International Journal of Electrical and Computer Engineering (IJECE)
Vol. 12, No. 2, April 2022, pp. 1548~1557
ISSN: 2088-8708, DOI: 10.11591/ijece.v12i2.pp1548-1557

Super-linear speedup for real-time condition monitoring using image processing and drones

Moath Alsafasfeh[1], Bradley Bazuin[2], Ikhlas Abdel-Qader[2]
[1]Department of Computer Engineering, College of Engineering, Al-Hussein Bin Talal University, Ma'an, Jordan
[2]Department of Electrical and Computer Engineering, College of Engineering and Applied Sciences, Western Michigan University, Kalamazoo, United States

Article Info

Article history:

Received Sep 30, 2020
Revised Oct 20, 2021
Accepted Nov 4, 2021

Keywords:

Large-scale solar system
Multiprocessing
Real-time inspection
Superliner speedup
Thermal video

ABSTRACT

Real-time inspections for the large-scale solar system may take a long time to get the hazard situations for any failures that may take place in the solar panels normal operations, where prior hazards detection is important. Reducing the execution time and improving the system's performance are the ultimate goals of multiprocessing or multicore systems. Real-time video processing and analysis from two camcorders, thermal and charge-coupling devices (CCD), mounted on a drone compose the embedded system being proposed for solar panels inspection. The inspection method needs more time for capturing and processing the frames and detecting the faulty panels. The system can determine the longitude and latitude of the defect position information in real-time. In this work, we investigate parallel processing for the image processing operations which reduces the processing time for the inspection systems. The results show a super-linear speedup for real-time condition monitoring in large-scale solar systems. Using the multiprocessing module in Python, we execute fault detection algorithms using streamed frames from both video cameras. The experimental results show a super-linear speedup for thermal and CCD video processing, the execution time is efficiently reduced with an average of 3.1 times and 6.3 times using 2 processes and 4 processes respectively.

Corresponding Author:

Moath Alsafasfeh
Department of Computer Engineering, Al-Hussein Bin Talal University
University Road King Hussien Bin Talal University Str، Ma'an, Jordan
Email: moath.alsafasfeh@ahu.edu.jo

1. INTRODUCTION

Many real-time applications including video processing need an algorithm to be executed in parallel on multicore or a multiprocessor system. Multicore or multiprocessor with parallel programming is used to address performance improvement. To achieve such improvements, efficient utilization of thread-level parallelism is elemental. In fact, the ability to divide the tasks among a multicore or multiprocessor system is sub-linear, linear, or superliner speedups. A multicore system adds processing power with minimal latency which delivers significant performance benefits for software. This trend is shaping the future of software development toward parallel programming [1]. This benefit will be clear in applications which have huge input data and work in real time. Parallelism can be used at the system level by spreading the workload of the handling requests among the processors and disks. Data level parallelism (DLP) is enabled data parallel reads and writes via distributing data across many disks. Taking advantage of instruction level parallelism (ILP) via an individual processor is also critical to achieving high performance, and pipelining is the simplest way to do this. Parallelism can also be employed at the level of detailed digital design; for example, modern all-

optical arithmetic logic unit (ALU) use carry-lookahead, or set-associative caches [2]. The principle of locality is one of the most important program properties. Programs tend to reuse instructions and data they have used recently; a program spends 90% of its execution time in only 10% of the code. The idea of locality is that the prediction of instructions and data that a program will use in near future is based on its accesses in the recent past. The locality has two types; spatial locality says that items whose addresses are near one another tend to be referenced close together in time. Temporal locality says that recently accessed items are likely to be accessed in the near future [2]. Talk about speedup related to parallel processing, the speedup is estimated in comparison of the runtime of the best sequential program versus the run time of the parallel program [3] as defined in (1).

$$SpeedUp = \frac{run\ time\ of\ the\ best\ sequential\ program}{run\ time\ of\ the\ parallel\ program} \quad (1)$$

However, a speedup metric is defined by Amdahl's law in (2) which indicates that it depends on two factors; the fraction of the computation time that can be converted to take advantage of the enhancement ($Fraction_{enhanced}$). The second factor is the improvement gained by the enhanced execution mode ($Speed_{enhanced}$). This is equal to the time of the original mode over the time of the enhanced mode.

$$Speedup_{overall} = \frac{1}{(1- Fraction_{enhanced}) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}}} \quad (2)$$

In Amdahl's law, the task speedup cannot be more than the reciprocal of 1 minus the fraction if an enhancement is only usable for a fraction of a task. Amdahl's law can be considered as a guide to how much enhancement can be achieved. The goal is to utilize resources proportionally to where time is needed. The speedup that will be achieved by n cores is based on the proportion of the program/tasks executed in parallel versus in serial. The speedup of parallelizing any computing problem is limited by the percentage of the serial portion, which is also in agreement with Amdahl's law. Gustafson's trend is based on that once the problem size is increased; the processor power also tends to increase. Also, the drastic increase in the ratio of parallel-to-serial tasks in the computational load presents an equally dramatic increase in the processing requirements, which means once the computing resources increase, the problem size also increases, and thus the serial portion becomes much smaller [4]. Gustafson modified Amdahl's law putting forth that while the size of the overall problem should increase proportionally to the number of processors (n), the size of the serial portion (s) of the problem should remain constant as the number of cores increases, as given by (3).

$$Speedup = s + n(1 - s) \quad (3)$$

Superlinear speedup is defined as computation using n processors that could be more than the same computation performed on a uniprocessor [5]. The speedup will be more than (n). There are many factors leading to this superlinear phenomena these include the increase in cache size where each processor has a local cache level 1 or level 2; hidden latency in communications; the different speeds of memory inherent in distributed memory ensembles, the shifting in time fraction spent on different-speed tasks [6]; the utilization of resources more efficiently that comes hand-in-hand with parallelization [7], and fitting the data in caches of multiple data nodes by partitioning the data.

In this work, we are using infrared as well as charge-coupling devices (CCD) videos for defect inspection in a solar system. Infrared images have been used for a wide range of applications including medical imaging, nondestructive testing, and quality controls. Other applications include helping firefighters and police to find warm bodies in search areas. With the development of image acquisition technology, the image is of higher quality such as image resolution. However, this leads also to increase in demands on memory and time. The high-resolution images extracted from videos at 60 frames per second, required a multicore system in order to process them for real-time systems [8]. We combined two videos and using several image processing algorithms to inspect solar panels in real time.

Regarding to to the literature review, the existing techniques of using multiprocessing for image processing are presented. Mostly, images will require some pre-processing for noise removal or extraction of certain features and/or segmenting the image that even leads to more tasks to be accomplished. For example, the image segmentation process is one of the primary steps of extracting different objects or regions. The larger the images, the higher the computational time for the segmentation process [9]. Happ *et al*. [9] enhanced the segmentation process of an image using a multicore processor and their results show a speedup.

The segmentation algorithm by Baatz [10] was improved by [9] used parallel processing, where the image is divided into tiles (regions). Using the sequential algorithm, one thread is utilized to process a local region growing for each one tile [9]. Once the image is divided into tiles and then the work divided into

threads, these should impact the final segmentation results. The number of threads should always be equal to the number of available cores. Three different sizes for the input images, 2800x2800, 2000x2000, and 1000x1000, the testing environment was on an Intel core 2 quad with speed 2.40 GHz, and 2 GB of RAM. The results show speed ups to around 1.5 times and 2.5 times using 2 threads and 4 threads respectively [9]. Saxena *et al*. [11] represented the sequential image processing algorithms using multicore processor by the parallel implementation, such as segmentation, histogram equalization, and noise reduction. The input images are dividing into different tiles equal to the number of threads cores or the number of cores. Each core or thread processed its tile and paid attention to the synchronization within the processor. The input image resolutions are 256x256, 256x768, and 128x843. The testing environment was intel core i3-2350 M Processor 2.30 GHz, 3 GB of RAM, and hard disk drive 320 GB Software with a 64-bit operating system. They used also matrix laboratory (MATLAB) R2011a and JAVA JDK 1.6.0_21 and. The results show that the parallel processing is better than sequential processing by 1 time. The results also show that for some algorithms the improvement reached 2 times [11].

Liu and Gao used a parallel programming tool for the implementation of the interpolation of the cubic convolution algorithm in images, for example OpenMP and threading building blocks (TBB) utilizing a multicore processor [12]. They also compared between the sequential and parallel implementations. The results show that the cubic algorithm is improved 200% and 400% using of Dual-core and Quad-core respectively compared with sequential implementation [12].

Kamalakannan *et al.* [13] proposed multithreaded color image processing using fuzzy method versus edge detection including contrast enhancement. They proposed simultaneous processing for equal blocks using separate cores where the entire image has been partitioned into blocks [13]. Their work tested using input images were 10 images of different pixel size using Core i5 Quad-core. The results show that using a four-thread model improved the performance 3.4 times over a sequential method.

2. RESEARCH METHOD

In this proposed system, we use the acquired videos from both the thermal and CCD cameras. In python and using OpenCV, we determine the length and the number of frames of the input video in offline processing. Figure 1 shows the main steps for video segmentation process in order to process each segment in by individual processes simultaneously. Ffmpeg is used for video portioning process using the following command which it is embedded in python code.

$$os.system('ffmpeg\ -ss\ ' + str(from\_time) + '\ -t\ ' + str(cutting\_perioed) + '\ -i\ ' + input\_video + '\ ' + output\_dir + '/' + file\_names + str(file\_num) + '.mp4')$$

Ffmpeg is installed with python and it is used to calculate the cutting interval by determining the duration of the input video using (4).

$$Cutting_{period} = \frac{input_{file_{duration}}}{number_{processes}} \quad (4)$$

The segmentation process for thermal and CCD videos is started simultaneously in a while loop, by which the starting time, initialized at zero, is determined, then it increased by the cutting period as shown in (5), the cutting period is decreased from the duration of the input video as shown in (6).

$$from_{time} = from_{time} + cutting_{period} \quad (5)$$

$$input_{file_{duration}} = input_{file_{duration}} - cutting_{period} \quad (6)$$

Multiple segments will be generated and stored in a specific path after the video is divided. In python, the number of processes is initialized using the multiprocessing module. Each specified video frames are celled using OpenCV by its specific process and start running simultaneously, Figure 2 shows the running diagram for the multiprocessor module in python. A while loop in each process can read frames from the specified video portion frames. During the reading of frames, the image processing operations for the fault detection algorithm will be started in each process. All processes are running simultaneously with the same operations; each process should exit from the execution after completing its specific task with no waiting for another process to tackle.

In this paper, the detection of the defects in the PV module and determining the longitude and latitude for the location of the solar panel is done using image processing algorithms. Different types of

defects in the PV modules are detected by implementing the different proposed algorithms. On different detection algorithms, different input data is implemented separately, and each algorithm is briefly presented in the following sections to show the computing demands.

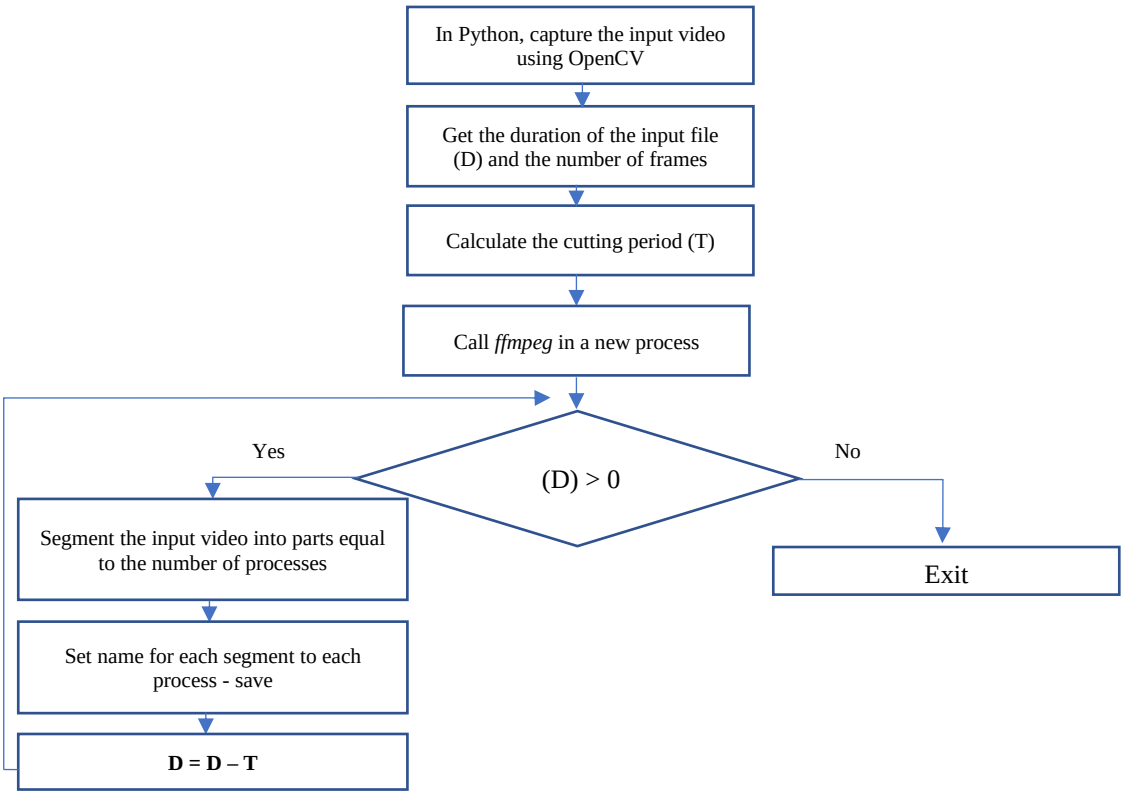

Figure 1. Videos processing using a multicore system

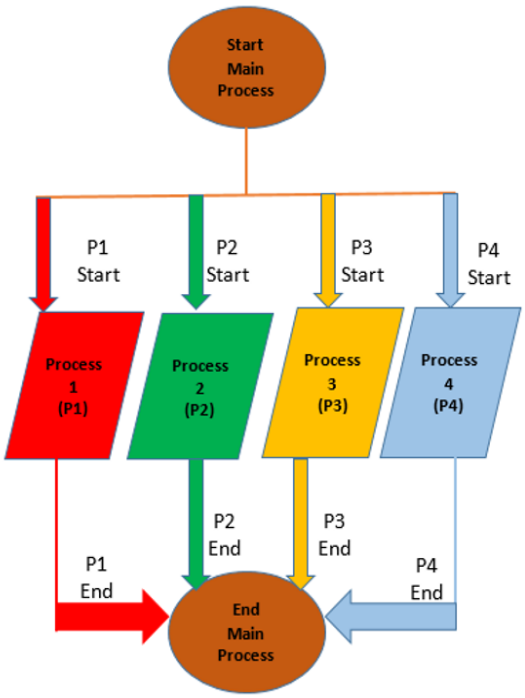

Figure 2. Running tasks by multiprocessing module in python

2.1. Morphological transformation with canny edge detector

In computer vision-based applications, canny edge detection is used to extract useful structural information from different objects which reduces the amount of data to be processed [14], and a canny detector is used to get the accurate information of the target object [15]. In this paper, a canny edge detector is used where the input image is converted to a binary image. Then the threshold process is applied on each frame. The value of the threshold, Th, is determined adaptively, and it was re-estimated for each frame in some experiments. A kernel (structuring element) is assigned to implement the morphological transformations [16] and followed by canny edge detection algorithm to detect the defective cells in the solar panel. Edge detection using canny algorithm provides excellent performance results in many practical problems, and it is considered an optimal edge detection algorithm [17].

In this paper, canny algorithm to be applied to identify significant intensity discontinuities in the image. The main idea is finding the direction of the gradient at each pixel. This can be done by finding the first derivative for the horizontal and the vertical directions using the soble filter. The (7) and (8) show the edge gradient and the angle calculations for each pixel respectively [18]. The Gradient direction is perpendicular to the edges; its value is rounded to one of four angles representing diagonal directions, horizontal or vertical [18].

$$Edge_{Gradient(G)} = \sqrt{(G_x^2 + G_y^2)} \tag{7}$$

$$Angle\ (\theta) = \tan^{-1}(\frac{G_y}{G_x}) \tag{8}$$

After computing the image gradients, the unwanted pixels should be removed by scanning the image in order to identify which pixels do not constitute the edges [18]. The last step is the thresholding of the edges. This can be done by using two values for thresholding, minimum (Th_{min}) and maximum (Th_{max}) values. Comparing computed gradients with these two Thresholding values, edges are identified under the conditions in (9). Using a morphological transformation and canny edge algorithm to monitor the real-time operations of solar panel and detect faults is introduced in [19].

$$Edges_{Declaration} = \begin{cases} Intensity_{gradiant} > Th_{max}, & Sure\_Edge \\ Intensity_{gardiant} < Th_{min}, & Sure\_not\_Edge \end{cases} \tag{9}$$

2.2. SLIC super-pixel algorithm

K-mean clustering is used to implement the spatial localization which is the main concept of simple linear iterative clustering (SLIC) super-pixel technique. Recently, superpixel algorithms are widely used for computer vision and multimedia applications, such as in [20] to close all the contours and reserve coherence across image boundaries. In addition, SLIC is used in the hyperspectral image (HSI) to solve the small sample problem [21]. Using SLIC, the image can be decomposed into small homogeneous regions, providing a perceptual understanding of content by locally grouping the pixels. The image complexity, thousands of thousands of pixels, is reduced to only a few hundred of pixels using super-pixel [22]. In order to minimize the outliers in SLIC which they would skew the results, a gaussian smoothing filter is used as a preprocessing phase.

Super-pixels is generated to effectively propose SLIC by Achanta *et al.* [23]. The desired number of approximately equally sized superpixels, k is the main parameter of the SLIC algorithm. Initializing cluster centers (C_k) at regular grid step is the first step in SLIC by sampling pixels using (10), the number of pixels is presented in N. Then, (11) is used to calculate the distance between the cluster center and the pixel. The cluster is moving to the lowest gradient position in a 3x3 neighborhood, the seed location, for each pixel in the 2Sx2S region around for each cluster center (C_k).

$$Seg. = \sqrt{\frac{N}{k}} \tag{10}$$

$$D = \sqrt{\left(\frac{d_s}{S}\right)^2 m^2 + \left(d_c{}^2\right)} \tag{11}$$

SLIC corresponds to clusters in *labxy* color space, where the color and spatial distances should be calculated using (12) and (13) respectively. They are combined in (14) in order to normalize color and spatial proximities by their respective maximum distances with a cluster, N_s and N_c.

$$d_s = \sqrt{\left(x_j - x_i\right)^2 + \left(y_j - y_i\right)^2} \quad (12)$$

$$d_c = \sqrt{\left(l_j - l_i\right)^2 + \left(a_j - a_i\right)^2 + \left(b_j - b_i\right)^2} \quad (13)$$

$$D' = \sqrt{\left(\frac{d_s}{N_s}\right)^2 + \left(\frac{d_c}{N_c}\right)^2} \quad (14)$$

The sampling interval value S is considered the maximum spatial distance N_s within a given cluster. From image to image and cluster to cluster, the color distance can be different so the constant value m in (11) is considered as the maximum color distance N_c. The new cluster centers will be computed when the pixel is assigned to the nearest cluster, then the distance is recalculated until the residual error between the new and the previous cluster center is less than the threshold value. Using SLIC to monitor the real-time operations of solar panel and detect faults is introduced in [24].

2.3. Hot pixels seeds based for segmentation

An image can be divided into constitutive parts or objects is called the segmentation process [25]. Segmentation the image provides many operations to be implemented on the image, such as object classification and recognition, the clusters identification, features of similarity or discontinuity between different pixels such as edges and lines [25]. The first step of the proposed segmentation method is determining a seed pixel S_P, (hot pixel), in the input image. The threshold would be more difficult due to the low contrast problem, it is solved by the pre-processing processes by Gaussian filter and histogram equalization for the input images. After image pre-processing, setting the value of the highest pixel is done using (15). The (16) is used to determine where the neighboring pixels are linked to the hot pixel, assigning them as seed pixels S_P, or to the background pixels B_P.

$$Hot_{Pixel} = MAX\ (pixel[row, column]) \quad (15)$$

$$S_P = \begin{cases} pixel[row, column] \ \geq (Hot_{Pixel} - margin), & Sure\_S_P \\ pixel[row, column] < (Hot_{Pixel} - margin\ ), & Sure\_B_P \end{cases} \quad (16)$$

The mean value μ_{S_P} for each seed pixel S_P is calculated using (17). For each seed region with 8 neighboring pixels of the S_P , the mean value μ_{S_P} is computed.

$$\mu_{S_P} = \frac{(\sum_{row=0,column=0}^{9,9} pixel[row,column)}{9} \quad (17)$$

At the same time, the average value for all hot pixels μ_{hot_pixels} for each thermal frame is computed. However, the value of hot pixels for the CCD frames is assigned to μ_{hot_pixels}=127 which is a value that worked fine in the most cases. An adaptive method for the selection of these parameters should be investigated further and developed in the near future. The actual seed pixel Act_S_p is determined using (18).

$$Act\_S_p = \begin{cases} \mu_{S_P} \geq \mu_{hot_{pixels}} & Sure_{Act\_S_p} \\ \mu_{S_P} < \mu_{hot\_pixels}\ , & B_P \end{cases} \quad (18)$$

Computation of the standard deviation using (20) to estimate the minimal deviation distance (MDD) based on (19) for each actual hot pixel.

$$MDD = \ min\ ((\Omega_{Act\_S_p})^2) \quad (19)$$

$$\Omega_{Act\_S_p} = \ \left|S_p - pixel\ [row, column]\right| \quad (20)$$

The selection of B_P to be defected or not is based on (22). For each background pixel with its' 8 neighbors the mean value μ_{B_p} is estimated, then delta value δ is computed using (21). The B_P is assigned as a defected pixel if MDD value is greater than (δ); otherwise, $B_P$is considered as a (zero) pixel. Using hot pixels seeds-based segmentation to monitor the real-time operations of solar panel and detect faults is introduced in [19].

$$(\delta = \left| mean_{B_p} - \mu_{S_P} \right|) \quad (21)$$

$$Defected_{B_P} = \begin{cases} \delta \leq \text{MDD}, \text{ defected pixel } B_P = 1 \\ \delta > \text{MDD}, \text{ not\_defected pixel } B_P = 0 \end{cases} \quad (22)$$

3. RESULTS AND DISCUSSION

The proposed system proves the use of multicore processors reduces the required execution time for real-time operations. In this paper, the results show that the importance of using a multicore processor with parallel processing using python is reducing the inspection time for large-scale solar system monitoring and detecting hazards. The system has two cameras; the FLIR Vue Pro is a thermal camera which has an accurate thermal resolution with 336x256 pixels which is high enough to show defects on solar panels, and with (NTSC) frame rate. GoPro Hero 4 Black is a CCD camera was used in the system; the camera has the max video resolution 3840x2160 and effective photo resolution 12.0 MP. These two cameras are connected on the Yuneec Q500 quadcopter. The input data were processed and implemented the offline system using python 2.7 and the eclipse IDE platform on a windows 10 environment, where the processor is Intel (R) Core (TM) i5-4210 M CPU with speed 2.60 GHz, and with 8 Gigabyte RAM. Other python modules, extensions and libraries are installed using a pip command; Multiprocessor module, matplotlib, NumPy, and Pillow. OpenCV is used with python for providing multiple modules for image processing.

A multicore system has been used for simultaneous thermal and CCD videos processing to detect defects in the solar panel with a reduction of the execution time. The results of defects detection are explained in previous work [19], [24]. The inspection process has been made on real experiments where the drone was flying on panels that were imposed with internal and external defects. The experiments were conducted outdoor in the daytime where the thermal camera would be able to detect the defects in the nighttime. The drone was flying on normal mode without specifying the angel where the altitude was different for many scenarios. Thermal frames and CCD frames are processed for the same panels at the same time. In this paper, the results are recorded for different scenarios for fault detection algorithms in PV systems, using 1 process, 2 processes, or 4 processes.

Table 1 shows the input thermal and CCD videos and the number of processed frames. A different number of frames is shown because the input videos have a different size. Multiprocessing module by python is used to process the input videos and improve the execution time which it is reduced significantly, where the whole system's performance is improved. The processing time is recorded after the segmentation process is completed.

Table 1. Input of thermal and CCD videos for defects detection

Input Video	Thermal Video			CCD Video		
	Size (MB)	# Frames	# Processed Frames	Size (MB)	# Frames	# Processed Frames
V_1	4.76	456	60	219	1832	60
V_2	7.66	856	120	418	349	120
V_3	11.4	1440	200	715	5968	200

Table 2 presents the processing time of using morphological transformation with canny edge detector where the faults can be detected in solar panels using thermal and CCD videos. This execution was done by using 1 process, 2 processes, and 4 processes with the speedups illustrated in Figure 3. The processing time was improved 3.5, and 4.2 times using 2 and 4 processes respectively. Table 3 presents the processing time of using SLIC super-pixel for different size of segments, 50 and 200 with maximum10 iterations for k-mean, where the defects are detected in the solar panel using thermal and CCD videos. This execution was done by using 1 process, 2 processes, and 4 processes with the speedups shown in Figure 4. The processing time was improved 3.2 and 8.2 times using 2 and 4 processes respectively.

Table 2. Processing time for morphological and canny edge detection execution for thermal and CCD videos using multicore

Input video	Processing time (in Min.) based on # of Processes			Speedup	
	P1	P2	P4	(P1/P2)	(P1/P4)
V_1	4.78	1.34	1.07	3.57	4.47
V_2	8.01	2.18	1.93	3.67	4.15
V_3	12.56	4.01	3.07	3.13	4.09

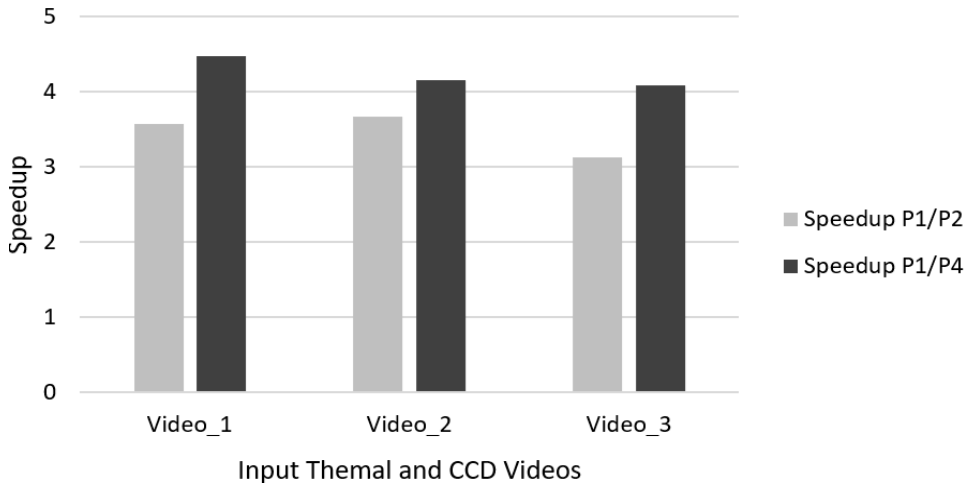

Figure 3. Speedup results of using morphological with canny edge detection algorithm for thermal and CCD videos using multicore

Table 3. Processing time for SLIC super-pixel execution for thermal and CCD videos using multicore

Input video	Processing time (in Min.) based on # of Processes			Speedup	
	P1	P2	P4	(P1/P2)	(P1/P4)
V_1	64.18	21.84	9.02	2.94	7.12
V_2	144.95	46.3	17.96	3.13	8.07
V_3	567.57	163.99	60.64	3.46	9.36

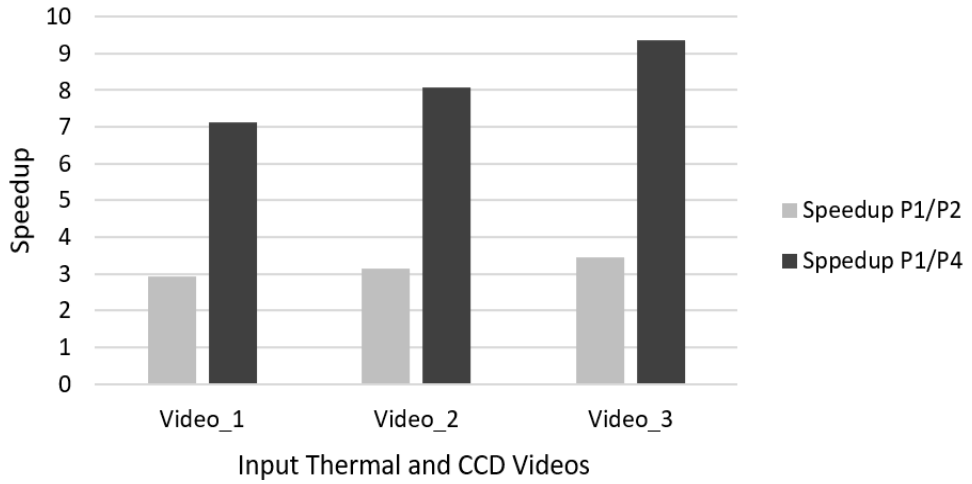

Figure 4. Speedup for SLIC super-pixel for thermal and CCD videos using multicore

Table 4 presents the processing time of using the defects detection algorithm, hot pixel seeds based for segmentation. The defects are detected in solar panels using thermal and CCD videos. The achieved speed up is shown in Figure 5 where the execution was done by using 1 process, 2 processes, and 4 processes. Using a multiprocessing module improved the execution time 2.7 and 6.4 times using 2 and 4 processes respectively.

Table 4. Processing time for hot pixels based for segmentation for thermal and CCD videos using multicore

Input Video	Processing time (in Min.) based on # of Processes			Speedup	
	P1	P2	P4	(P1/P2)	(P1/P4)
V_1	64.78	25.41	8.9	2.56	7.28
V_2	107.58	36.1	18.6	2.98	5.78
V_3	196.1	76.06	31.67	2.58	6.19

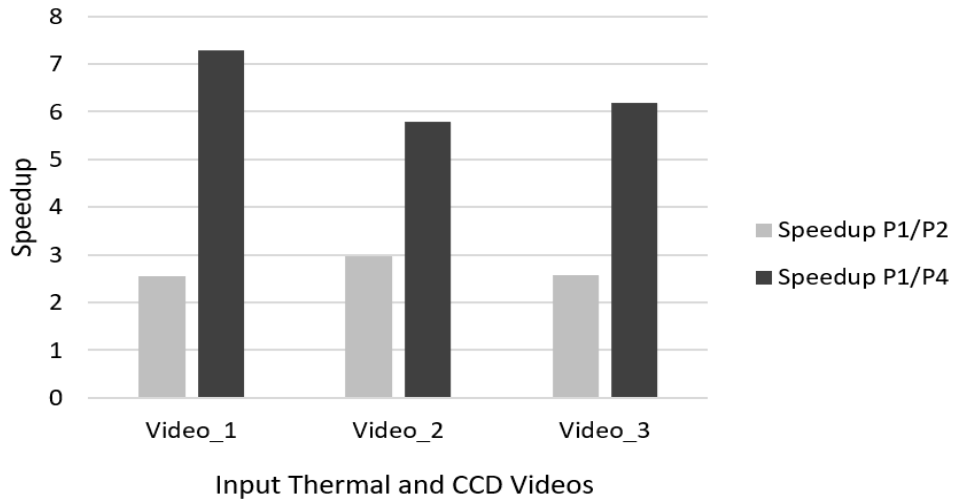

Figure 5. Speed up for hot pixels based for segmentation for thermal and CCD videos using multicore

4. CONCLUSION

Real-time condition monitoring of large-scale solar system needs more processing time in order to monitor and detect faults. The inspection system is implemented using thermal and CCD cameras and the processing time was very long without using parallel processing. This problem is solved in this paper where the captured videos are proceeded using multiple processes simultaneously which reduces the execution time. The speedup we achieved with image processing algorithms is a very significant improvement. The average improvement for the processing time was 3.1 times and 6.3 times using 2 processes and 4 processes respectively. This is due to many reasons including the problem size is large (the number of processed frames), and once the execution time for each frame is long, the speedup using simultaneous processes resulted in a superlinear speedup. The results show that when the problem size is divided into portions and executed among processes simultaneously, the execution time will have a significant reduction and result in a superlinear speedup. In addition, the computer resource utilization will be more effective once the problem is divided into portions; for example, the cache effect will take place once the problem is divided into more than one process via multicore CPU and run simultaneously.

REFERENCES

[1] B. Goldworm, "Multi-core processor benefits and trends," *Techtarget.com*, 2007. http://searchitchannel.techtarget.com/feature/Multi-core-processor-benefits-and-trends (accessed Sep. 17, 2021).
[2] J. L. Hennessy and D. A. Patterson., *Computer architecture: A quantitative approach*. 5th ed. Waltham, MA, USA, Morgan Kaufmann, 2012.
[3] S. Akhter and J. Roberts, *Multi-core programming*. Hillsboro: Intel press, 2006.
[4] M. Gillespie, "Amdahl's law, Gustafson's trend, and the performance limits of parallel applications," *intel.com*, 2008. https://software.intel.com/en-us/articles/amdahls-law-gustafsons-trend-and-the-performance-limits-of-parallel-applications (accessed Sep. 17, 2021).
[5] D. P. Helmbold and C. E. McDowell, "Modelling speedup (n) greater than n," *IEEE Transactions on Parallel and Distributed Systems*, vol. 1, no. 2, pp. 250–256, Apr. 1990, doi: 10.1109/71.80148.
[6] J. L. Gustafson, "Fixed time, tiered memory, and superlinear speedup," in *Proceedings of the Fifth Distributed Memory Computing Conference, 1990.*, 1990, vol. 2, pp. 1255–1260, doi: 10.1109/DMCC.1990.556383.
[7] J. Shan, *Superlinear speedup in parallel computation*. CCS, Northeastern Univ., Boston, MA, USA, Course Report, Tech. Rep., 2002.
[8] X. Hongye and G. Ruilin, "Multi-channel thermal infrared image edge detection method under multi-core environment," in *2016 Eighth International Conference on Measuring Technology and Mechatronics Automation (ICMTMA)*, Mar. 2016, pp. 405–408, doi: 10.1109/ICMTMA.2016.103.
[9] P. Happ, R. Ferreira, C. Bentes, G. Costa, and R. Feitosa, "Multiresolution segmentation: a parallel approach for high-resolution image segmentation in multicore architectures," *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 38, no. 4, 2010.
[10] M. Baatz and A. Schape, "Multiresolution segmentation: An optimization approach for high quality multi-scale image segmentation," *Angewandte Geographische Informationsverarbeitung XII. Wichmann*, pp. 12–23, 2000.
[11] S. Saxena, N. Sharma, and S. Sharma, "Image processing tasks using parallel computing in multi core architecture and its applications in medical imaging," *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 2, no. 4, pp. 1896–1900, 2013.
[12] Ying Liu and Fuxiang Gao, "Parallel implementations of image processing algorithms on multi-core," in *2010 Fourth International Conference on Genetic and Evolutionary Computing*, Dec. 2010, pp. 71–74, doi: 10.1109/ICGEC.2010.26.

[13] A. Kamalakannan and G. Rajamanickam, "High performance color image processing in multicore CPU using MFC multithreading," *International Journal of Advanced Computer Science and Applications*, vol. 4, no. 12, 2013, doi: 10.14569/IJACSA.2013.041207.
[14] X. Li, X. Wang, A. Yang, and M. Rong, "Partial discharge source localization in GIS based on image edge detection and support vector machine," *IEEE Transactions on Power Delivery*, vol. 34, no. 4, pp. 1795–1802, Aug. 2019, doi: 10.1109/TPWRD.2019.2925034.
[15] H.-W. Lee, "The study of mechanical arm and intelligent robot," *IEEE Access*, vol. 8, pp. 119624–119634, 2020, doi: 10.1109/ACCESS.2020.3003807.
[16] "Morphological transformations," *Opencv.org*. https://docs.opencv.org/master/d9/d61/tutorial_py_morphological_ops.html (accessed Sep. 17, 2021).
[17] B. Green, "Canny edge detection tutorial," *Donntu.com*. http://masters.donntu.org/2010/fknt/chudovskaja/library/article5.html (accessed Sep. 17, 2021).
[18] "Canny edge detection," *Opencv.org*. http://docs.opencv.org/3.1.0/da/d22/tutorial_py_canny.html#gsc.tab=0 (accessed Sep. 17, 2021).
[19] M. Alsafasfeh, I. Abdel-Qader, B. Bazuin, Q. Alsafasfeh, and W. Su, "Unsupervised fault detection and analysis for large photovoltaic systems using drones and machine vision," *Energies*, vol. 11, no. 9, Aug. 2018, Art. no. 2252, doi: 10.3390/en11092252.
[20] Q. Zhao, F. Dai, Y. Ma, L. Wan, J. Zhang, and Y. Zhang, "Spherical superpixel segmentation," *IEEE Transactions on Multimedia*, vol. 20, no. 6, pp. 1406–1417, Jun. 2018, doi: 10.1109/TMM.2017.2772842.
[21] Y. Zhang, K. Liu, Y. Dong, K. Wu, and X. Hu, "Semisupervised classification based on SLIC segmentation for hyperspectral image," *IEEE Geoscience and Remote Sensing Letters*, vol. 17, no. 8, pp. 1440–1444, Aug. 2020, doi: 10.1109/LGRS.2019.2945546.
[22] X. Zhang, S. E. Chew, Z. Xu, and N. D. Cahill, "SLIC superpixels for efficient graph-based dimensionality reduction of hyperspectral imagery," in *SLIC superpixels for efficient graph-based dimensionality reduction of hyperspectral imagery*, May 2015, Art. no. 947209, doi: 10.1117/12.2176911.
[23] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk, "SLIC superpixels compared to state-of-the-art superpixel methods," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 11, pp. 2274–2282, Nov. 2012, doi: 10.1109/TPAMI.2012.120.
[24] M. Alsafasfeh, I. Abdel-Qader, and B. Bazuin, "Fault detection in photovoltaic system using SLIC and thermal images," in *2017 8th International Conference on Information Technology (ICIT)*, May 2017, pp. 672–676, doi: 10.1109/ICITECH.2017.8079925.
[25] I. Abdel-Qader, S. Yohali, O. Abudayyeh, and S. Yehia, "Segmentation of thermal images for non-destructive evaluation of bridge decks," *NDT and E International*, vol. 41, no. 5, pp. 395–405, Jul. 2008, doi: 10.1016/j.ndteint.2007.12.003.