Parallel Image Segmentation Using Map-Reduce Framework

Mohammad Nishat Akhtar, Junita Mohamad Saleh*, Elmi Abu Bakar and Ayub Ahmed Janvekar

Abstract— As a result of the expansive information set size of high-resolution image data, most desktop workstations do not have sufficient configurable scheduling to perform image processing assignments in a convenient manner due to which the image processing tasks are meant to be divided into straight forward assignments. The processing power of any regular computing machine in this way becomes a severe bottleneck with respect to high execution time and low throughput. Many image processing tasks exhibit a high level of information region and parallelism and map quite readily to a parallel computing system. This paper shows an alternative to sequential image processing by introducing Map-Reduce technique to segment multiple images with the help of Hadoop framework. The evaluation of the proposed scheduling algorithm is done by implementing parallel image segmentation algorithm to detect lung tumor for up to 1 GB size of CT image dataset. The results have shown improved performance with parallel image segmentation when compared to sequential image segmentation method particularly when data capacity reaches a particular threshold. This is because the process of parallel image processing has been able to exploit the multi-cores thread level parallelism which ultimately gave the CPU usage with octacores up to 96%, hence reducing the task execution time up to approximately 1.6 times compared with the sequential style of image segmentation using Map-Reduce algorithm implemented with FIFO scheduler. The proposed parallel image segmentation design has shown to be useful for researchers at performing bulk image segmentation in parallel, which can save tremendous execution time.

Keywords— Hadoop, Execution Time, Task parallelism, Parallel Image Segmentation, Map, Reduce.

I. Introduction

High end computing machines have not been savvied enough (as far as the necessary equipment and programming speculation) to increase across the broad usage. Maybe, it appears that in near future, parallel computing will be dominated by medium-grain distributed memory machines in which every processing node will have the capabilities of a desktop workstation [1]. In reality, as network innovations keep on maturing, bunches of workstations are themselves being progressively seen as a parallel computing asset. The upsides of medium-grain standard computing are low cost and high quality. The disadvantage comprises irregular load designs on the processing nodes [2]. The proposed research depicts the outline and implementation of parallel image segmentation using Hadoop framework. It is also worth to be noted that Hadoop framework is not based on the model of Message Passing Interface (MPI) standard and is specifically designed to support parallel execution on heterogeneous workstation nodes [3, 4]. Many image processing algorithms exhibit natural parallelism in a sense that the input image data required to compute a given portion of the output is spatially localized and is compatible to be implemented on a cloud framework [5, 6]. In the simplest case, the output image could be computed simply by independently processing single pixels of the input image.

Image Processing with parallel computing is a viable approach to take care of image processing issues that require extensive processing time [6, 7]. It is evident that restorative imaging requires heaps of memory space and time to process, so by parallelizing, it would be helpful to discover productive and quick outcome. In parallel processing, a program can make numerous assignments that cooperate to take care of the issue of multi-tasking [8]. Parallel image processing cannot be connected to all issues, or in other words it can be stated that not every one of the issues can be coded in a parallel shape. A parallel program ought to must have a few elements for a right and proficient operation; else, it is conceivable that run-time may not have the normal execution. These components incorporate the processing parameters such as granularity, coarse grained and fine-grained parallelism [9]. The remaining parts of this manuscript are arranged as follows. Section II highlights the background for parallel image segmentation. Section III describes the model for multiple image segmentation simultaneously. Section IV describes and demonstrates the proposed parallel image segmentation algorithm along with illustration of mapper and reducer for parallel image segmentation. Section V shows the results and discussion followed by a conclusion in Section VI.

Mohammad Nishat Akhtar is with School of Aerospace Engineering, Universiti Sains Malaysia, 14300, Nibong Tebal, Penang, Malaysia (email: nishat.akhtar2000@gmail.com)
*Junita Mohamad Saleh is affiliated with School of Electrical and Electronics Engineering, Universiti Sains Malaysia, 14300, Nibong Tebal, Penang, Malaysia (e-mail: jms@usm.my; Tel: +60194732732)
Elmi Abu Bakar is affiliated with School of Aerospace Engineering, Universiti Sains Malaysia, 14300, Nibong Tebal, Penang, Malaysia (email: meelmi@usm.my)
Ayub Ahmed Janvekar is affiliated with School of Mechanical Engineering, Universiti Sains Malaysia, 14300, Nibong Tebal, Penang, Malaysia (email: ayubjanvekar@gmail.com)

II. BACKGROUND

Over the years, multiple image segmentation algorithms have been used to analyze the images. Nowadays, wide range of algorithm is being used to carry out the process of image segmentation such as texture which is an essential feature that reflects important information about the image surface. The aim of image segmentation is to cluster the entire pixels into specified salient image regions, i.e., regions corresponding to individual objects, surfaces or natural part of objects. Segmentation is an essential process of object recognition, image compression, image database look-up and occlusion boundary estimation within stereo or motion system. The researchers these days are dealing with the problem of over segmentation of images which ultimately leads to inaccurate results and therefore, leaves a room for enhancing this problem with the help of principal component analysis techniques [10, 11]. The basic image properties dealt with image segmentation are its dissimilarity and similarity. Sharp changes in the intensity of image causes dissimilarity whereas similarity corresponds to the process of combining and matching the pixels with the neighboring one based on its gray level pixel value match and scale invariant feature transform [12, 13]. Some of the widely recognized techniques to implement image segmentation are; Otsu's threshold method for automated image segmentation, region growing and region merging technique, edge detection method, watershed transformation and histogram thresholding-based algorithms [14].

Amongst all the techniques, Otsu's method is widely renowned method to carry out the process of image segmentation. Since it is an automated process, therefore, it is easier to be applied on the bulk image data simultaneously. Since the proposed research is dealing with image data, therefore, it is appropriate to use OpenCV library and it is also to be noted that Otsu's threshold technique has high degree compatibility with OpenCV [15]. Furthermore, OpenCV has capability to exploit high degree of parallelism due to its available rich set of libraries [16]. This scenario makes the condition more favorable for parallel image processing in an efficient manner. There is also an API called Hadoop Interface for Image Processing (HIPI) which is an extensive set image processing framework and is only compatible with Hadoop Map-Reduce parallel programming model [17]. HIPI has full potential to accommodate high throughput image processing using Map-Reduce algorithm which can be implemented on a cluster of nodes. In order to perform segmentation process for multiple images in parallel, the following Section III will describe the segmentation model for multiple images.

III. SEGMENTATION MODEL FOR *N* IMAGES

Thresholding is considered to be an important technique for image segmentation which has got potential to identify and extract the target portion of an image from its actual background on the principal of distribution of gray levels in an image object. According to Otsu's method, an image is considered to be a two-dimensional grayscale intensity function which contains *N* pixels including gray levels ranging from 1 to *L* [18]. As per Otsu's analysis, the number of pixels having gray level '*i*' is denoted by 'f_i'. Therefore, the probability function (P_i) of gray level '*i*' in an image with *N* pixels could be written as (1) [19]:

$$P_i = f_i / N \tag{1}$$

For the analysis of bi-level thresholding of an image, the pixels could be divided into two classes C_1 and C_2 respectively. C_1 consists of first tier of gray level (1........,t) and C_2 consists of second tier of gray level ($t+1$............,L). Therefore, the gray level probability distribution for the two classes could be written as (2) and (3) [20]:

$$C1 = P_1/\omega_1(t)..................P_t/\omega_1(t) \tag{2}$$

$$C2 = P_{t+1}/\omega_2(t), P_{t+2}/\ \omega_2(t),.......P_L/\omega_2(t) \tag{3}$$

Where $\omega_1(t) = \sum_{i=1}^{t} P_i$ and $\omega_2(t) = \sum_{i=t+1}^{L} P_i$

Above grey level probability distribution method could also be applied for *M* number of classes assuming that there are *M*-1 thresholds, $\{t_1, t_2............,t_{M-1}\}$ which divide the original image into *M* classes: C_1 for [1......,t_1], C_2 for [t_1+1........,t_2].......,C_i for [$t_{i-1}+1$.........,t_i] and C_m for [$t_{M-1}+1$..........,L] [20].

Equation (4) represents a column vector:

$$X = \begin{matrix} X_1 \\ \vdots \\ X_n \end{matrix} \tag{4}$$

If the entered values in (4) are random pixel variables with a precise mean, then the segmented matrix [$seg(X_i, X_j)$] value $\sum$ is given by (5):

$$\sum\nolimits_{ij} = seg(X_i, X_j) = val[(X_i - \sigma_i)(X_j - \sigma_j)] \tag{5}$$

Where $\sigma_i = val(X_i)$ and $\sigma_j = val(X_j)$ are the assumed value of the i_{th} and the j_{th} entry in the vector *X*.

Now let us assume there are *n* such images to be segmented and if a single image is denoted by vector *x*, then the sample computed segmentation could be given by the formula in (6):

$$Seg = \frac{1}{n}\sum_{i}^{n}(x_i - \bar{x})(x_i - \bar{x})^T = \frac{1}{n}\hat{X}\hat{X}^T \tag{6}$$

Where i = index for the set of n images, $\bar{x}$ = average of n image pixels

Equation (6) could also be rewritten in matrix form using $\hat{X}$ to denote the mean centred images $(x_i - \bar{x})$ in (7)

$$\begin{pmatrix} \vdots & & \vdots \\ \hat{x}_i & \cdots & \hat{x}_n \\ \vdots & & \vdots \end{pmatrix} * \begin{pmatrix} \cdots & \hat{x}_1 & \cdots \\ \cdots & \vdots & \cdots \\ \cdots & \hat{x}_n & \cdots \end{pmatrix} \tag{7}$$

Let us divide the image patches into *v* number of pixels based on their similarity. On similarity basis, let us categorize the set of pixels into different clusters i.e., C_1, C_2,.....,C_v .

Now let us define the set group of every unsigned pixel which at least borders one of the clusters as defined in (8) [20]:

$$S = \{x \notin \bigcup_{i=1}^{v} C_i \bigwedge \exists k : N(x) \bigcap C_k \neq \emptyset\} \tag{8}$$

Here, *x* is the pixel to be assigned, where *N(x)* denotes the current neighboring pixel of point *x* which is a part of cluster C_k. As per (8), *x* does not lie the cluster C_i and *k* belongs to pixel *x* such that *N(x)* is a part of cluster C_k (Cluster with *k* pixels).

Now let us denote δ as the difference of measure between the pixels as defined in (9) [20]:

$$\delta(x, C_i) = |l(x) - mean_{y \in C} \tag{9}$$

Where *l(x)* denotes the pixel value of point *x* and *i* denotes the index of the cluster such that *N(x)* intersect C_i. *l(y)* denotes the pixel value of point *y*.
Now to select whether $q \in S$ and cluster C_j where $j \in [1,n]$ such that:

$$\delta(q, C_j) = \min_{x \in s, k \in (1,n)} \{(x, C_k)\} \tag{10}$$

Where *S* is defined in (8)

Now if $\delta(q, C_j)$ is lesser than the predefined threshold point t_p set by the programmer, the pixel is assigned to cluster C_j, else it must be assigned to another most considerable cluster *C* such that:

$$C = arg \min_{C_k} \{\delta(Z, C_k)\} \tag{11}$$

Now if $\delta(q, C_n) < t_p$, then the pixel is allocated to C_n . If neither of the condition is satisfied, then the formation of new cluster C_{n+1} takes place.
After the pixel has been allocated to the cluster, the mean pixel value of the cluster must be updated.

According to Gedraite and Hadad [21], the function which is used to generate the kernel is a Gaussian function comprising of 2 dimensions and could be defined using (12):

$$f(q, r) = a . e^{-\left\{\frac{(q-q_0)^2}{2\delta q^2} + \frac{(r-r_0)^2}{2\delta r^2}\right\}} \tag{12}$$

Where *q* and *r* are the vectors, *a* is the amplitude, (q_0, r_0) is the centre, δq and δr is the standard deviation in *q* and *r* direction.

Filter is defined using the variance of the Gaussian distribution. This parameter drastically affects the filtering results. The quality factor (*Q*) function defined for segmented image using Gaussian blur is given by the (13) [21]:

$$Q(s, t) = \frac{\sigma_{s.t}}{\sigma_s{}^2 . \sigma_t{}^2} . 2 . \frac{\bar{s} . \bar{t}}{s^2 + t^2} . \frac{2 . \sigma_s{}^2 . \sigma_t{}^2}{\sigma_s{}^2 + \sigma_t{}^2} \tag{13}$$

Where, *t* is the image without noise and *s* is the filtered image. $\sigma_{s.t}$ is the covariance between two images, $\sigma_s{}^2$ is the variance of filtered image and $\sigma_t{}^2$ is the variance of source image without noise. Here, $\bar{s}$ and $\bar{t}$ are the mean of images *s* and *t*. This quality factor determines the covariance between two images, the distribution in the contrast and distortion in luminance.

Now Section IV will illustrate the proposed parallel image segmentation algorithm along with the implementation of the Hadoop mapper and reducer to execute parallel image segmentation.

IV. Proposed Image Segmentation Framework

Hadoop Interface for Image Processing (HIPI) is an extensive set image processing API which is only compatible with Hadoop Map-Reduce parallel programming framework [22, 17].

The input images have been taken from Lung Image Database Consortium image collection (LIDC-IDRI) [23]. In order to implement the proposed parallel image segmentation algorithm, the input image files comprising of CT image samples is converted into HIPI format with HIB extension before it is passed to the main configuration files for mapping and reducing. Once the image file is successfully converted to the OpenCV compatible format (Mat), then the image file is passed to the mapper so as to enable the task distribution to the java threads. It is worth to be noted that prior to image data processing, the mapper ensures that the input images are in grayscale format.

Post channel check, numerous image processing functions are applied to segment the input images in parallel during the mapping phase. Post segmentation of bundled input images, the segmented data for each image is stored in a variable. The segmented data is in the form of Region of Interest (ROI) pixels. The stored ROI pixels data for each image is then passed to the reducer. Pseudocode in Fig. 1 represents the illustration of the mapper function.

In the Reduce phase, ROI pixels variable is received and is stored in an array list. The reducer then computes the average segmented pixels of all the input images before giving the final average pixel value. Once the reducer is done with final output, then the output image data gets stored in the HDFS. In Fig. 2, the illustration of the reducer is shown using the pseudocode.

For the pre-processing of image samples, we have converted the acquired input image from its original form to bilateral blur form in order to remove the noise from the image. Post noise removal, we try to apply image thresholding in order to get the estimated diseased portion area of the entire image. Post diseased portion estimation, the contour is drawn in order to get the clarity of the diseased portion detection. Fig. 3 shows the sample input CT image marked with tumor region. Let's say for *n* number of patients if there is a need to detect average number of tumor area in lungs for stage 2 cancer. Fig. 4 shows an illustration to segment *n* number of lung cancer images.

In order to segment multiple images parallelly using the proposed Map-Reduce algorithm, firstly all the input images are converted into HIB format and then, at stage1 a channel check is applied to ensure all images are in grayscale. At stage 2, the thresholding is applied on the lung images. For the proposed parallel image segmentation framework, optimal threshold is applied. At stage 3, segmentation of the lung images is done in parallel to highlight the ROI area which can also be called as tumor segmentation. At stage 4, in order to enhance the segmented image contour correction is applied to the ROI pixels and is passed to the reducer for feature extraction and analysis.

```
1.  Input image .JPG format
2.  Image conversion to HIB.Dat
3.  Pass image to Mapper<HipiImageHeader, FloatImage,
      IntWritable, IntWritable> //IntWritable is a Hadoop
      variant of integer
4.  Get image resolution
5.  Check image channel:  IF = 3
      THEN  covert to grayscale
6.  Set kernel size (width * height pixels)
7.  Apply Blur Filters to remove noise
8.  Image conversion to HSV(SourceImage, TargetImage, size,
      (0,0)); //(0,0 is the anchor point)
9.  Apply Otsu's Threshold (TargetImage)
10. Obtain ROI pixels; // (0[Black],  255[White]
       Pixels
9.   Store ROI pixels  //White/Black pixel
10.  Emit ROI pixels variable  to reducer
     context.write(new IntWritable(1),new
     IntWritable (ROI pixels));
```

Fig. 1 Illustration of mapper

```
1.  Reducer receives image Reducer<IntWritable, IntWritable,
      IntWritable,
      Text>
2.  Initialize a counter and iterate over IntWritable/int records
      from mapper
3.  Compute average segmented ROI pixel value
       // Emit output of job which will be written to HDFS context.
      write(key, new Text(result));
4.  Output the resultant  pixel value
```

Fig. 2 Illustration of Reducer

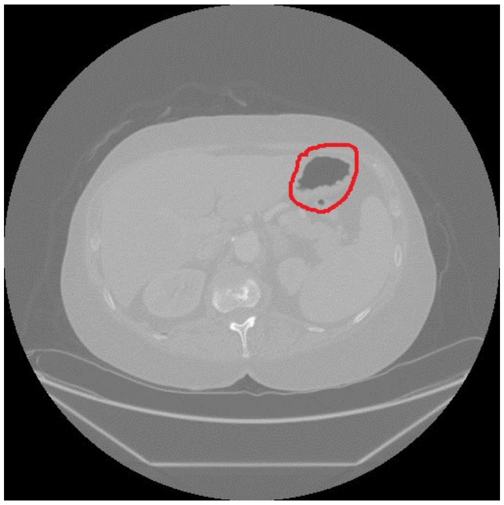

Fig. 3 Sample CT image marked with tumour region

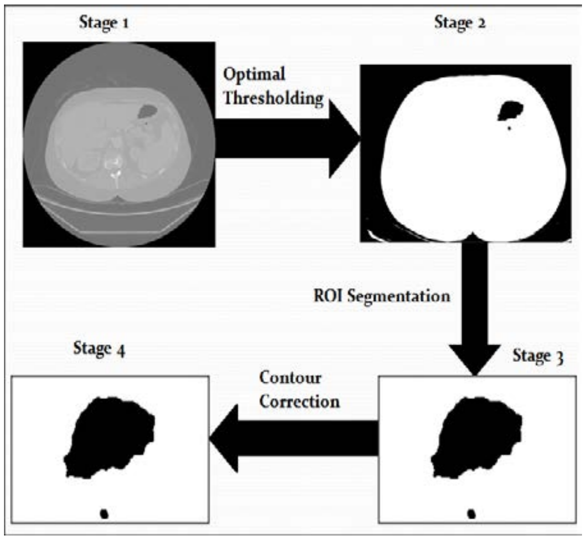

Fig. 4 Image segmentation process of lung tumour

V. RESULTS AND DISCUSSION

Our experiment setup consists of single node with a configuration of 8 GB RAM, 500 GB ROM, Intel i7, 3.4 GHz processor. For the proposed experiment, Hadoop has been implemented using Pseudo-Distributed mode where master node and slave node is encapsulated within a single machine. The master node is responsible for running the Job Tracker and Name Node while slave node is responsible to run Task Tracker and Data Node. The nodes were installed with Ubuntu 14.1 and Open JDK 1.6.0.24, and the execution environment was Hadoop 2.6.1.

Configuration of machine:

- Operating system: Ubuntu 14.1
- RAM: 8 GB
- Internal HDD (Dedicated to Ubuntu): 50 GB
- Processor: Intel i7 3.4 GHz- Sandy Bridge
- Level 2 Cache: 2048 KB

The configuration of Hadoop parameters are set as follows:

- HDFS Block Size: 128 MB
- No. of task mapping per node: 2
- No. of task reducing per node: 1
- Replication factor: 2
- Scheduler: FIFO

Experiments have been carried out using Lung Image Database Consortium image collection (LIDC-IDRI) [23].

A. *Performance Metrics*

It is worth to be noted that a series of performance indicators i.e., task execution time, CPU cores usage, throughput, impact of task mapping, impact of task reducing and accuracy are usually required to evaluate a Map-Reduce task. However, for the proposed experiment, our focus is on task execution time and CPU cores utilization. For the proposed experiment, all the 8 cores of the system is at the disposal of the Map-Reduce job by considering it a NP hard problem. Thus, an experiment was conducted to analyze the performance of Map-Reduce job by implementing parallel image segmentation for different image data sets taken from LIDC-IDRI ranging between 100 MB to 1 GB.

B. *Execution Time Analysis for Parallel Image Segmentation Using Hadoop*

For this study, an experiment is performed to run the task of image segmentation comprising of 100 MB, 200 MB, 400 MB, 600 MB, 800 MB and 1 GB image dataset using HIPI API ported in Hadoop framework. This should provide a clear understanding on the execution time of parallel programming mode for implementing image segmentation by using varying size of image dataset. The input split size divides the input bulk image dataset into multiple blocks. A Hadoop block is a file on the underlying file system. Each Hadoop block has one dedicated thread worker. For the proposed experiment, Hadoop version 2.6.1 has been used for which each block size is 128 MB. If a data block is filled completely to its capacity, then its associated thread worker is utilized 100%. The number of map tasks spawned depends on the number of blocks generated by the input split size. For the analysis purpose, the resolution of all the images has been kept intact for accurate results. Fig. 5 show the task execution time for various size of image datasets implemented using Hadoop framework.

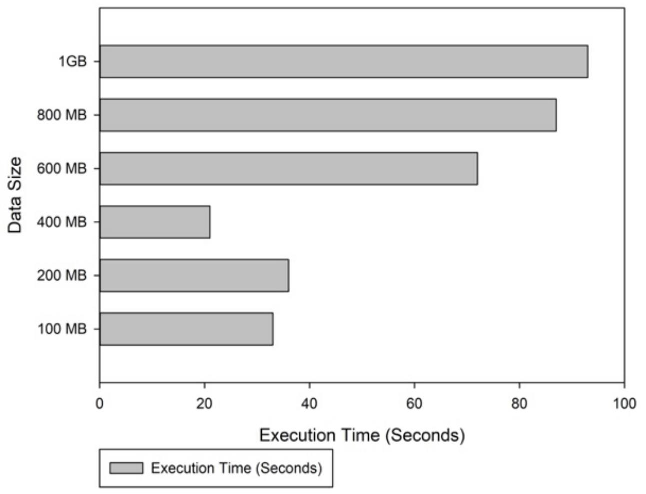

Fig. 5 Task execution time for various size of image dataset on Hadoop framework

C. *Image Datasets of 100 MB and 200 MB*

Firstly, the performance result using 100MB image data benchmark to evaluate the task execution time of Map-Reduce job is presented. This test was performed using BytesWritable data type and a constant key-value pair size of 1 KB with varying range of map and reduce task. For the implementation of 100 MB image dataset, it took 33 seconds to complete the job of image segmentation as observed in Fig. 5. The reason for the faster execution of 100 MB image data is due to the input split size. Parallel image segmentation implemented using Hadoop executes any job by dividing it into several blocks and each block has a fixed size [24]. For the version of Hadoop framework used in the proposed research, the block size value is set to 128 MB which is the fixed value in Hadoop version 2.6.1. Moreover, it has been verified by several benchmarks testing that 128 MB is the optimum block size value to execute the high-end data size jobs faster [7]. It is also worth to be noted that for all the previous version of Hadoop framework, the maximum value of block size was either 32 MB or 64 MB.

Therefore, let’s say for executing 50 MB image dataset which not even the half of 128 MB Hadoop block size, the number of input split(s) and the number of spawned map task is only 1 and minimum number of worker thread of the block i.e., 39.62% is allotted to execute the job whereas to implement the 100 MB image dataset, majority of the worker thread of the block i.e., 78.25% is allotted to execute the job as a result of which 100 MB image dataset takes less time to be executed. Now coming to the image dataset of 200 MB, which is divided into 2 Hadoop blocks as it is more than the single block size of 128 MB, the number of input splits and the number of spawned map task is 2 due to which worker threads of almost one and half blocks are allotted to execute the job as a result of which the completion time of the job for 200 MB image dataset is only 36 seconds as shown in Fig. 5.

D. Image Datasets of 400 MB and 600 MB

Now let us have a look at the 400 MB image dataset. From the graph in the Fig. 5, it could be observed that the total task execution time for 400 MB image dataset is 21 seconds. It is worth to be noted that Hadoop adjusts the division of 400 MB image dataset also into 3 blocks, therefore, the number of input splits and the number of spawned map task is equal to 3 as a result of which complete 100% threads of all the three blocks are used to execute the job. In addition to this, it could also be observed that there is a difference of 12 seconds in the total task execution time between 400 MB and 100 MB image dataset due to the fact that 100% threads of all the three Hadoop block are used to implement the 400 MB image dataset which is not the case with 100 MB image dataset.

Now let us analyze the 600 MB image dataset. From the graph in Fig. 5, it could be observed that the total task execution time for 600 MB image dataset is 72 seconds. The 600 MB image dataset gets divided into 5 Hadoop blocks, therefore, the number of split size and the number of spawned map task is again 5. The difference in the task execution time between 600 MB image dataset and 400 image datasets is again 39 seconds which is due to the fact that for 600 MB image dataset, apart from the 100% utilization of the threads of the first four Hadoop blocks, 70% threads of the fifth Hadoop block is utilized to execute the job of image segmentation in parallel.

E. Image Datasets of 800 MB and 1 GB

Now let us come to the 800 MB image dataset. From the graph in Fig. 5, it could be observed that the total time required to execute the task is 87 seconds. The 800 MB image dataset gets divided into 7 Hadoop blocks, therefore, the number of splits and the number of spawned map task is equal to 7 as a result of which 100% threads of the first six blocks and approximately 25% threads of the seventh Hadoop block is utilized to execute the job. Similarly for 1 GB image dataset, the total time required to execute the task was 96 seconds as observed from the graph in Fig. 5. The 1 GB image dataset was also divided into 7 Hadoop blocks, therefore, the number of splits and the number of spawned map task is equal to 7 as a result of which 100% threads of all the six Hadoop blocks is utilized to execute the job. For 1 GB image dataset and 800 MB image dataset, there is only a difference of 5% in the task execution time as observed in Hadoop log files due to the fact that 1 GB image dataset utilized complete thread usage of all the seven Hadoop blocks which in turn increases the degree of task parallelization.

F. CPU Cores Usage Analysis for Parallel Image Segmentation Using Hadoop

In this section, analysis of the CPU cores usage for the proposed parallel image segmentation on Hadoop framework along with different segment of task execution time is done. It is worth to be noted that in order to maximize the CPU cores usage up to its maximum, the data to be processed needs to be divided into several blocks so that the task parallelization could be increased. The more the data blocks, the more is the CPU cores usage. In addition to this, higher number of data blocks also increases the number of splits and the number of spawned map tasks. The following sub-sections discuss the distribution of CPU cores usage over various time segments for the implementation of parallel image segmentation for various sizes of image datasets using parallel image segmentation on Hadoop framework. Fig. 6 shows the maximum CPU cores usage attained for implementing the parallel image segmentation using Hadoop framework for various sizes of image datasets and Fig. 7 shows the distribution of overall CPU cores usage.

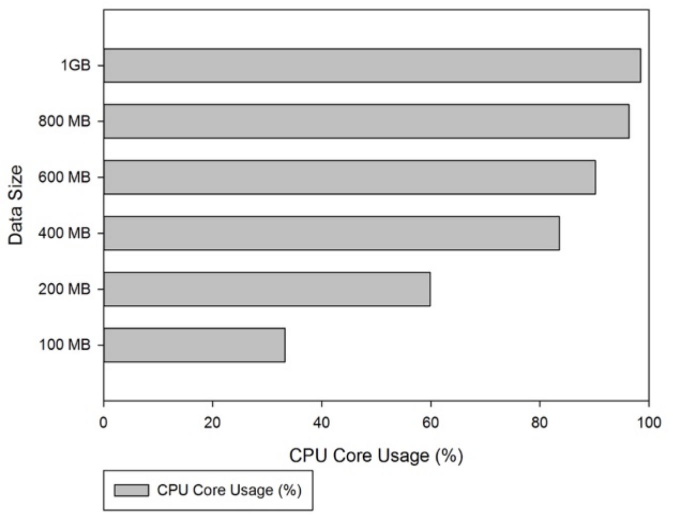

Fig. 6 Maximum CPU cores usage using parallel image segmentation on Hadoop framework

G. Image Datasets of 100 MB and 200 MB

Now from Fig. 6, it could be observed that, the maximum CPU cores usage value attained for 100 MB image dataset is 33.23%. As per Fig. 7, for 100 MB image dataset, it could be observed that the maximum CPU cores usage is attained at 15th second which is again almost the middle value of the total task execution time. It is worth to be noted that for 100 MB image data, majority of the threads of the 128 MB block is put into action to execute the job. However, since the size of the image dataset does not cross 128 MB, therefore, the number of input split and number of spawned map task is only 1.

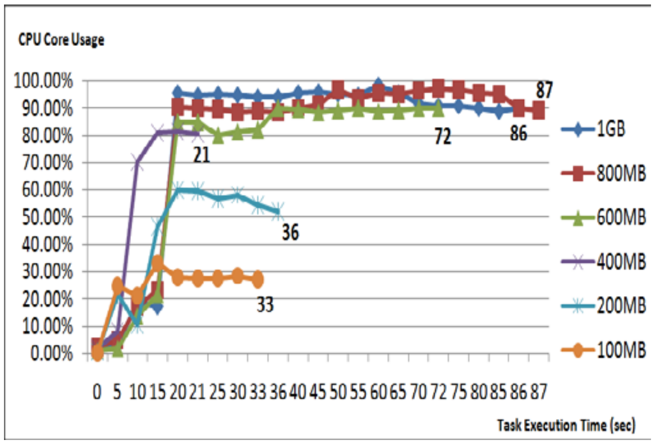

Fig. 7 Distribution of CPU cores usage using parallel image segmentation for 100 MB-1 GB image dataset on Hadoop distributed mode

Now let us focus on the 200 MB image dataset, from Fig. 6, it could be observed that the maximum CPU cores usage attained for 200 MB image dataset is 59.90%. Moreover, from the graph in Fig. 7 it could be observed that there is a wide gap between the maximum CPU cores usage between 100 MB image data and 200 MB image data. Since the 200 MB image dataset is greater than the single block size of 128 MB. Therefore, 200 MB image dataset is divided into 2 Hadoop blocks, as a result of which 75 % of the Hadoop threads in total of two blocks together are allotted to complete the task execution of 200 MB image dataset. For 200 MB image dataset, the number of split and the spawned map task is 2 since it is allotted 2 blocks. It is worth to be noted that maximum CPU cores utilization is achieved at 20th second which again lies at the middle of the total task execution time. It is worth to be noted that for all the two image datasets, i.e., 100 MB and 200 MB the maximum CPU cores utilization is achieved at the middle of the task execution time.

H. Image Datasets of 400 MB and 600 MB

Coming to the 400 MB image dataset, it could be observed from Fig. 6 that the maximum CPU cores usage attained for 400 MB image dataset is 83.58%. Moreover, for 400 MB image dataset, it could be observed from the graph in Fig. 7 that the maximum CPU cores usage is attained at the 12th second which lies again near the mid-point of the total task execution time and then after a stable CPU cores usage of more than 80% is observed till the finish time. For 400 MB image dataset, the number of split size and the number of spawned map task is equal to 4 which clearly specifies that the 400 MB image dataset is divided into 4 blocks. Therefore, 100% threads of the first three blocks and less than 20% threads of the fourth block are utilized to execute the job.

The maximum CPU cores usage attained for 600 MB image dataset is 90.17% as observed from Fig. 6. For 600 MB image dataset, again it could be observed from the graph in Fig. 7 that the maximum CPU cores usage is attained at 35th second which lies almost at the middle of the total task execution time and then it could be seen that there is a stable CPU cores usage of around 89-90%. The number of splits and number of spawned map tasks for the 600 MB image dataset is 5 which show that it is divided into 5 Hadoop blocks as result of which 100% threads of the first five blocks and more than 70% of the threads of the fifth block is utilized to execute the job.

I. Image Datasets of 800 MB and 1 GB

Now let us shift our focus to 800 MB image dataset. It could be observed from the graph in Fig. 6 that the maximum CPU cores usage attained for this dataset is 96.70%. The 800 MB image dataset gets divided into 7 Hadoop blocks, therefore, the number of input splits and the number of spawned map task is 7 as a result of which 100% threads of the first six Hadoop blocks and less than 25% threads of the seventh Hadoop block is utilized to execute the job. It could be observed from the graph in Fig. 7 that the maximum CPU cores usage for 800 MB image dataset is attained at the 50th second and soon after attaining this value there is a stable CPU cores usage of around 95-96%.

Now let us highlight at the 1 GB image dataset. It could be observed from the graph in Fig. 6 that the maximum CPU cores usage attained for this image dataset is 98.49%. It is worth to be noted that 1 GB image dataset gets divided into 8 Hadoop blocks, therefore, the number of input splits and the number of spawned map task is 8 as a result of which 100% threads of all 8 Hadoop are utilized to execute the job. It could be observed from the graph in Fig. 7 that the maximum CPU cores usage is attained at the 60th second. Moreover, for 1 GB image dataset, throughout the execution time a stable CPU cores usage of more than 90% could be observed at majority of the time segments. A difference of 2.21% could be observed in the maximum CPU cores usage value between 1 GB image dataset and 800 MB image dataset due to the fact that the seventh block thread is almost utilized up to 100% for 1 GB image dataset.

From the graph in Fig. 7, it could be observed that there is a sudden rise in the CPU cores usage after initial 5 seconds second which forms a spike like trend for all the size of image dataset. The reason for this trend is due to the fact that during initial 5 seconds to 10 seconds, Hadoop initializes the input data from the job tracker to the task tracker and during these initial seconds Hadoop daemons, i.e., name node, data node, job tracker and task tracker initiates.

J. Execution Time Analysis for Image Segmentation Using Sequential Programming

For this study, implementation for the task of image segmentation comprising of 100 MB, 200 MB, 400 MB, 600 MB, 800 MB and 1 GB image dataset using sequential style of programming on visual studio 2010 integrated development environment platform is done. This should provide a clear contrast between the difference in task execution time between sequential style of programming and parallel programming. This comparison will also enable us to analyze the threshold of the data size at which the task execution associated with various image datasets for parallel programming overcomes the task execution time associated with sequential style of programming. Fig. 8 shows the task execution time for various sizes of image datasets implemented using sequential style of programming to carry out the process of image segmentation.

K. Image Datasets of 100 MB and 200 MB

It could be observed from the graph in Fig. 8 that it takes 15.17 seconds to implement the image segmentation for 100 MB image dataset. It could be observed that the execution time has almost doubled for the 100 MB image dataset. If compared with the execution time of parallel image segmentation, the 100 MB image dataset took 33 seconds to complete the job. Let us now analyze the 200 MB image dataset.

It could be observed from the graph in Fig. 8 that in order to implement the 200 MB image dataset, it takes 32.5 seconds. Again, it is worth to be noted that the execution time almost got doubled if compared with the execution time of 100 MB image dataset. If compared with the execution time of parallel image segmentation, the 200 MB image dataset took 36 seconds to complete the job. It could be observed that for 200 MB image dataset, the execution time difference between sequential programming and parallel programming has narrowed down a lot if compared with previous image datasets due to the fact that with respect to parallel image segmentation, multiple threads provide a method of splitting the work between numerous cores.

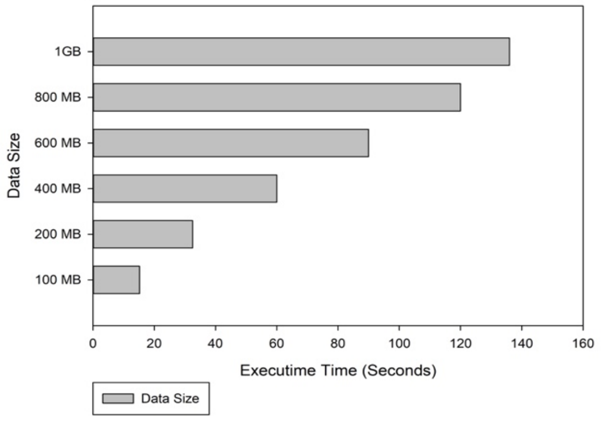

Fig. 8 Task execution time (sec) to implement image segmentation for various image dataset in sequential programming mode

For instance, if suppose a program that executes with a maximum efficiency with two threads on single core will most likely run at its peak with about four threads on dual cores. Now let us discuss about the analysis of the CPU execution time for the datasets whose size is more than 200 MB.

L. Image Datasets of 400 MB and 600 MB

As per Fig. 8, the total execution time to segment the 400 MB image dataset using sequential programming mode was 60 seconds whereas if compared with the execution time of parallel image segmentation, the 400 MB image dataset took only 21 seconds to complete the segmentation process in parallel manner. From the graph in Fig. 8, it could be observed that it took 90 seconds to complete the image segmentation for 600 MB image data size, whereas, if compared with the execution time of parallel image segmentation, the 600 MB image dataset just took 72 seconds to complete the job in parallel manner. From this comparison it could be clearly inferred that due to the rise in hyper-threading there is a wide gap generated between the sequential execution and the parallel execution.

The rise of hyper-threading within a single core ensures that the CPU time does not get wasted for other processes running on the node. It is worth to be noted that for sequential programming as the size of the image dataset starts getting higher, then there arises an impact on the Input-Output bound process too since the implementation of sequential image segmentation process is Input-Output bound. It is considered that hyper-threading on a single core can increase efficiency on heavy Input-Output bound processes. The following section will analyze the execution time for the higher category of image datasets and will also highlight the impact of heavy Input-Output bound process on these types of image datasets.

M. Image Datasets of 800 MB and 1 GB

Before coming up with a conclusive remark, let us have a look for the execution time of 800 MB and 1 GB image dataset. From the graph in Fig. 8, it could be observed that the execution time for 800 MB and 1 GB are 120 seconds and 136 seconds. A difference of 16 seconds could be clearly observed between these values. However, for parallel image segmentation the 800 MB and 1 GB image dataset took 87 seconds and 93 seconds to get implemented in parallel manner. Now from the sequential point of execution, it could be clearly inferred that threads do not make the computing machine run at a rapid rate. All they do is enhance the efficiency of the computing machine by utilizing the complete cores which would have been wasted otherwise on other processes. It is to be clearly noted that, with respect to hyper-threading when the number of threads increases within a single core then the degree of Input-Output bound also increases. For heavy Input-Output bound processes, the threads generated using hyper-threading must perform switching at several points. Therefore, if too many threads are to be run simultaneously, then the CPU spends most of its time in performing thread switching and not on the problem task. This process of overloaded thread context switching is called thrashing. Therefore, it could be concluded that for sequential processing the higher the size of image dataset, the more is the degree of thrashing.

N. CPU Cores Usage Analysis for Image Segmentation Using Sequential Programming

In this section, the analysis of the CPU cores usage for the implementation of image segmentation using sequential programming mode along with different segment of task execution time is done. The graph in Fig. 9 shows the maximum CPU cores usage attained for various size of image datasets executed using sequential programming mode.

From Fig. 9 it could be observed that, image segmentation implemented using sequential programming has a relatively stable CPU cores usage which averages around 14.3% over the entire execution. However, a theoretical CPU cores usage should be of 15.3%. The 1% difference is due to the Input-Output disk usage operation. It is also to be noted that the image segmentation implemented sequentially is totally cache bound. However, if the application wants to access the memory that is not in the cache then it might have to compete with the other memory access of numerous cores and in the mean time, if the application wants to write to the memory location, then there might arise a cache eviction(s) for other cores.

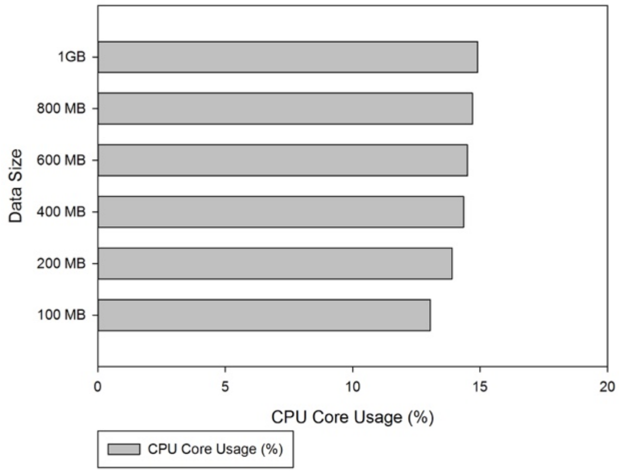

Fig. 9 Maximum CPU cores usage in sequential programming mode

The graph from Fig. 10 shows the distribution of CPU cores usage for the execution of various sizes of image datasets using sequential programming. It is worth to be noted that this process of implementing image segmentation sequentially is totally an Input-Output bound operation. Therefore, the graph in Fig. 10 also shows a spike like trend similarly to parallel image segmentation. The spike like trend arises in sequential implementation only when the degree of Input-Output bound process increases. It is also worth to be noted that if the users want to leverage on a lower end machines to carry out image processing tasks with lower size of image dataset, then the sequential computing is preferable over parallel computing.

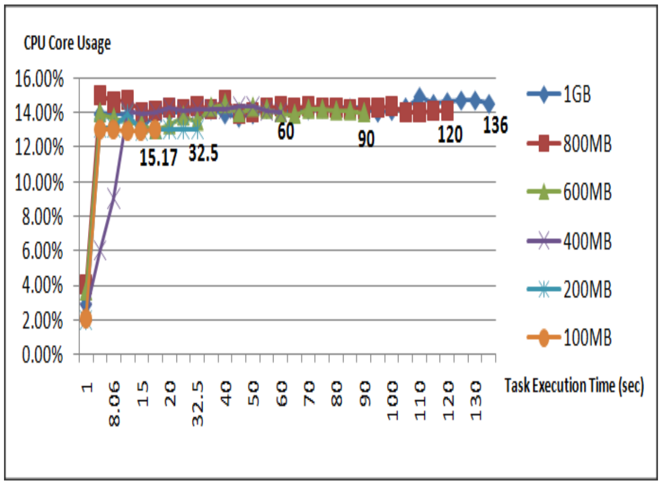

Fig. 10 Distribution of CPU cores usage for 100 MB-1 GB image dataset using sequential programming

O. Performance analysis based on CPU execution time on a single node

Fig. 11 shows the effect of data size on performance of job scheduling algorithms i.e., FIFO, Fair and Capacity schedulers and sequential execution of image segmentation on 1 GB of image data over a single node. The FIFO, Fair and Capacity schedulers are also compared with the sequential execution of image segmentation for the same 1 GB image data. From the graph it could be inferred that FIFO scheduler takes the lowest CPU execution time i.e., 96 seconds to complete the parallel image segmentation job while Capacity Scheduler takes the highest time i.e., 140 seconds to process the same amount of data in the parallel framework category. However, if we compare the parallel framework category with the sequential execution, then a high difference could be observed in the CPU execution time as the sequential execution takes 140 (136 sec + 4 sec) seconds to implement the image segmentation algorithm. It is to be noted that, the overhead of 4 second is due to the generation of log files.

Now from the sequential point of execution, it could be clearly inferred that threads do not make the computing machine run at a rapid rate. They only enhance the efficiency of the computing machine by utilizing the complete cores which would have been wasted otherwise on other processes.
It is to be clearly noted that, with respect to hyper-threading when the number of threads increases within a single core then the degree of Input-Output bound also increases. For heavy Input-Output bound processes, the threads generated using hyper-threading must perform switching at several points. Therefore, if too many threads are to be run simultaneously, then the CPU spends most of its time in performing thread switching and not on the problem task. This process of overloaded thread context switching is called thrashing. Therefore, it could be concluded that for sequential processing, higher the size of image dataset, the more is the degree of thrashing.

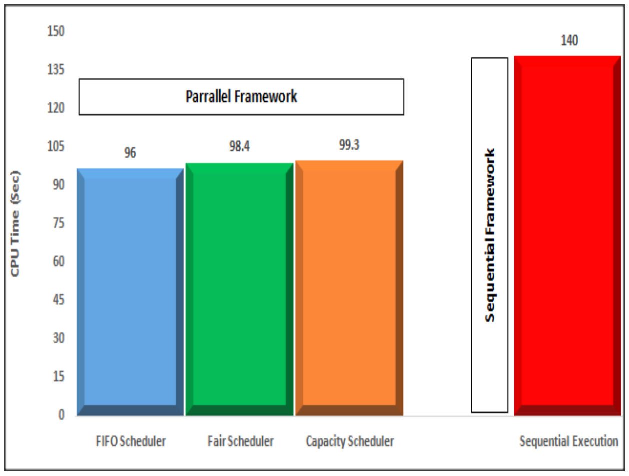

Fig. 11 Comparison of performance measure in terms of CPU time between parallel and sequential framework

P. Performance analysis based on data processing per second on a single node

The performance analysis based on data processing per second was conducted for 1 GB image dataset in the parallel frame category comprising of FIFO, Fair, and Capacity scheduler and sequential framework category. As shown in Fig. 12, the Fair scheduler and Capacity scheduler processed less bytes per second compared to FIFO scheduler. Moreover, if the parallel framework category is compared with the sequential execution, then it could be clearly inferred that FIFO scheduler in Hadoop processes data 1.45 times more than the sequential execution using OpenCV on visual studio platform and subsequently Fair scheduler processes data 1.42 times more than the sequential execution whereas Capacity scheduler process data 1.41 times more than sequential execution.

VI. CONCLUSION

In this research work, Hadoop framework has been used to implement the bulk image processing task in parallel. It is worth to be noted that there has been no other framework which has the potential to implement the task of bulk image processing in parallel. To assess the performance of Hadoop framework, the speedup in task execution time along with CPU cores usage at different segment of task execution time on the segmentation process of various sizes of image dataset ranging from 100 MB to 1 GB have been analyzed successfully. Post experimental analysis, it could be inferred that, with respect to task execution time, parallel image segmentation using Hadoop took 86 seconds to process 1 GB image data whereas with respect to sequential process of image segmentation, it took 136 seconds to process the same 1 GB image data.

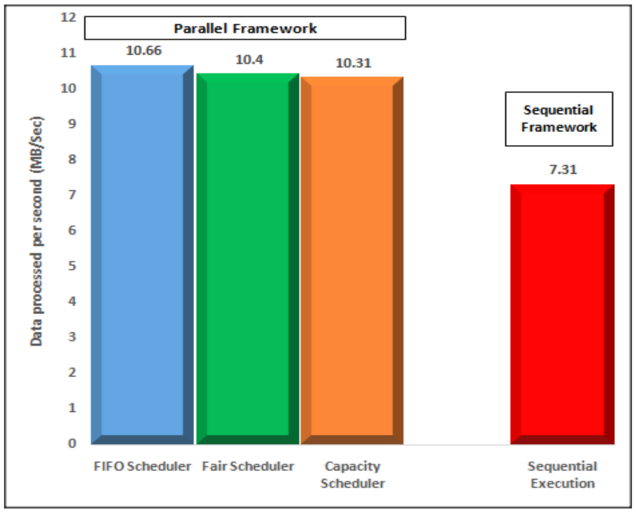

Fig. 12 Comparison of performance measure in terms of data processed per second between parallel and sequential framework

In terms of performance of scheduling the proposed parallel image segmentation gives best results with Map-Reduce based FIFO scheduler in terms of throughput and execution time if compared with Fair and Capacity scheduler. Therefore, it is evident from this analysis that parallel image segmentation could reduce the task execution time to process image data up to 1.6 times compared to sequential process of image segmentation. Nonetheless, the proposed parallel image segmentation algorithm could also be attempted to get implemented using parallel adaptive arbitration algorithm [25,26]. Overall this experiment ensures lower task execution time and utilization of maximum CPU cores up to 96% using task parallelism by keeping a balance between preemptive and non-preemptive process. The resulting speedups in task execution time along with maximum CPU cores usage demonstrate the potential of parallel computing for numerous images processing algorithm according to different stream of image data.

ACKNOWLEDGMENT

This research is supported by the School of Aerospace and the School of Electrical and Electronic Engineering of Universiti Sains Malaysia. The authors would like to acknowledge the Institute of Postgraduate Studies (IPS), Universiti Sains Malaysia for the Global Fellowship [USM.IPS/USMGF(06/14)] financial support to carry out this research.

REFERENCES

[1] Cappello, F. and Etiemble, D., "MPI versus OpenMP on the IBM SP for the NAS Benchmarks", *in Supercomputing, ACM/IEEE 2000 Conference*, 2000, pp. 80-92.

[2] Jost, G., Jin, H.-Q., Mey, D. and Hatay, F. F., "Comparing the OpenMP, MPI, and hybrid programming paradigm on an SMP cluster", *Research Article*, Computer Sciences Corp.; Moffett Field,CA, United States NASA Ames Research Center; Moffett Field, CA, United States. 2003, pp. 121-131.

[3] Slabaugh, G.; Boyes, R. and Yang, X.," Multicores image processing with OpenMP applications Corner", *IEEE Signal Processing Magazine* 27(2), 2010, pp. 134-138.

[4] Yang, C.-T. Huang, C.-L. and Lin, C.-F., "Hybrid CUDA, OpenMP, and MPI parallel programming on multicores GPU clusters", *Computer Physics Communications*, 182(1), 2011, pp. 266-269.

[5] Costantini, L. and Nicolussi, R., "Performances evaluation of a novel Hadoop and Spark based system of image retrieval for huge collections". *Advances in Multimedia*, 20(1), 2015, pp.11-16.

[6] Shuzhi Nie., "An improved parallel scalable K-means++ massive data clustering algorithm based on cloud computing", *International Journal of Circuits, Systems and Signal Processing*, Vol. 11, 2017, pp. 420-424.

[7] Wang, Y., Cao, S., Wang, G., Feng, Z., Zhang, C. and Guo, H., "Fairness scheduling with dynamic priority for multi workflow on heterogeneous systems", *in Cloud Computing and Big Data Analysis (ICCCBDA), IEEE*, 2017, pp. 404-409.

[8] Saxena, S., Sharma, S. and Sharma, N., "Parallel image processing techniques, benefits and limitations". *Research Journal of Applied Sciences, Engineering and Technology*, 12(2), 2016, pp.223-238.

[9] Ladkat, A.S., Date, A.A. and Inamdar, S.S., "Development and comparison of serial and parallel image processing algorithms", *in 'Inventive Computation Technologies (ICICT), IEEE International Conference'*, 2016, pp. 1-4.

[10] Jolliffe, I.T. and Cadima, J., "Principal component analysis: A review and recent developments", *Philosophical Transactions of the Royal Society A*, 374(2065), 2016, pp.201-502.

[11] Ieosanurak, W., Sakha, S., Klongdee, W., "Face classification based on PCA by using the center and foci of an ellipse", *International Journal of Circuits, Systems and Signal Processing*, Vol. 12, 2018, pp. 653-660.

[12] Demsar, U.," Principal component analysis on spatial data: An overview", *Annals of the Association of American Geographers*, 103(1), 2013, pp. 106-128.

[13] Jing Li., "Image feature matching based on improved SIFT algorithm", *International Journal of Circuits, Systems and Signal Processing*, Vol. 12, 2018, pp. 500-504.

[14] Firdousi, R. and Parveen, S., "Local Thresholding Techniques in Image Binarization". *International Journal Of Engineering And Computer Science*, 3(3), 2014, pp.4062-4065.

[15] Ali, M., Siarry, P. and Pant, M., "Multi-level Image Thresholding Based on Hybrid Differential Evolution Algorithm. Application on Medical Images", *Metaheuristics for Medicine and Biology*, 8(1), 2017, pp. 23-36.

[16] Otsu, N.," A threshold selection method from gray-level histograms", *Journal of Automatica*, 11(2), 1975, pp. 285-296.

[17] Sweeney, C., Liu, L., Arietta, S. and Lawrence, J.,, "HIPI: A Hadoop image processing interface for image-based map-reduce tasks", *Research Article, Chris. University of Virginia*. 2011, pp. 12-16.

[18] Hancock P.J.B., Baddeley R.J., Smith L.S., "The principal components of natural images Network", Vol. 3, 1992, pp. 61-72.

[19] Tremeau, A. and Borel, N. , "A region growing and merging algorithm to color segmentation", *Pattern recognition*, 30(7),1997, pp. 1191 -1203.

[20] Yin, P.Y. and Wu, T.H., "Multi-objective and multi-level image thresholding based on dominance and diversity criteria", *Applied Soft Computing*, 54(2), 2017, pp.62-73.

[21] Gedraite, E.S. and Hadad, M., "Investigation on the effect of a Gaussian Blur in image filtering and segmentation", in *IEEE ELMAR, Proceeding*, 2011, pp. 393-396.

[22] Gu, S., Zuo, W., Xie, Q., Meng, D., Feng, X. and Zhang, L., "Convolutional sparse coding for image super-resolution", in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1823-1831.

[23] Armato III, Samuel G., McLennan, Geoffrey, Bidaut, Luc, McNitt-Gray, Michael F., Meyer, Charles R., Reeves, Anthony P., Clarke, Laurence P. (2015). Data From LIDC-IDRI. The Cancer Imaging Archive. http://doi.org/10.7937/K9/TCIA.2015.LO9QL9SX

[24] M. Nishat Akhtar, Junita Mohamad Saleh, C. Grelck, "Parallel Processing of Image Segmentation Data Using Hadoop", *International Journal of Integrated Engineering*, 10(1), 2018, pp. 74-84.

[25] M. Nishat Akhtar, Junita Mohamad Saleh, O. Sidek, "Design and simulation of a parallel adaptive arbiter for maximum CPU utilization using multicore processors", *Journal of Computer and Electrical Engineering*, Vol. 47, 2015, pp. 51-68.

[26] M. Nishat Akhtar, Sidek O. "An Intelligent Adaptive Arbiter for Maximum CPU Utilization, Fair Bandwidth Allocation and Low Latency", *IETE Journal of Research*, 59(2), 2013, pp. 48-52.

Mohammad Nishat Akhtar received his B.E in Computer Science from VTU, Karnataka during the year 2010 , MS in Electrical and Electronics from Universiti Sains Malaysia during the year 2013 and PhD in Electrical and Electronics from Universiti Sains Malaysia during the year 2017. Currently He is a faculty member in School of Aerospace Engineering at Universiti Sains Malaysia and is teaching Advanced Engineering Calculus and Statics to first year engineering students. Prior to this, He has also taught Object Oriented Programming and Computer Organization to first year and third year engineering students. His research interests include High Performance Computation, Parallel Computing using OpenMP and MPI, Remote Sensing techniques, Image Processing, Multi-core Computing, Scheduling Algorithms, Embedded Systems and System-on-Chip.

Junita Mohamad-Saleh received her B.Sc (in Computer Engineering) degree from the Case Western Reserve University, USA in 1994, the M.Sc. degree from the University of Sheffield, UK in 1996 and the Ph.D. degree from the University of Leeds, UK in 2002. She is currently an Associate Professor in the School of Electrical & Electronic Engineering, Universiti Sains Malaysia. Her research interests include computational intelligence, tomographic imaging and parallel processing.

Elmi Abu Bakar received his Dip. Eng Mechanical, from Kisarazu, Japan, B. Eng Mechanical from Iwate, Japan, M. Eng Production System from Toyohashi, Japan and PhD. Eng Electronics and Information from Toyohashi, Japan. He is an Associate Professor in School of Aerospace Engineering at Universiti Sains Malaysia and is teaching Control Systems and Robotics to 3rd year students. He has several industrial collaborations and is actively working in the area of remote sensing for river monitoring. His research interests includes remote sensing, machine vision (image based measurement), control systems and robotics, abnormal detection using signal processing method, shape classification and analysis and tool and die quality inspection.

Ayub Ahmed Janvekar recieved hi B.E and M. Tech in Mechanical Engineering from VTU, Karnataka. Currently He is a PhD Scholar in School of Mechanical Engineering, Universiti Sains Malaysia. Prior to joining PhD, He was working as a Lecturer in Mechanical Engineering Department, at PESIT, Bangalore, India. His research interest include assessment of porous media burner, distress evaluation in thermal imaging and flame analysis.