A Real Time Micro-expression Detection System with LBP-TOP on a Many-core Processor

Xin Rong Soh, Vishnu Monn Baskaran, Adamu Muhammad Buhari, Raphael C.-W. Phan
Faculty of Engineering, Multimedia University,
Cyberjaya campus, Selangor, Malaysia.
E-mail: xinrong0815@hotmail.com, vishnu.monn@mmu.edu.my, adam_m.buhari@yahoo.com, raphael@mmu.edu.my

Abstract – **The implementation of a micro-expression detection system introduces challenges to sustain a real time recognition result. In order to surmount these problems, this paper examines the algorithm of a serial Local Binary Pattern from Three Orthogonal Planes (LBP-TOP) in order to identify the performance limitations for real time system. Videos from SMIC and CASMEII were up sampled to higher resolutions (280×340, 560×680 and 1120×1360) to cater the need of real life implementation. Then, a parallel multicore-based LBP-TOP algorithm is studied as a benchmark. Experimental results show that the parallel LBP-TOP algorithm exhibits 7× and 8× speedup against serial LBP-TOP for SMIC and CASMEII database respectively for the highest tested video resolution utilising 24-logical processor multi-core architecture. To further reduce the computational time, this paper also proposes a many-core parallel LBP-TOP algorithm using Compute Unified Device Architecture (CUDA). In addition, a method is designed to calculate the threads and blocks required to launch the kernel when processing videos from different resolutions. The proposed algorithm increases the performance speedup to 117× and 130× against the serial algorithm for the highest tested resolution videos.**

***Keywords* — LBP-TOP, micro-expression detection, parallel computing, CUDA, GPGPU**

I. Introduction

Micro-expressions are brief, involuntary facial expressions which reveal emotions that people trying to hide [1]. These expressions are difficult to detect and recognize as it happens in a very short duration probably 1/25th to 1/3th of a second [2] and the intensities of involved muscle movements are subtle [3]. In contrary, macro-expressions are easier to identify as it happens between two and three seconds and can be found over the entire face region [4]. The ability to recognize micro-expressions is significant for our live as it is beneficial for wide range of applications such as psychological diagnosis and police and military interrogation.

Local Binary Pattern from Three Orthogonal Planes (LBP-TOP) is popularly used for dynamic texture analysis due to its simplicity and illumination invariant. Recently, LBP-TOP is widely used as the baseline for performance comparison in micro-expression detection experiments [4, 5]. Even though LBP-TOP is simple and easy to implement, but a serial LBP-TOP algorithm which runs on a single processors requires long time to process a high resolution video (i.e., 1120×1360).

The rapid development in computing technology has introduced various multi-core processors which can be utilized to speed up the computational time of a program with proper parallel computing skills. As a result, this has encouraged the growth of the many-core processing technology. As such Graphics Processing Unit (GPU) has been widely used in industry as it can solve general purpose computing problems with a large number of processors, hence the term, General purpose computing on GPU (GPGPU). With the introduction of Compute Unified Device Architecture (CUDA), a parallel computing platform from NVIDIA, GPGPU using a many-core architecture has become more popular nowadays. GPUs which are designed as parallel, throughput-oriented computing engines with little overhead between threads are favorable for massive data parallelism in several applications.

Howbeit, a serial LBP-TOP algorithm would not be able to enjoy the potential performance speed up provided by both multi-core and many-core architectures without applying a parallel computing design. Henceforth, this paper investigates the bottleneck of LBP-TOP algorithm in assessing the necessity for parallel architecture to accelerate the micro-expression detection process. Software application threads were used in implementing a parallel LBP-TOP algorithm. The performance speedups against a serial LBP-TOP is first observed as a benchmark. In order to further improve the speedup, the design and implementation of the parallel LBP-TOP algorithm is then extended into a GPU processor using the CUDA parallel computing platform. The proposed CUDA LBP-TOP algorithm divides the video frames into blocks of threads in generating the histogram. Moreover, a method is designed to cope with the calculations of threads and blocks required when processing videos from different resolutions. This method prevents creating redundant threads and blocks, resulting in a longer computational time as the available Streaming Multiprocessors may be used to accommodate these idle threads, thereby maximizing the computational performance.

The paper is organized as follows. Section II analyzes the performance impact of a serial based LBP-TOP algorithm. Then, section III describes and analyzes a parallel LBP-TOP algorithm on a multi-core architecture. Section IV extends the performance of a parallel LBP-TOP algorithm by proposing a design based on using the CUDA parallel computing platform on many-core architecture. Finally, Section V concludes this paper.

II. Performance Impact Of A Serial Lbp-Top Algorithm

A. LBP-TOP Algorithm

Given a 3×3 pixel, the LBP of the center pixel is computed by comparing the neighboring pixel with the value of itself and considering the result as a binary number. The binary number will be converted into decimal number afterwards. This computational is expressed as [5]:

$$LBP_{P,R}(x_c, y_c) = \sum_{p=0}^{P-1} s(i_p - i_c) \times 2^P \qquad (1)$$

where

$$s(x) = \begin{cases} 0 \; if \;\; x < 0 \\ 1 \; if \;\; x \geq 0 \end{cases} \qquad (2)$$

i_c denotes the intensity of the center pixel, P denotes the total number of neighbors points parameterized by the radius of the neighborhood *R*, while i_p denotes the intensity of the neighboring pixels. Then, a histogram that holds the texture features of an image is computed based on the LBP patterns obtained from all pixels in the image. A video is represented as a stack of XY planes in axis T, YT planes in axis X and XY planes in axis Y respectively [7]. Considering micro-expression as an illustration, the XY plane holds the spatial information which includes both identity and appearance of a face whereas the XT and YT planes contain the information about the changes of pixel in vertical and horizontal directions with time which is relevant to the facial movement displacement. Hence, in order to obtain this information, three instances of co-occurrence statistics obtained independently from three orthogonal planes are used.

Since the direction of motion for the face is unknown, the changes of neighboring points around the central pixel with time are considered too. The example images from three planes are shown in Fig. 1. Fig. 1a shows the image in XY, XT and YT planes where XY plane contains the identity and appearance of the mouth while XT and YT plane comprises the information of the mouth's movement in row and column direction in temporal space respectively. Then, a LBP-TOP histogram is formed by concatenating the statistic obtained from 3 planes as shown in Fig. 1b. Generally, the radii in X, Y and T axis and the number of neighbor points on XY, XT and YT planes can be different.

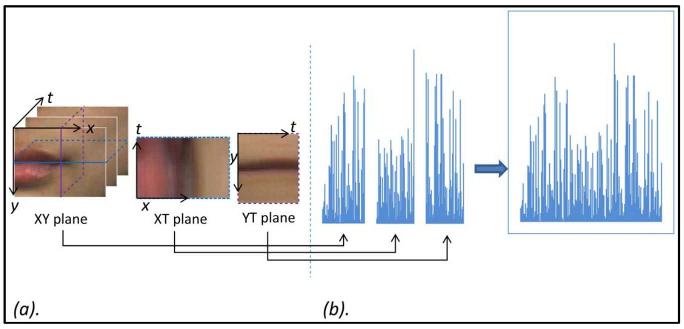

Fig. 1: Concatenation of LBP-TOP histogram from textures of three planes [8]

B. Implementation and Performance Assessment

A serial implementation of the LBP-TOP algorithm is developed using visual C++. This implementation is tested on an Intel Xeon E5-2650 v4 platform with 20 logical processors and 32 GBytes of memory.

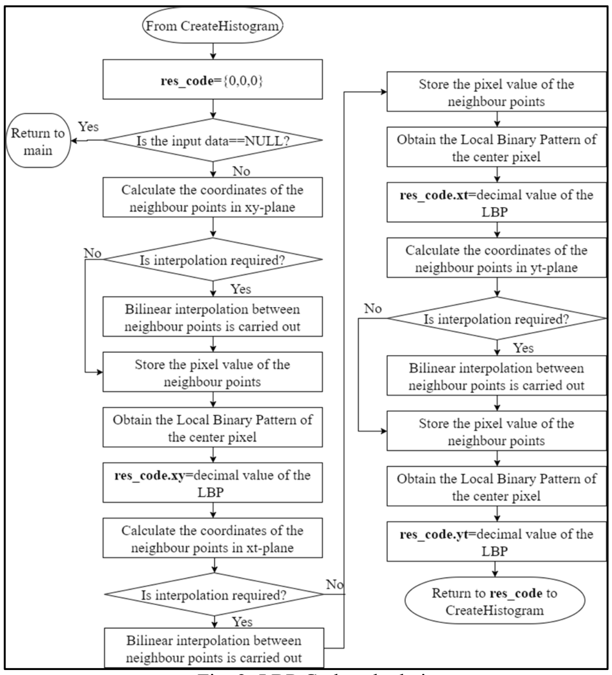
From CreateHistogram
res_code={0,0,0}
Is the input data==NULL?
Yes
Return to main
No
Calculate the coordinates of the neighbour points in xy-plane
Is interpolation required?
No
Yes
Bilinear interpolation between neighbour points is carried out
Store the pixel value of the neighbour points
Obtain the Local Binary Pattern of the center pixel
res_code.xy=decimal value of the LBP
Calculate the coordinates of the neighbour points in xt-plane
Is interpolation required?
No
Yes
Bilinear interpolation between neighbour points is carried out
Store the pixel value of the neighbour points
Obtain the Local Binary Pattern of the center pixel
res_code.xt=decimal value of the LBP
Calculate the coordinates of the neighbour points in yt-plane
Is interpolation required?
No
Yes
Bilinear interpolation between neighbour points is carried out
Store the pixel value of the neighbour points
Obtain the Local Binary Pattern of the center pixel
res_code.yt=decimal value of the LBP
Return to res_code to CreateHistogram

Fig. 2: LBP Code calculation.

Fig. 2 illustrates the implementation of LBP-TOP algorithm. **`Res_code`** is a structure variable that is used to store the LBP Code value of the pixel on three planes. As illustrated in Fig. 2, to compute the LBP Code of a pixel, the coordinate of neighbor points in xy plane are first obtained. If bilinear interpolation is not required, the pixel value of the neighbor points is obtained and then LBP of the center pixel is computed and stored into **`res_code.xy`**. The same algorithm is used for XT and YT planes.

In this project, experiments were conducted using SMIC and CASMEII database. In SMIC, there are 164 videos from 16 subjects and 3 micro-expression classes which are positive, negative and surprise. The average resolution of the video in SMIC is 140×170. In contrary, CASMEII consists of 245 videos from 26 subjects and 5 micro-expression classes which are happiness, disgust, surprise, repression and others. For real life implementation, videos with higher resolution are more preferable. Hence, videos were up sampled to different resolution (280×340, 560×680 and 1120×1360) using OpenCV. For LBP-TOP, videos are divided into 5×5 block size as this block setting is able to generate better recognition accuracy [4]. The histogram of each video is obtained by considering four neighbor points on each plane and setting the radii to $\{R_X, R_Y, R_T\} = \{1, 1, 3\}$ and $\{R_X, R_Y, R_T\} = \{1, 1, 4\}$ for SMIC and CASMEII respectively. Uniform binning and normalization were applied to each histogram before classification. The classification is carried out using SVM with leave one subject out cross validation. SVM is chosen as the available database for micro-expressions is very limited.

Fig. 3 and 4 illustrate the recognition accuracy obtained with different video resolutions from SMIC and CASMEII. The highest accuracy approximately 42% is obtained with the original resolution video from SMIC. In contrast, accuracy of 42.5% is obtained with the highest tested video resolution fr-

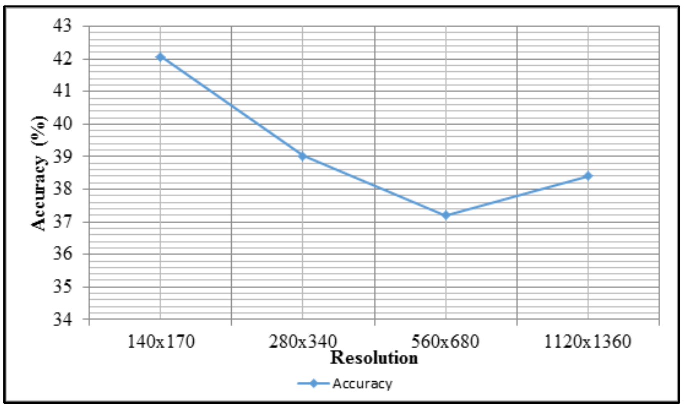

Fig. 3: Accuracy for different video resolution from SMIC.

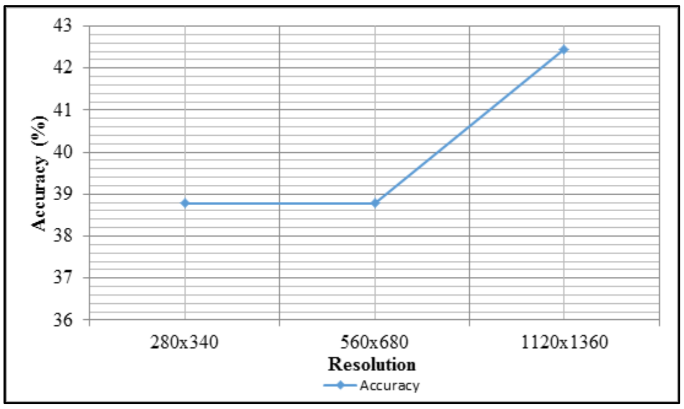

Fig. 4: Accuracy for different video resolution from CASMEII.

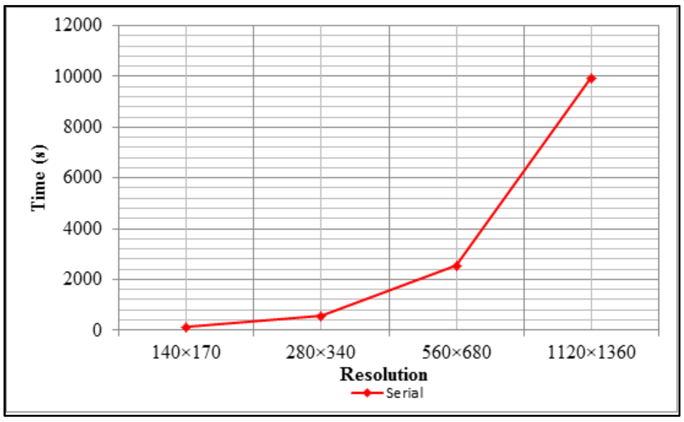

Fig. 5: Serial LBP-TOP algorithm performance on SMIC.

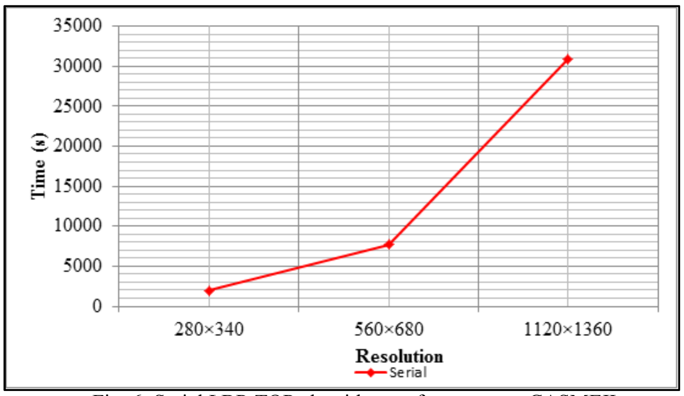

Fig. 6: Serial LBP-TOP algorithm performance on CASMEII.

om CASMEII. The accuracy obtained from both datasets is not consistent due to the additive distortion to the ground truth values during up sampling.

Fig. 5 and 6 suggest that as video resolution increases, the processing time (from storing the pixel until a histogram is formed) for the serial LBP-TOP algorithm increases exponentially. The time taken to compute the histogram for all the videos with highest tested resolution from SMIC and CASMEII database is 10000 seconds and 31000 seconds respectively.

The CPU utilization window showed that only 1/24th of the logical processors was used to execute the program. This has proven that the CPU was not fully utilized even though it has hardware that supports multicore and hyperthreading.

As the video resolution increases, the total number of pixels needed to obtain its LBP Code also increases, resulting in a longer computational time. Hence, to shorten the execution time, a more time-efficient programming model is required.

C. Bernstein's Conditions

Prior to parallelize a program, it is crucial to identify whether the bottleneck of the program is parallelizable or otherwise by examine it using Bernstein's Conditions. The repetitive process of getting the LBP Code of each pixel takes the longest time in the entire program. Bernstein's Conditions are examined to ensure that the process of computing the LBP Code of each pixel is independent of each other. Fig. 7 shows the pixel values within a frame while Fig. 8 shows the computation of the LBP code of two pixels as circle in red and yellow in Fig. 8.

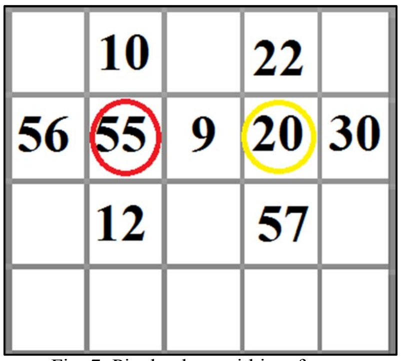

Fig. 7: Pixel values within a frame.

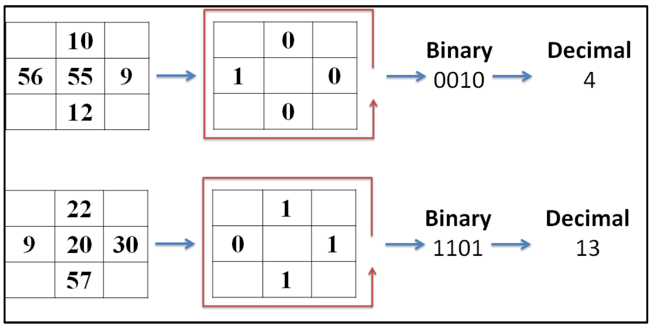

Fig. 8: Computation of LBP Code of 2 pixels.

From Fig. 8, suppose that I represents a set of neighbor points required to compute the LBP Code of the center pixel O represents the LBP Code of the center pixel. Hence, let

$$I_0 = \{56, 10, 9, 12\} \quad \text{where } O_0 = \{4\}; \text{ and}$$
$$I_1 = \{9, 22, 30, 57\} \quad \text{where } O_1 = \{13\} \tag{3}$$

$$\text{Since} \quad I_0 \cap O_0 = \emptyset, \text{ and} \quad I_1 \cap O_0 = \emptyset, \text{ and} \quad O_0 \cap O_1 = \emptyset, \tag{4}$$

Since all 3 Bernstein's Conditions are satisfied, hence it has proven that the data involved in computing the LBP Code of pixels has no dependency on each other. Thus, it has been confirmed that the bottleneck part of the serial LBP-TOP algorithm can be executed simultaneously which in other words, it is parallelizable.

D. Potential Performance Improvement

Prior to parallelization, the potential performance improvement on serial LBP-TOP algorithm is first examined. Using Amdahl's law, the expected improvement is calculated as follows:

$$\text{Speed Up}, S_p = \frac{1}{r_s + \frac{r_p}{N}} \tag{5}$$

where r_s represents the ratio of non-parallelizable portion, r_p represents the ratio of parallelizable portion and N is the number of threads. Equation 5 is used to obtain the potential theoretical speedup when $N > 1$ and the maximum theoretical speedup when $N \approx \infty$. Fig. 9 illustrates the maximum theoretical speedup can be obtained from parallelizing the LBP-TOP algorithm at different video resolutions. Fig. 9 suggests that the impact of parallelization is notable, whereby a maximum theoretical speedup of 266× can be achieved for SMIC dataset (black circle) and 303× for CASMEII dataset (purple square) when N ≈ ∞.

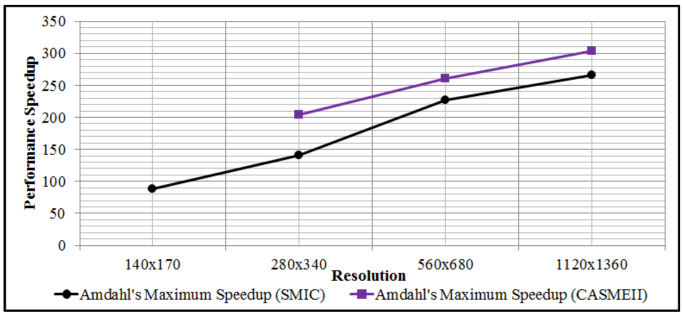

Fig. 9: Amdahl's Maximum Speedup (N≈∞).

III. PARALLEL LBP-TOP ON A MULTICORE ARCHITECTURE

To further improve the performance of serial LBP-TOP, this section describes the design and development of parallel LBP-TOP algorithm on a multicore architecture using OpenMP application programming interface. The program flow for this parallel algorithm is akin to the serial LBP-TOP algorithm, except on the workload distribution in computing the LBP Code of pixels within a video as it involves performing the same tasks on different pixels.

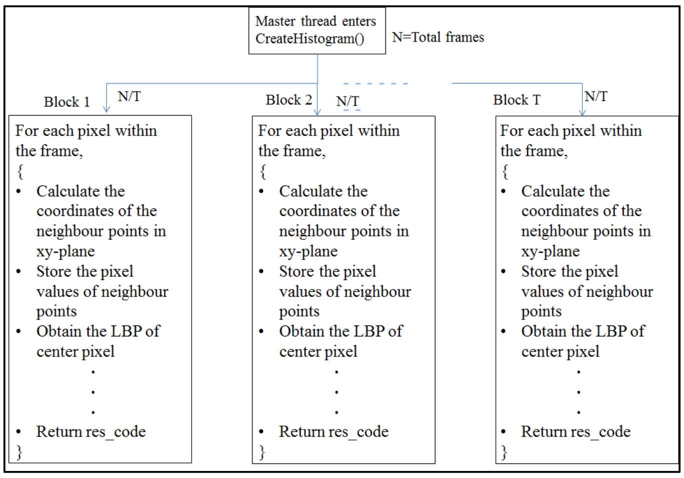
Master thread enters CreateHistogram()
N=Total frames
Block 1
N/T
Block 2
N/T
Block T
N/T
For each pixel within the frame,
{
Calculate the coordinates of the neighbour points in xy-plane
Store the pixel values of neighbour points
Obtain the LBP of center pixel
Return res_code
}

Fig. 10: Execution Model of computing LBP Code using OpenMP.

Fig. 10 portrays the execution model of the parallel LBP-TOP algorithm using OpenMP. T numbers of software application threads are created to compute the LBP Code of pixels based on N/T workload distribution. T represents the number of available logical processors on a multicore architecture while N represents the number of frame of a video. Atomic operation is performed when updating the array which stores the histogram bins to prevent different threads write to the same memory location at the same time.

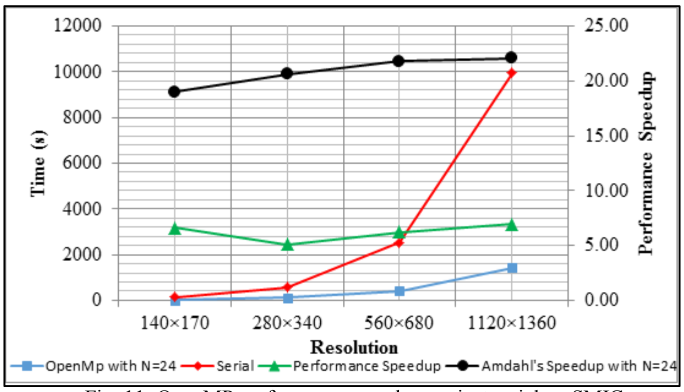

Fig. 11: OpenMP performance speedup against serial on SMIC.

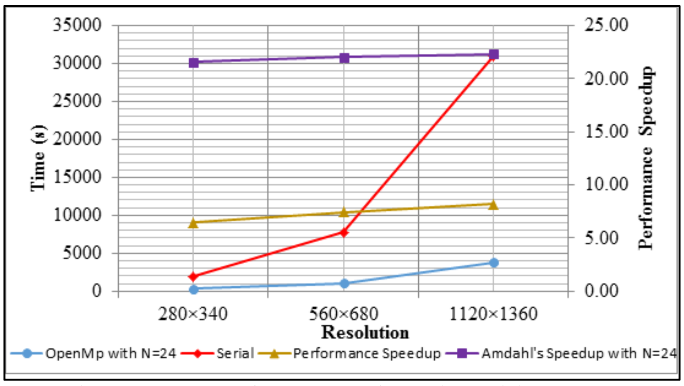

Fig. 12: OpenMP performance speedup against serial on CASMEII.

With the same test setup and dataset in Section 2, the performance of parallel multicore LBP-TOP algorithm was compared with the serial algorithm on SMIC dataset. Based on the equation 5, the Amdahl's theoretical speedup with N=24 for SMIC and CASMEII is portrayed in Fig. 11 (in black circle) and Fig. 12 (in purple square) respectively. It shows that a maximum speedup of 22× can be achieved for both datasets. Fig. 11 also includes the performance of serial algorithm (in red diamond), performance of OpenMP (in blue square) and the actual performance speedup of OpenMP over serial (in green triangle). Result of the actual performance shows significant improvements in OpenMP implementation with computational time of 1400 seconds to compute the histogram for 164 videos (resolution 1120×1360). This interprets into a performance speedup of 7× against serial implementation. Furthermore, the 560×680, 280×340 and 140×170 resolutions also highlight a notable performance speedup of 6.3×, 6× and 6.6× respectively.

Similarly, Fig. 12 also includes the performance of serial algorithm (in red diamond), performance of OpenMP algorithm (in blue circle) and also the actual speedup of OpenMP against serial (in brown triangle). From Fig. 12, the performance speedup of OpenMP against serial increase linearly when the video resolution increases. Performance speedup of 6×, 7× and 8× against serial implementation were obtained for 280×340, 560×680 and 1120×1360 videos. This has highlighted a noteworthy performance speedup for OpenMP implementation.

However, the actual speedup attained in each resolution is lower than the theoretical speedup suggested by Amdahl's law for both datasets. For instance, the maximum theoretical speedup for the video with highest tested resolution suggested by Amdahl is 22× but in reality only 8× is attained for both datasets. This difference may contribute by the atomic operation in updating the histogram as when more than 1 thread is attempting to write to the same location the program has degenerated into serial. In addition, threading overhead and memory latency may also contribute to these differences. Further speedups are attainable if T increases. This motivates the concept of leveraging many core architecture with CUDA platform to further speed up the algorithm which will discuss in the following section.

IV. Parallel LBP-TOP Using CUDA On A Many Core Architecture

To further improve the performance of LBP-TOP algorithm, this section proposes a parallel LBP-TOP algorithm design on a many-core GPU architecture using the CUDA parallel computing platform. In CUDA, launching a kernel typically generates a large number of threads to execute the same code in parallel. The threads are grouped into blocks, which in turn are grouped into a grid. Thus, the kernel is executed as a grid of blocks of threads.

Since each video in both databases has different number of frames. Also, the number of pixel within the block may larger than the maximum number of threads per block as the video's resolution becomes higher. For instance, dividing a video with 1120×1360 resolutions into 5×5 blocks will result in 224×272 per block which is larger than the maximum allowable number of threads within a block in CUDA (1024). Therefore, the block is further divided into smaller tiles where the tile dimension is equal to the block of threads. Each thread within a block is assigned to compute the LBP Code of a pixel. In order to avoid creating abundant threads and blocks, a function is created to identify the number of threads per block (the size of the tile) and the number of blocks per grid required for CUDA kernel when dealing with different video resolutions. The algorithm of the function is described in Algorithm 1.

Algorithm 1: Threads and blocks calculations

```
tx = roi_w – (2 × Rx)
ty = roi_h – (2 × Ry)
if tx × ty > 1024
      remain= 1024 / tx
      iterator i = remain
      While ty % i != 0 do
            decrease the iterator by 1
      end
      ty=i
bx = 1
by = (roi_h (2 × Ry)) / ty
bz =  tlength – (2 × Rt)
```

In Algorithm 1, the **tx** and **ty** represent the number of threads in x and y direction respectively. Likewise, **bx**, **by** and **bz** represent the number of blocks in x, y and z direction respectively. The width and height of the block of images is denoted as **roi_w** and **roi_h**. While the radius in x, y and t direction is represented as **Rx**, **Ry** and **Rt**.

With Algorithm 1, the number of threads per block and number of blocks per grid required by different video resolutions are tabulated in Table I. However, the number of blocks required in *z* direction depends on the number of frames per video.

TABLE I Number Of Threads Per Block And Number Of Blocks Per Grid For Different Video Resolutions.

Video Resolution	Number of thread per block (x, y)	Number of block per grid (x, y)
140×170	26, 32	1, 1
280×340	54, 11	1, 6
560×680	110, 2	1, 67
1120×1360	222, 3	1, 90

Fig. 13 illustrates the execution model of the LBP-TOP algorithm using CUDA. An array (in the GPU global memory) stores the value of each pixel within a block of images. Each block of threads will process one tile where each thread within a block will compute the LBP Code of a pixel within the tile. The pixel value is copied from GPU global memory into the register of a thread to allow faster access by thread. Similar to the multi-core implementation of the LBP-TOP algorithm in Section III, atomic operation is utilized when updating the array which stores the histogram bins. The function of atomic operation in CUDA minimizes race condition as the operation is performed without interference from other threads.

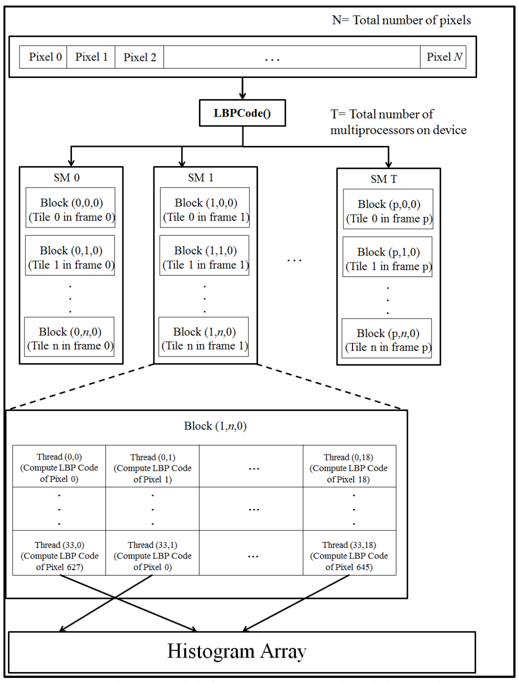
N= Total number of pixels
Pixel 0
Pixel 1
Pixel 2
...
Pixel N
LBPCode()
T= Total number of multiprocessors on device
SM 0
SM 1
SM T
Block (0,0,0) (Tile 0 in frame 0)
Block (0,1,0) (Tile 1 in frame 0)
Block (0,n,0) (Tile n in frame 0)
Block (1,0,0) (Tile 0 in frame 1)
Block (1,1,0) (Tile 1 in frame 1)
Block (1,n,0) (Tile n in frame 1)
Block (p,0,0) (Tile 0 in frame p)
Block (p,1,0) (Tile 1 in frame p)
Block (p,n,0) (Tile n in frame p)
Block (1,n,0)
Thread (0,0) (Compute LBP Code of Pixel 0)
Thread (0,1) (Compute LBP Code of Pixel 1)
Thread (0,18) (Compute LBP Code of Pixel 18)
Thread (33,0) (Compute LBP Code of Pixel 627)
Thread (33,1) (Compute LBP Code of Pixel 0)
Thread (33,18) (Compute LBP Code of Pixel 645)
Histogram Array

Fig. 13: Execution model for LBP-TOP algorithm using CUDA.

TABLE II NVIDIA QUADRO M6000 DEVICE PROPERTIES.

Compute capability	5.2
Global memory	24578Mbytes
CUDA cores	3072 (128 CUDA cores/SM)
Streaming multiprocessors (SM)	24
Warp Size	32
Maximum number of threads per block	1024
Maximum number of threads per SM	2048
Maximum number of blocks per SM	32

With the similar test setup and dataset in the aforementioned sections, the performance of parallel LBP-TOP algorithm using CUDA was carried out. The GPU used in this project is NVIDIA Quadro M6000. The GPU device property is summarized in Table II.

The maximum number of blocks per SM is 32 as shown in Table 2. When processing videos with lowest tested resolution, each block is launched with 832 threads, therefore only 2 blocks can reside in an SM as the maximum number of threads per SM is 2048. Since there are 24 SMs on this device, a maximum of 48 blocks can be executed simultaneously. In contrary, for a video with highest tested resolution, 666 threads per block is launched, 3 blocks can reside in an SM. With 24 SMs on the device, a maximum of 72 blocks can be executed concurrently. The rest of the blocks will be in queues.

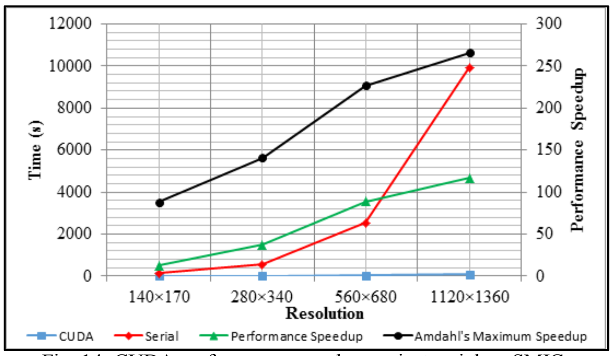

Fig. 14: CUDA performance speedup against serial on SMIC.

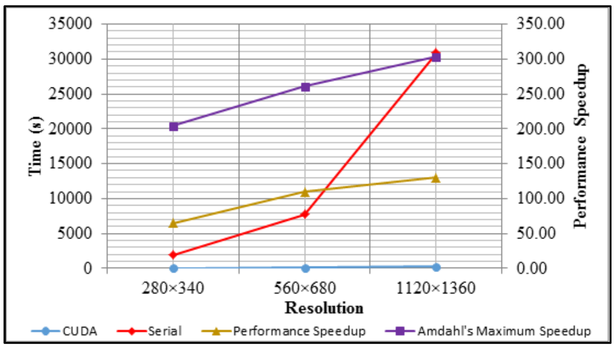

Fig. 15: CUDA performance speedup against serial on CASMEII.

The execution time for the LBP-TOP algorithm includes the time taken for data transfer between the host and the device. Fig. 14 and 15 compares the performance of the proposed parallel LBP-TOP algorithm using CUDA with the serial LBP-TOP algorithm for SMIC and CASMEII database respectively.

Fig. 14 illustrates the performance of serial algorithm (in red diamond), performance of CUDA (in blue square) and the actual performance speedup of CUDA over serial (in green triangle) for SMIC dataset. According to the equation 5, Amdahl's theoretical speedup with N=∞ says that a maximum speed up of 270× can be achieved as illustrated in Fig. 15 (in black circle). The actual performance speedup shows significant improvements in CUDA implementation with computational time of 85 seconds to compute the histogram for 164 videos with the highest tested resolution. A performance speedup of 110× against serial implementation is obtained. Furthermore, the 560×680, 280×340 and 140×170 resolutions also highlight a notable performance speedup of 90×, 40× and 10× respectively.

Fig. 15 portrays the performance of serial algorithm (in red diamond), performance of CUDA (in blue circle), the actual performance speedup of CUDA over serial (in brown triangle) also the Amdahl's theoretical speedup with N=∞ (in purple square) for CASMEII dataset. The actual performance speedup shows notable improvements in CUDA implementation against serial. The time taken to compute the histogram for 245 videos with highest tested resolution has shortened to 237 seconds which interprets into a performance speed up of 303×. In addition, for 280×340 and 560×680 resolution, performance speed up of 65× and 110× were achieved respectively.

Note that the actual speedup attained in each resolution for both datasets is lower than the theoretical speedup suggested by Amdahl's law. For instance, the maximum theoretical speedup of 1120×1360 videos in SMIC suggested by Amdahl is 270× but in reality only 110× is achieved which is about 59% lower than the theoretical speedup. The percentage differences between the actual and theoretical speedups increases as the video resolution decreases. The high percentage difference here is primarily attributed to latencies in data transfers between host (CPU) and device (GPU) memories. Also, Amdahl's theoretical speedup only considers the computational time, in reality, both the communication and computational time are taken into account. A lower resolution video would require fewer GPU resources since the number of threads required is lesser. This in turn translates into a reduced computation versus communication time ratio, which results in an overall smaller actual performance speedup [9]. In addition, the atomic operation performed while updating the histogram has degenerated the program into serial may also contribute to these differences.

The performance between the proposed parallel LBP-TOP algorithm using CUDA and the benchmarked multi-core parallel LBP-TOP algorithm can be assessed further by comparing the computational performance between the two algorithms, Fig. 16 and 17, on both SMIC and CASMEII databases respectively. Both Fig. 16 and 17 also compute the performance speedup between the benchmarked and proposed parallel LBP-TOP algorithms (green triangle). The resultant of these figures shows that the proposed parallel LBP-TOP algorithm using CUDA performs 17× and 16× faster for SMIC and CASMEII respectively over the multi-core parallel LBP-TOP algorithm.

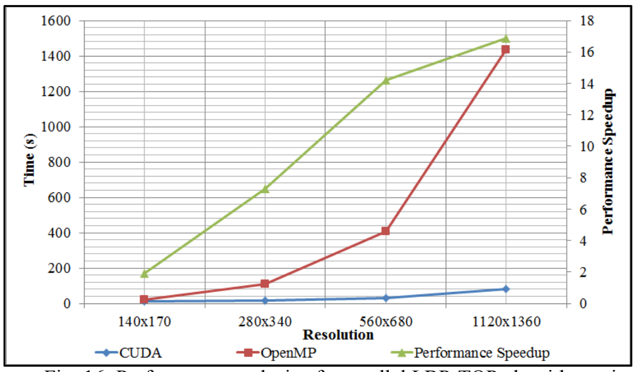

Fig. 16: Performance analysis of a parallel LBP-TOP algorithm using CUDA against a parallel LBP-TOP on a multicore architecture for SMIC.

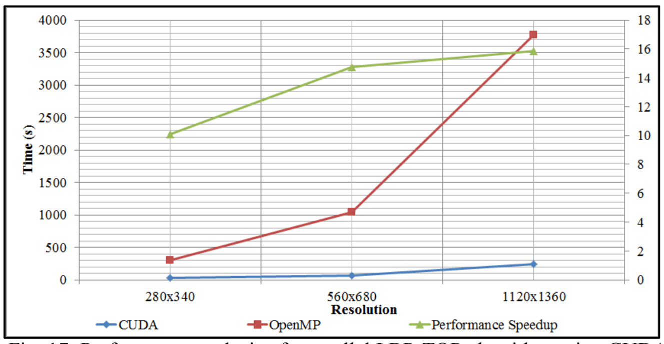

Fig. 17: Performance analysis of a parallel LBP-TOP algorithm using CUDA against a parallel LBP-TOP on a multicore architecture for CASMEII.

V. CONCLUSION

In this paper, the algorithm of a serial LBP-TOP was analyzed to ascertain its limitations in processing high resolution video. A multi-core version using OpenMP was implemented as benchmark. A performance speedup of 7× and 8× were obtained against a serial LBP-TOP algorithm while computing a 1120×1360 video from SMIC and CASMEII dataset respectively, these performance speedup is limited by the number of logical processors on a multi-core architecture. Therefore, this paper proposed a parallel many-core LBP-TOP algorithm using the CUDA parallel computing platform. Before implementing the LBP-TOP algorithm using CUDA, a function has created to calculate the threads and blocks required. With this function, no redundant threads and blocks will be created therefore it maximizes the computational performance as Streaming Multiprocessors do not need to accommodate the idle threads. The proposed algorithm improved the performance speedup to 117× and 130× against a serial LBP-TOP algorithm for SMIC and CASMEII databases respectively. Howbeit, the performance speedup is limited by the memory latency between CPU and GPU.

For future work, the proposed CUDA LBP-TOP algorithm could be further optimized by using multiple GPUs within a single machine or across several machines.

REFERENCES

[1] P. Ekman, "Darwin, deception, and facial expression," annals of the New York Academy of Sciences, vol. 1000, no. 1, pp. 205-221, 2003.

[2] W.-J. Yan, Q. Wu, J. Liang, Y.-H. Chen, and X. Fu, "How Fast are the Leaked Facial Expressions: The Duration of Micro-Expressions," Journal of Nonverbal Behavior, vol. 37, no. 4, pp. 217–230, 2013.

[3] S. Porter and L. T. Brinke, "Reading Between the Lies: Identifying Concealed and Falsified Emotions in Universal Facial Expressions," Psychological Science, vol. 19, no. 5, pp. 508–514, 2008.

[4] S.-T. Liong, J. See, K. Wong and R. C.-W. Phan, "Less is More: Micro-expression Recognition from Video using Apex Frame," arXiv preprint arXiv: 1606.01721, 2016.

[5] Y. Wang, J. See, R. C.-W. Phan, Y.-H. Oh, "LBP with Six Intersection Points: Reducing Redundant Information in LBP-TOP for Micro-expression Recognition," in Asian Conference on computer Vision, pp. 525-537, 2015.

[6] S. Moore and R. Bowden, "Local binary patterns for multi-view facial expression recognition," Computer Vision and Image Understanding, vol. 115, no. 4, pp. 541–558, 7 January 2011.

[7] G. Zhao and M. Pietikainen, "Dynamic Texture Recognition Using Local Binary Patterns with an Application to Facial Expressions," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 29, no. 6, pp. 915–928, 2007.

[8] W.-J. Yan, X. Li, S.-J. Wang, G. Zhao, Y.-J. Liu, Y.-H. Chen, and X. Fu, "CASME II: An Improved Spontaneous Micro-Expression Database and the Baseline Evaluation," PLoS ONE, vol. 9, no. 1, 27 January 2014.

[9] Buhari, Adamu Muhammad, Huo-Chong Ling, Vishnu Monn Baskaran, and Koksheik Wong. "Real-time high-resolution downsampling algorithm on many-core processor for spatially scalable video coding." Journal of Electronic Imaging, vol. 24(1), pp. 01325-0-01325-11, Feb 2015.