29%	31%	24%	24%	26%	25%
4	27%	31%	25%	22%	24%	29%	30%	32%	36%	29%	31%	30%
5	21%	28%	22%	23%	16%	17%	23%	26%	28%	23%	24%	24%

TABLE II. Full search on multi-core processor

Number of Threads	QCIF			CIF			4CIF			1080i		
	Forman	City	Coast Guard	Foreman	City	Coast Guard	Foreman	City	Coast Guard	Mob Cal 30F	Mob Cal 50F	Park Run 30F
2	42%	44%	43%	38%	39%	37%	42%	44%	36%	30%	45%	34%
3	49%	51%	49%	51%	52%	46%	50%	46%	49%	41%	51%	45%
4	51%	52%	51%	46%	48%	50%	54%	51%	46%	50%	54%	51%
5	33%	35%	34%	37%	38%	37%	44%	40%	30%	42%	49%	43%

TABLE III. Diamond searchon single-core processor

Number of Threads	QCIF			CIF			4CIF			1080i		
	Forman	City	Coast Guard	Foreman	City	Coast Guard	Foreman	City	Coast Guard	Mob Cal 30F	Mob Cal 50F	Park Run 30F
2	3.9%	4.0 %	3.6 %	4.0%	4.2%	4.3%	3.8%	3.8%	3.9%	3.3%	3.5%	3.8%
3	4.7%	4.8 %	3.9 %	4.2%	5.0%	4.8%	4.1%	4.4%	4.5%	4.2%	4.3%	4.1%
4	4.5%	4.2 %	4.4 %	4.3%	4.7%	4.2%	4.3%	4.2%	4.2%	4.1%	4.0%	4.2%
5	4.0%	4.1 %	4.0 %	4.0%	4.6%	3.9%	4.2%	4.3%	3.9%	4.0%	4.1%	3.9%

TABLE IV. Full search on single-core processor

Number of Threads	QCIF			CIF			4CIF			1080i		
	Forman	City	Coast Guard	Foreman	City	Coast Guard	Foreman	City	Coast Guard	Mob Cal 30F	Mob Cal 50F	Park Run 30F
2	4.2%	4.0%	4.2%	3.8%	3.9%	4.3%	4.1%	3.9%	4.0%	3.7%	4.1%	3.6%
3	4.9%	4.9%	4.9%	5.0%	4.7%	4.6%	4.9%	4.6%	4.8%	4.1%	4.7%	4.4%
4	5.1%	5.0%	5.1%	4.6%	5.1%	4.9%	5.2%	4.9%	4.7%	4.7%	5.4%	4.9%
5	3.8%	3.9%	4.0%	4.1%	4.2%	2.1%	4.3%	4.0%	3.7%	4.2%	5.3%	4.3%