College of Science
Department of Computer Science
COMP5557 High Performance Computing

# Project Progress Report

## Using Parallel Computing for Image Processing and Perform Fruit Recognition using MPI

**Prof. Khalid Day**

**Students Name:**

1- Nasser Saif Hamdan Al Jabri 83592

2- Aviral Goel 132206

# Abstract:

Nowadays, the categorization of fruits can be used in many sectors. It can be used in shopping to check the price. Also, it can be used in the agriculture sector to differentiate between good fruit and bad fruit or recognize different types of fruit. All these can be automated by using computer vision to let the computer help and accelerate the categorization. In this project, the system will use a set of fruit images as training to model and test it with a testing set of fruit images. First, the system will extract some features using colors, shapes, and textures from the training image and testing image then save it as tabular data. Then, the system will use these tabular data for the training phase and the testing phase. In the training phase and testing phase, different classification methods are used to choose the best one. And all these computations will take time if we do it in sequential. We'll work on parallelize the computation. First, we'll work on processing the training image and the testing image in parallel where we'll process more than one image at a time. After the image processing in the algorithms training and testing phase we'll test more than one algorithm at a time instead of testing one algorithm at a time. Our project is concerned with performing the high number of computation and model fitting in parallel. So, as to achieve faster training results and fit more than a few numbers of machine learning models. The project designs and implements an MPI based solution and compares the performance with different number of worker nodes.

## Introduction:

Machine Learning has been used in many sectors to develop our life. Many of the techniques used in machine learning are relying on computer vision. The Methods used in the various paper are Feature-based recognition, detection-based recognition, or deep-learning-based recognition [1].

In fruit recognition, the main basic features that are used to recognize the type of fruit are intensity, color, shape, and texture [2]. These basic features are used to extract other features like statistical features or morphological features.

Deciding which technique will be used to recognize the object will affect the type of features which are needed to be extracted.

In this project, we have a set of fruit images, and we need to build a classification system so that it can classify the type of fruit in the image. The classification system should extract the required feature from the image after pre-processing this image. The dataset which we have will be divided into two parts. One used for the training and the other will be used for testing our classifier.

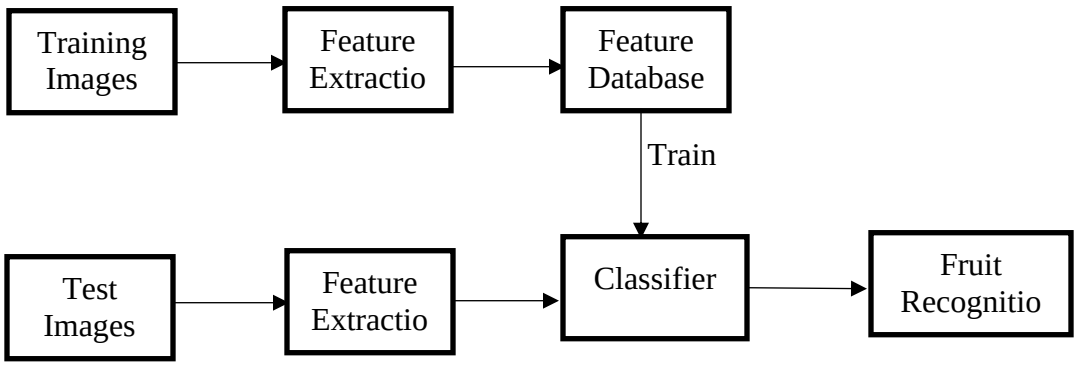
A dataset called fruit-360 is used as a training set and testing set. It was downloaded from Kaggle [3] on 11-May-2020. The configuration of this dataset as follows: It is divided into testing, training, and test various fruits. The training and testing have folders for 120 types of fruit. Moreover, the test multiple fruits contain a combination of different fruits with a diverse background.

We will rely on a fruit-360 data set, which has an image of 100 x 100 pixels, and the background is white. The work in this project is limited to the white background, not the complex background. Each image will contain only one type of fruit, and there is no mixture of fruit in one image.

The aim of this project to build a system that takes an image of a fruit and classify it based on its image. The system should extract the features from the image. The images in the dataset are saved as RGB. The system will use the training set to build the model of the classifier. Then, this classifier will be used to classify the test set.

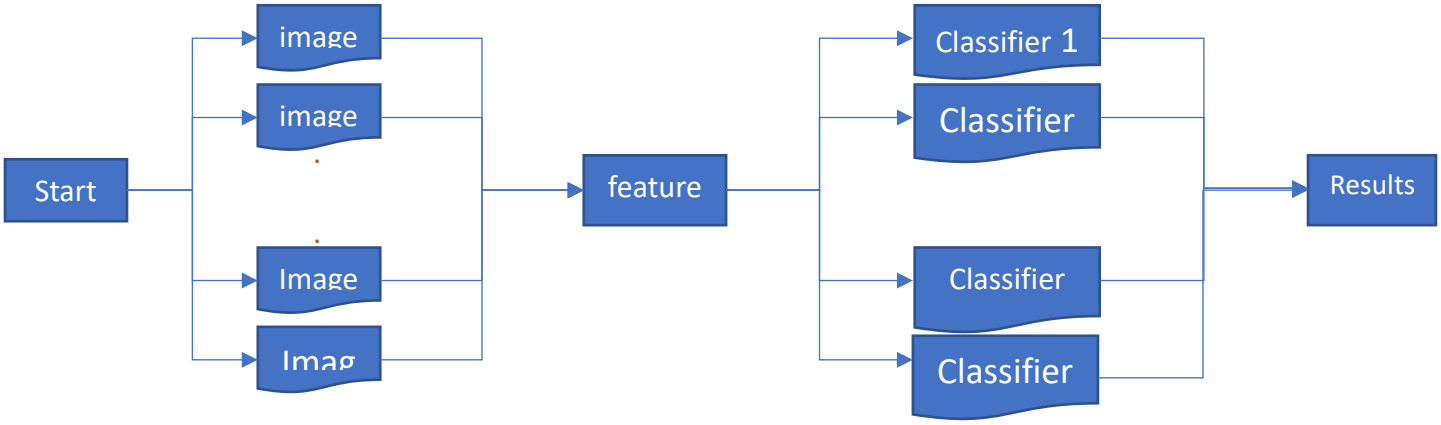
In object recognition using an image, a big data set to be used to train the machine. Then another data set to be used to test the trained model. The process to perform this a set of features to be extracted from each image. Then, these tabulated features will be used to train the model. After

that, this model will be used to test a test data set. The following diagram shows the steps of how to train the classifier using training images and use that classifier to recognize test images to check its performance.




Fruit Recognition System [2]

These are the steps of the experiment:




The features which will be extracted are color features, Morphological features and texture features. These are the equations which will be used to calculate the statistical in color features and morphological features [4] [5]:

Mean and this equation used for that: $$\mu_i = \frac{1}{MN}\sum_{j=1}^{M}\sum_{k=1}^{N} P_i(j,k)$$

Standard Deviation: $\sigma_i = \sqrt{\frac{1}{MN}\sum_{j=1}^{M}\sum_{k=1}^{N}(P_i(j,k)-\mu_i)^2}$

Skewness: $s_i = \sqrt[3]{(\frac{1}{MN}\sum_{j=1}^{M}\sum_{k=1}^{N}(P_i(j,k)-\mu_i)^3)}$

Kurtosis: $k_i = \frac{\mu_4}{\sigma_i^4}$

Where:

M = the number of pixels rows, N = the number of pixels columns, Pi = the pixel value at the i color channel, i = the color channel (Red, Green or Blue)

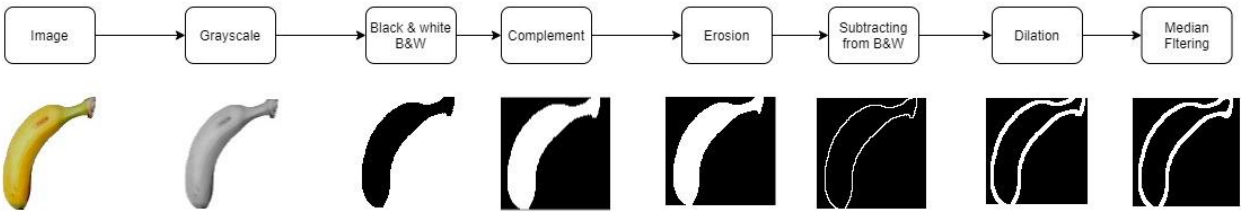
**Extracting color features CF:**

To extract the image color features, the image will be open as RBG, and we have three channels (Red, Blue, and Green) for each channel; the Statistical features will be calculated, so we have four statistical features for each channel. The total will be 12.

$$CF = [\mu_r, \mu_g, \mu_b, \sigma_r, \sigma_g, \sigma_b, s_r, s_g, s_b, k_r, k_g, k_b]$$

**Extracting Morphological Features MF:**

Below steps will be applied to get the shape and the boundary of the object:



Then the extracted boundary image will be used to calculate statistical features of it. So we have four features in total as MF: $MF = [\mu_{bi}, \sigma_{bi}, s_{bi}, k_{bi}]$

**Extracting texture features TF:**

The texture feature (TF) is calculated from the gray-level co-occurrence matrix (GLCM [6]). These equations below used to calculate the features in four directions 0º, 45º, 90º, 135º:

$$Ct = \sum_{j=0}^{M-1}\sum_{k=0}^{N-1} |j-k|^2 \, S_{jk}$$

$$Cn = \sum_{j=0}^{M-1}\sum_{k=0}^{N-1} \frac{(j-\mu_j)(k-\mu_k)S_{jk}}{\sigma_j \sigma_k}$$

$$Ey = \sum_{j=0}^{M-1}\sum_{k=0}^{N-1} S_{jk}^2$$

$$Hy = \sum_{j=0}^{M-1}\sum_{k=0}^{N-1} \frac{S_{jk}}{1+|j-k|}$$

Where [6]:

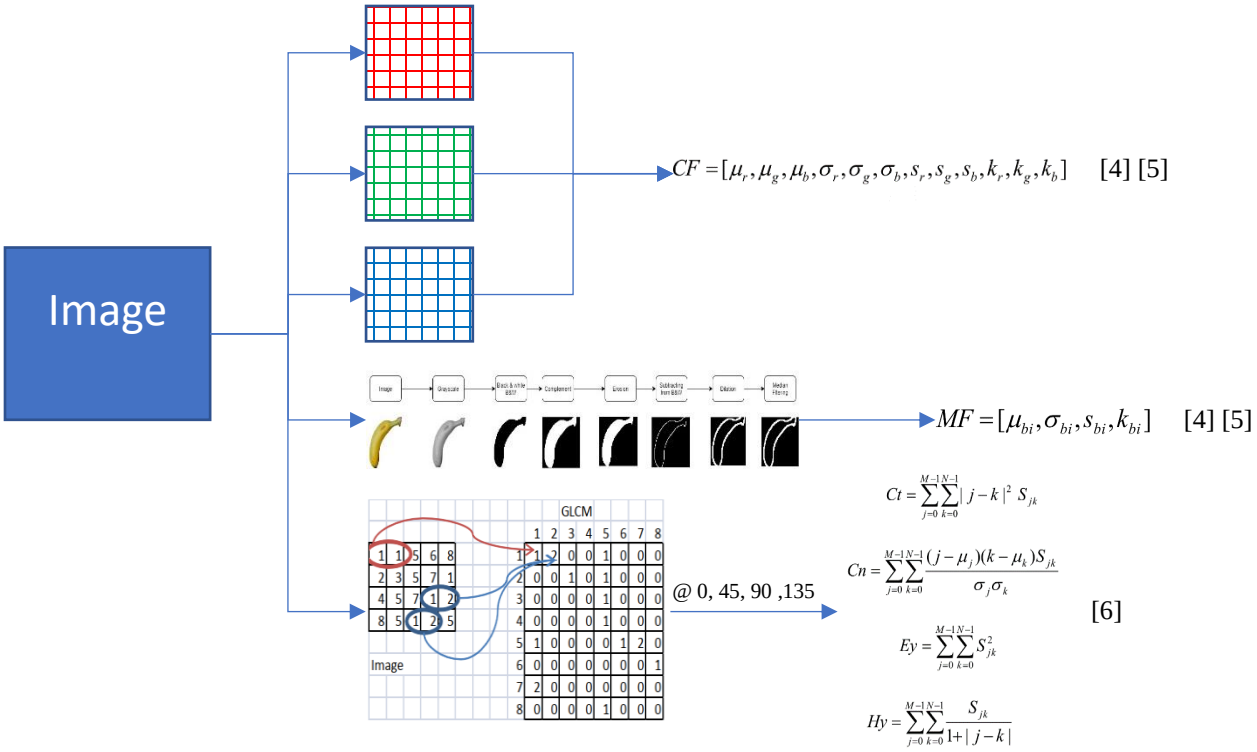
Ct = Contrast ,Cn = Correlation, Ey = energy, Hy = Homogeneity

All these values will be calculated in each direction. So, we have 16 texture features.

$$\begin{aligned} TF = [&Ct_{0^\circ}, Ct_{45^\circ}, Ct_{90^\circ}, Ct_{135^\circ}, \\ &Cn_{0^\circ}, Cn_{45^\circ}, Cn_{90^\circ}, Cn_{135^\circ}, \\ &Ey_{0^\circ}, Ey_{45^\circ}, Ey_{90^\circ}, Ey_{135^\circ}, \\ &Hy_{0^\circ}, Hy_{45^\circ}, Hy_{90^\circ}, Hy_{135^\circ}] \end{aligned}$$

In total 32 features will be extracted from each image.

The following figure is summarizing the features extraction from each image.




After Extracting these features, they will be saved as CSV files one for the training set and other for the testing set. Then, a different classifier was used with different parameters. These are the classifiers used in this study: KNN and SVM.

For this experiment the fruit360 dataset [3] of 120 types of fruits will be used. The total number of images 81104 consist of 60486 images as a training set and 20618 images as a testing set.

These images will be divided between processes to let each process extract the features from a part of the images, and at the end the extracted data will be combined together. So, the extraction can be done in parallel.

After finish the extraction, train and test the classifier can be done in parallel by letting each process work on one classifier.

**The Parallel Programming Tool:**

**mpi4py:** MPI for Python provides bindings of the Message Passing Interface (MPI) standard for the Python programming language, allowing any Python program to exploit multiple processors.

This package is constructed on top of the MPI-1/2/3 specifications and provides an object-oriented interface which resembles the MPI-2 C++ bindings. It supports point-to-point (sends, receives) and collective (broadcasts, scatters, gathers) communications of any picklable Python object, as well as optimized communications of Python object exposing the single-segment buffer interface (NumPy arrays, built-in bytes/string/array objects) (https://mpi4py.readthedocs.io)

**Luban High Performance Computation Facility – Sultan Qaboos University**

The authors acknowledge the Sultan Qaboos University HPC Luban supercomputing resources (http://squ.edu.om) made available for conducting the research reported in this report.

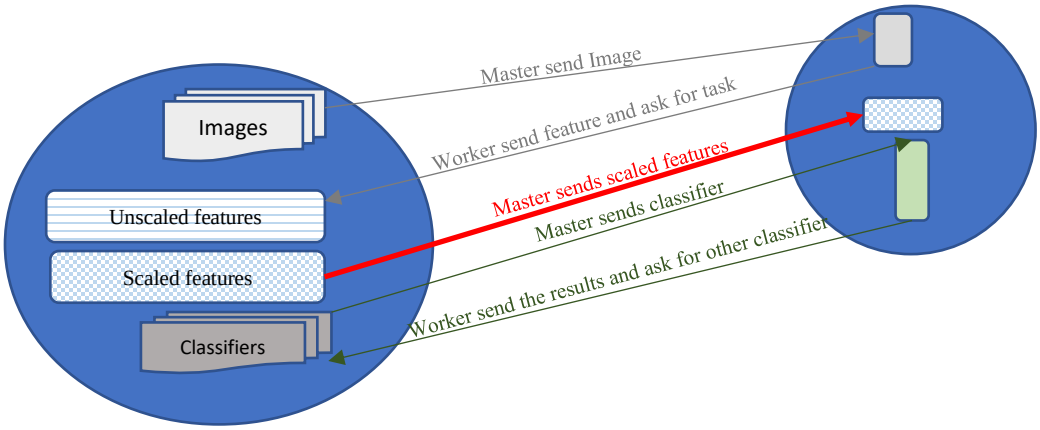
**Image Processing classification tools:**

**Scikit-learn:** Scikit-learn is probably the most useful library for machine learning in Python. The sklearn library contains a lot of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction. (https://www.analyticsvidhya.com/blog/2015/01/scikit-learn-python-machine-learning-tool/)

**Pandas:** is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the python programming language. (https://pandas.pydata.org/)

# Solution Design:

The performance will be enhanced using parallel computing because the image processing will be conducted for more than one image at a time. This design will focus on the parallel section of the solution. We will mention about image processing as a function and we'll consider the cost of processing one image is equal for all image as all images are 100 x 100 pixels and same processing function will be done for all images. In the pseudo code we'll use *Process_image(image)* which mean do all the processing function and extracting all features from the image.




Communication Design between Master and worker Processes

# Pseudo Code:

```
Init_MPI();
If(rank == 0){ //Master Process
        image_list[] = Init_image_list(); //get the image list
        feature_table[] = empty;
        int itr;
        for(itr = 1; itr <= num_proc; itr++){ //send image for each slave process
                Send(image_list[itr], req_tag, itr); //the tag is request tag
        }
        for(i = 0; i < image_list.length(); i++){//receive the data for each image sent
                Rcve(data, proc_num);
                Add_features(feature_table[], data); //add the received data to the feature table
                if(itr < Image_list.length()){//if there image in the list need to be processed
                        Send(image_list[itr], req_tag, proc_num);
                        itr++;
                }else{//if all images are processed
                        Send(empty, end_tag, proc_num);
                }
        }
```

```
        scaled_feature_table[] = Apply_Scaling(feature_table[]);//apply the required scaling for the
features
        for(int i = 0; i < proc_num; i++){//send the feature table to all processes
                send(scaled_feature_table[], i);
        }
        result_table[] = empty;
        classifier_list[] = get_classifier_list();//get the required classifier list
        for(itr = 1; itr <= num_proc; itr++){//send one classifier request to each process
                if(itr < classifier_list.length()){
                        Send(classifier_list[itr], req_tag, i);
                }else{
                        Send(empty, end_tag, i);
                }
        }

        for(i = 0; i < classifier_list.length(); i++){//receive the result from each process
                Rcve(result, proc_num);
                Add_result(result_table[], result);
                if(itr < classifier_list.length()){//if there is classifier to be applied
                        Send(classifier_list[itr], req_tag, proc_num);
                        itr++;
                }else{//if no classifier to be applied
                        Send(empty, end_tag, proc_num);
                }
        }
        Print(result_table[]);
}
Else{ //in slave processes
        Rcve(image, tag, 0); //receive a request to process an image
        While(tag != end_tag){ //loop until there is no image to process
```

```
            data = process_image(image); //process the image
            Send(data, 0); // send the features to the master
            Rcve(image, tag, 0);
        }
        Rcve(scaled_feature_table[], 0); //get the feature table to be used in the classifier
        Rcve(classifier, tag, 0); //receive  a request to apply a classifier
        While(tag != end_tag){ //loop until there is no request to use classifier
            result = apply_classifier(classifier,scaled_feature_table[]);
            Send(result, 0); //send the result to the master
            Rcve(classifier, tag, 0);
        }

}
```

# Implementation:

The Following code is used to divide the image processing between the worker

```
#send one image to each process
for i in range(num_proc-1):
    #Send using MPI. the tag = 10 for send image
    comm.send(image_list[i],dest=i+1,tag=10)
    itr += 1
```

The above loop will send one image to each process to be processed and send back to the master the resulted feature.

The following code in the master will keep receiving results from worker and send another image to be processed if there is still image to be processed. And the master will send a termination tag to all processes after receiving all the resulted features.

```
for i in range(num_image):
        #get the feature from the worker
        status=MPI.Status()
        features = comm.recv(tag=MPI.ANY_TAG,status=status)
        #construct the dataframe using the recieved features
        df_item = pd.DataFrame(features,columns = col)
        #check the tag. if it is 11 then this is training image. Else it is testing
        if(status.Get_tag() == 11):
                df_train = df_train.append(df_item,ignore_index = True,sort = False)
        else:
                df_test = df_test.append(df_item,ignore_index = True,sort = False)


        #print the progress of processing the images
        print(str(itr) + "/" + str(num_image) , end = "\r")
        #if there is image to be processed send it. Else send termination tag 13
        if(itr<num_image):
                comm.send(image_list[itr],dest=status.Get_source(),tag=10)
                itr += 1
        else:
                comm.send("empty",dest=status.Get_source(),tag=13)
```

In the worker side the following code is used to keep receiving an image and process it until get termination tag.

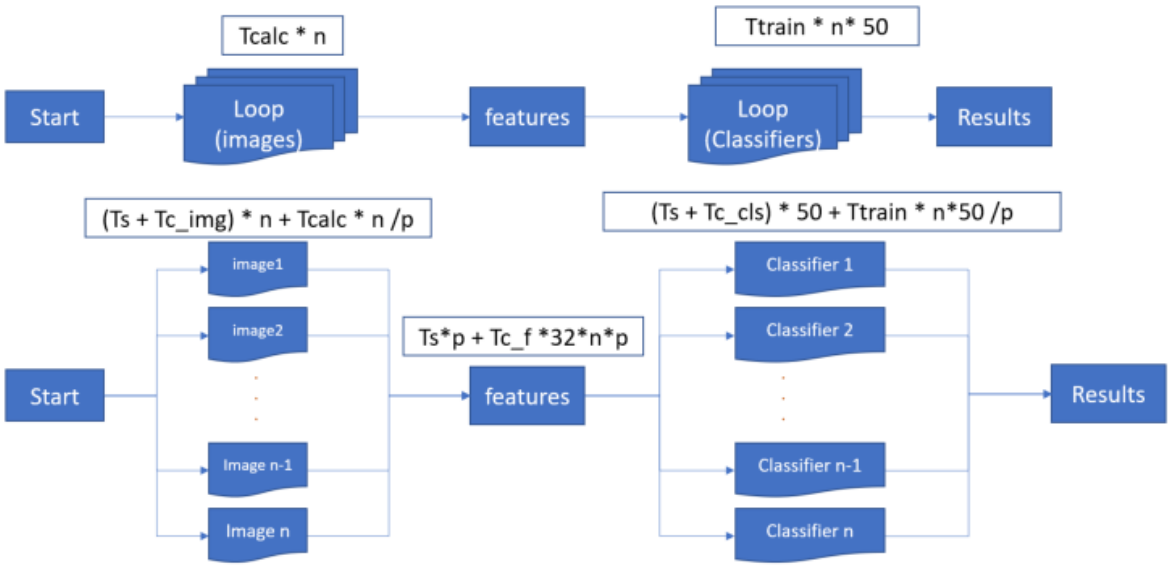
```
#recieve the request from the master
        status = MPI.Status()
        image_item = comm.recv(source=0, tag=MPI.ANY_TAG, status=status)
        #keep receiving tasks to process the images until get termination from the master
        while(status.Get_tag() == 10):
            #process the image
            features = image_feature2.get_image_dataframe(image_item[0],image_item[1])
            #check if it is training image or test to send tag based on that
            if(image_item[2] == 0):
                comm.send(features,dest=0,tag=11)
            else:
                comm.send(features,dest=0,tag=12)

            #recive from the master
            image_item = comm.recv(source=0, tag=MPI.ANY_TAG, status=status)
```

For the classifier work, same methodology is used to divide the work between the workers.

# Theoretical Analysis

The efficiency of the presented parallel approach can be theoretically determined by carefully analyzing the time complexity of the parallel algorithm as well as the sequential algorithm.

The following figure highlights the dominant communication and calculation intensive parts of both the sequential as well as parallel design.




### Sequential Time

Let: n=number of images, Tcalc = time needed to process one image, Ttrain = Time needed to train and test one classifier and we have 50 classifier

Step1: process all the images and get the features → Tcalc * n

Step2: train and test all the 50 classifiers using the resulted features from n images →Ttrain*n*50

Total time for sequential = Tcalc*n + Ttrain*n*50 = n* (Tcalc+Ttrain*50)

## Parallel Time

Let Ts=startup time to communicate, Tc_img=the time needed to send one image, p= number of processes, Tc_f = the time needed to send one feature, Tc_cls=the time needed to send one classifier

Step1: divide n images between the workers by sending one image each time and get the result. And each worker will work on around n/p images → (Ts + Tc_img) * n + Tcalc * n/p → n *(Ts + Tc_img + Tcalc /p)

Step2: the master will broadcast (or send one by one) the table of the scaled feature to all workers → Ts*p + Tc_f+32*n*p

Step3: divide 50 classifier between the workers and the worker will do the calculations → (Ts + Tc_cls) * 50 + Ttrain * n*50 /p

Total parallel time = n* (Ts + Tc_img + Tcalc /p) + Ts*p + Tc_f+32*n*p + (Ts + Tc_cls) * 50 + Ttrain * n*50 /p

→ n* (Ts + Tc_img + Tcalc /p + Tc_f+32*p + Ttrain *50 /p) + Ts*p + (Ts + Tc_cls) * 50

assume Ts =0

→ n* (Tc_img + Tcalc /p + Tc_f+32*p + Ttrain *50 /p) + Tc_cls* 50

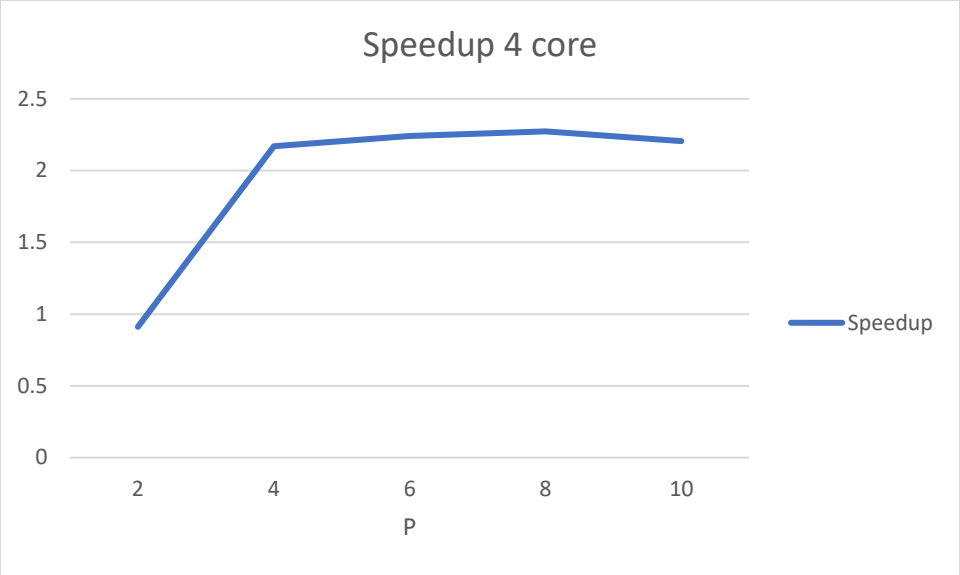
**Speedup =** $\frac{Sequential\ Time}{Parallel\ Time}$

**Efficiency = Speedup/p**

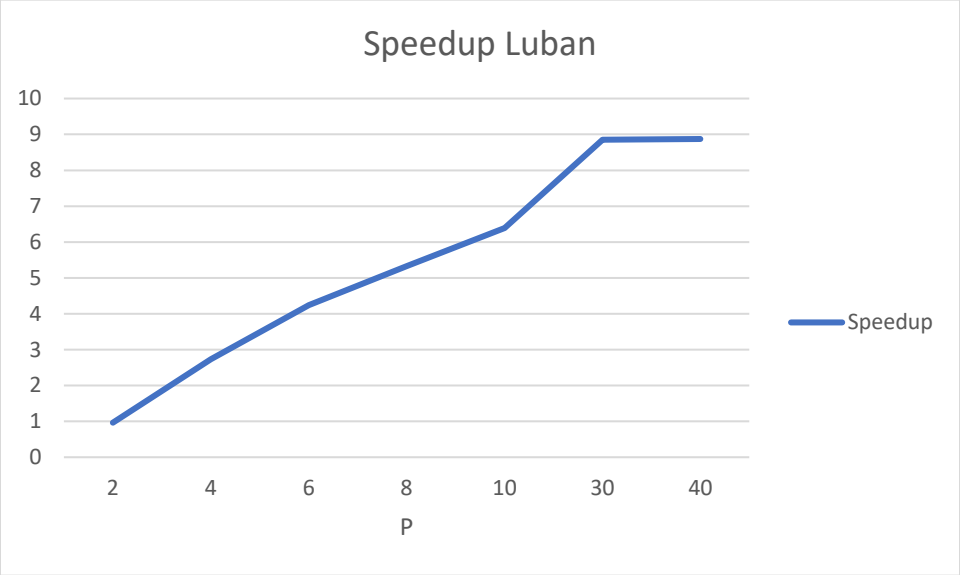
# Experimental Results

We have tested this experiment in two different machines. On with 4 core processor and the other is SQU HPC Luban.

The two graphs shows the speedup in the two machine. We notice that in 4 core machine the max speedup was when we use 4 process because when we use more than 4 processes it will not be a real parallel as there will be some processes competing with same core.




Where in Luban we notice that the speedup is keep increasing until it reach a limit where there will be no more speedup since that time is used for the communication for n images which is same for all number of processes since we will send the images one by one to the workers.



# Conclusion

Image classification is computation intensive task which can be parallelized with ease. The advantages of a parallel approach to image processing involves preprocessing large amount of data, training multiple models with varying permutations of parameters and getting quicker results. The performance of a multi process algorithm design can be first determined by theoretical analysis. In this project, the theoretical analysis concludes that the efficiency of the design is inversely proportional to twice the communication time between each worker process and master process. This result is backed up by our experimental data. We observe that for the largest dataset of 81K images (n=81K) the speedup is 2.27, indicating that a parallel approach works twice as fast. The efficiency is maximum for 4 processes (on a 4-core machine). The efficiency increases as the number of processes are increased, however it plateaus as the number of processes becomes equal to the number of cores in the system, indicating that processes are fighting for limited processing resource. Similar performance improvement was observed across all the dataset sizes.

# Future Scope

The project can further be improved by designing a better algorithm with lower communication overhead. The MPI based approach to parallel image recognition can also be extended to other machine learning tasks such as recommender systems, neural networks and deep learning. The presented algorithm may be further optimized and the communication overhead can be reduced. There is also room for trying out other parallel paradigms such as pthreads, CUDA etc as an alternative to MPI.

## References:

[1] M. San, M. M. Aung and P. P. Khaing, "Fruit Recognition Using Color and Morphological Features Fusion," MECS, 2019.

[2] R. S. ,. S. N. ,. L. S.Arivazhagan, "Fruit Recognition using Color and Texture Features," CIS Journal, India, 2009-2010.

[3] Fruits 360," [Online]. Available: https://www.kaggle.com/moltean/fruits. [Accessed 11 05 2020].

[4] M. San, M. M. Aung and P. P. Khaing, "Fruit Recognition Using Color and Morphological Features Fusion," MECS, 2019.

[5] D. Zwillinger and S. Kokoska, CRC Standard Probability and Statistics Tables and Formulae., New York: Chapman & Hall, 2000.

[6] D. S. S. A. Shruti Singh, "GLCM and Its Application in Pattern Recognition," 5th International Symposium on Computational and Business Intelligence.

# Appendix

```
# -*- coding: utf-8 -*-


import os

import pandas as pd

import image_feature2

from datetime import datetime

import time

from mpi4py import MPI

import pandas as pd

from sklearn import neighbors

from sklearn.naive_bayes import GaussianNB

from sklearn import svm

from sklearn.neural_network import MLPClassifier

from sklearn.model_selection import GridSearchCV

from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

from sklearn.preprocessing import StandardScaler


comm = MPI.COMM_WORLD

rank = comm.Get_rank()

num_proc = comm.Get_size()



if (rank==0): #in the master

        image_list = [] #List which contain all images info (file name, class num, train/test)

        log_file = open("mpi" + str(num_proc) + ".txt", "a")

        log_file.write("##########################################\n")

        log_file.write("#################New Test+++++++++++++++++\n")

        log_file.write("##########################################\n")

        df_train = pd.DataFrame() #store the training data

        df_test = pd.DataFrame() #store the testing data

        num_image=0


        ############Creating the list of images++++++++++++
```

```
start_time = time.time()
#loop to get the list of testing images info
for root, dirnames, filenames in os.walk("train/"):
        class_num = 0
        for class_name in dirnames:
                class_num+=1
                for rt, classname, files in os.walk(root+class_name):
                        for img_file in files:
                                image_item = [rt+"/"+img_file, class_num, 0]


                                image_list.append(image_item)
                                num_image+=1


#loop to get the list of testing images info
for root, dirnames, filenames in os.walk("test/"):
        class_num = 0
        for class_name in dirnames:
                class_num+=1
                for rt, classname, files in os.walk(root+class_name):
                        for img_file in files:
                                image_item = [rt+"/"+img_file, class_num, 1]


                                image_list.append(image_item)
                                num_image+=1


log_file.write("Total Number of images = " + str(num_image) + "\n")
log_file.write("Time to create a list of images is:\n")
log_file.write(str(time.time() - start_time) + "\n")
log_file.write("##################################################\n")
############Sending image and getting features from workers+++++++#####


start_time = time.time()#time stamp
itr = 0
```

```
#send one image to each process
for i in range(num_proc-1):
        #Send using MPI. the tag = 10 for send image
        comm.send(image_list[i],dest=i+1,tag=10)
        itr += 1

#loop to recieve the features from the workers and assign job to them
for i in range(num_image):
        #get the feature from the worker
        status=MPI.Status()
        features = comm.recv(tag=MPI.ANY_TAG,status=status)

        #construct the dataframe using the recieved features
        col = ["label","ur","ug","ub","stdr","stdg","stdb","skwr","skwg","skwb","kirtr","kirtg","kirtb",
"u_mf","std_mf","skw_mf","kirt_mf",
"Ct0","Ct45","Ct90","Ct135","Cn0","Cn45","Cn90","Cn135",
"Ey0","Ey45","Ey90","Ey135","Hy0","Hy45","Hy90","Hy135"]

        df_item = pd.DataFrame(features,columns = col)

        #check the tag. if it is 11 then this is trainging image. Else it testing
        if(status.Get_tag() == 11):
                df_train = df_train.append(df_item,ignore_index = True,sort = False)
        else:
                df_test = df_test.append(df_item,ignore_index = True,sort = False)

        #print the progress of processing the images
        print(str(itr) + "/" + str(num_image) , end = "\r")

        #if there is image to be processed send it to the worker. Else send termination tag 13
        if(itr<num_image):
                comm.send(image_list[itr],dest=status.Get_source(),tag=10)
                itr += 1
        else:
                comm.send("empty",dest=status.Get_source(),tag=13)
```

```
print(str(num_image) + "/" + str(num_image))
#print how much time it take to process the images
log_file.write("Time to process these images is:\n")
log_file.write(str(time.time() - start_time) + "\n")
log_file.write("#####################################################\n")
############Creating the Scalled features++++++++++++

start_time = time.time()
CF = [1,2,3,4,5,6,7,8,9,10,11,12]#Color Features
MF = [13,14,15,16]#Morphological Features
TF = [17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32]#Texture Features

trainX = df_train.iloc[:, CF+MF+TF].values # train features
trainY = df_train.iloc[:, 0:1].values.flatten() # train labels

testX = df_test.iloc[:, CF+MF+TF].values # test features
testY = df_test.iloc[:, 0:1].values.flatten() # test labels

scaler = StandardScaler()
scaler.fit(trainX)

testX_Z_score=scaler.transform(testX)#scale the test data
trainX_Z_score=scaler.transform(trainX)#scale the train data

scalled_feature = [trainY, testY, trainX_Z_score, testX_Z_score]

log_file.write("Time to create the scalled features is:\n")
log_file.write(str(time.time() - start_time) + "\n")
log_file.write("#####################################################\n")

############send the Scalled features++++++++++++

start_time = time.time()
for i in range(num_proc-1):
```

```
            #Send scalled features to all workers. the tag = 14 for send scalled features
            comm.send(scalled_feature,dest=i+1,tag=14)

#print how much time it take to send the scalled features
log_file.write("Time to send the scalled features to all workers is:\n")
log_file.write(str(time.time() - start_time) + "\n")
log_file.write("##################################################\n")
############Creating the list of Classifire+++++++#####

start_time = time.time()
classifier_list = []

#add KNN classifier to the list
knn = neighbors.KNeighborsClassifier(n_neighbors=1,algorithm='ball_tree')
classifier_list.append(knn)

#add GaussianNB classifier to the list
gau = GaussianNB()
classifier_list.append(gau)

#add SVM to the classifier list with different parameters
for C1 in [0.1,1, 10, 100]:
            for gamma1 in [1,0.1,0.01,0.001]:
                        for kernal1 in ['linear','rbf', 'sigmoid']:
                                    classifier_list.append(svm.SVC(C=C1, gamma=gamma1, kernel=kernal1))

log_file.write("Time to create the list of classifires is:\n")
log_file.write(str(time.time() - start_time) + "\n")
log_file.write("##################################################\n")
############Sending Classifires to the worker+++++++#####

start_time = time.time()

itr = 0
for i in range(num_proc-1):
```

```
                if(i<len(classifier_list)):# if there is classifier to be sent
                        classifier = [classifier_list[i], i]
                        comm.send(classifier,dest=i+1,tag=15)
                        itr += 1
                else:#if no classifier send termination to the rest of the process
                        comm.send("empty",dest=i+1, tag=17)
        class_file = open("mpi" + str(num_proc) + "class.txt", "a")
        status = MPI.Status()
        for i in range(len(classifier_list)):
                print(str(i) + "/" + str(len(classifier_list)), end = "\r")
                result = comm.recv(tag=16, status=status)

                class_file.write("#########################\n")
                class_file.write("get result from process no.: " + str(status.Get_source()) + "\n")
                class_file.write(str(classifier_list[result[0]]) + "\n")
                class_file.write("Accurecy --> " + str(result[1]) + "\n")

                if(itr<len(classifier_list)):#if still there is a classifire in the list
                        classifier = [classifier_list[itr], itr]
                        comm.send(classifier,dest=status.Get_source(),tag=15)

                        itr+=1
                else:#if no classifier send the termination
                        comm.send("empty",dest=status.Get_source(), tag=17)
        class_file.close()
        log_file.write("#################################################\n")
        log_file.write("Time to calculate the accurecy of classifires is:\n")
        log_file.write(str(time.time() - start_time) + "\n")

        log_file.write("\n")
        log_file.close()



####################Workers+++++++++++++++++####
```

```
else: #in the worker
        #recieve the request from the master
        status = MPI.Status()
        image_item = comm.recv(source=0, tag=MPI.ANY_TAG, status=status)

        #keep reciving tasks to process the images until get termination from the master tag=13
        while(status.Get_tag() == 10):
                #process the image
                features = image_feature2.get_image_dataframe(image_item[0],image_item[1])

                #check if it is training image or test to send tag based on that
                if(image_item[2] == 0):
                        comm.send(features,dest=0,tag=11)
                else:
                        comm.send(features,dest=0,tag=12)

                #recive from the master
                image_item = comm.recv(source=0, tag=MPI.ANY_TAG, status=status)

        scalled_feature = comm.recv(source=0, tag=14)
        trainY = scalled_feature[0]
        testY = scalled_feature[1]
        trainX_Z_score = scalled_feature[2]
        testX_Z_score = scalled_feature[3]
        #print(trainY)
        #print("worker no. " + str(rank) + " get scalled features")
        classifier = comm.recv(source=0, tag=MPI.ANY_TAG, status=status)
        #print("worker no. " + str(rank) + " get classifire --> " + str(classifier[1]))
        while(status.Get_tag() == 15):
                classifier[0].fit(trainX_Z_score,trainY)
                predY = classifier[0].predict(testX_Z_score)
                accuracy=accuracy_score(testY, predY)
                result = [classifier[1], accuracy]
                comm.send(result,dest=0,tag=16)
                #print("worker no. " + str(rank) + " get classifire --> " + str(classifier[1]))
```

```
                    classifier = comm.recv(source=0, tag=MPI.ANY_TAG, status=status)
```

```
# -*- coding: utf-8 -*-
import cv2
import scipy.stats as stats
import numpy as np
from skimage.feature import greycomatrix,greycoprops
import pandas as pd

def get_image_cf(image):
    #image = cv2.imread(image_file) #Read the image file

    red = image[:,:,0:1]#get Red Channel
    blue = image[:,:,1:2]#get blue channel
    green = image[:,:,2:3]#get the green channel

    red = np.reshape(red,(len(red)*len(red[1,:]),1))#Reshape the red channel
    blue = np.reshape(blue,(len(blue)*len(blue[1,:]),1))#reshape the blue channel
    green = np.reshape(green,(len(green)*len(green[1,:]),1))#reshape the grren channel

    #get the mean and standard deviation shape (1,3)
    (u, std) = cv2.meanStdDev(image)
    u = u[:,0]
    std = std[:,0]

    #calculate the skew for each channel
    skwr = stats.skew(red,axis = 0)
    skwb = stats.skew(blue,axis = 0)
    skwg = stats.skew(green,axis = 0)

    ##calculate the kitosis for each channel
    kirtr = stats.kurtosis(red)
    kirtb = stats.kurtosis(blue)
```

```
    kirtg = stats.kurtosis(green)

    skw = [skwr[0],skwb[0],skwg[0]]
    kirt = [kirtr[0],kirtb[0],kirtg[0]]

    return u,std,skw,kirt

############
def get_boundry_image(image_gray):
    #image = cv2.imread(image_file,0)#open the image file as gray

    #convert the image to black and while
    (thresh, bw_img) = cv2.threshold(image_gray, 200, 255, cv2.THRESH_BINARY)

    #get the complement of the image 1 to 0 and 0 to 1
    bw_img = abs(255-bw_img)

    #perform the erosion
    kernel = np.ones((3,3),np.uint8)
    eros_img = cv2.erode(bw_img,kernel,iterations = 1)

    #subtract the eroded image from the black and white image
    sbtr_img = bw_img-eros_img

    #perform Dilation
    dilat_img = cv2.dilate(sbtr_img,kernel,iterations = 1)

    #Apply median filter to remove the noise
    med_img = cv2.medianBlur(dilat_img,5)

    return med_img

############
def get_image_mf(image_gray):
    image = get_boundry_image(image_gray)
```

```
    (u, std) = cv2.meanStdDev(image)
    u = u[0,0]
    std = std[0,0]
    image = np.reshape(image,len(image)*len(image[1,:]))

    skw = stats.skew(image)
    kirt = stats.kurtosis(image)

    return u,std,skw,kirt

###########
def get_image_tf(image_gray):
    #image = cv2.imread(image_file,0)

    glcm_img = greycomatrix(image_gray, [1], [0, np.pi/4, np.pi/2, 3*np.pi/4],levels=256)

    Ct = greycoprops(glcm_img,'contrast')
    Cn = greycoprops(glcm_img,'correlation')
    Ey = greycoprops(glcm_img,'energy')
    Hy = greycoprops(glcm_img,'homogeneity')

    Ct = Ct[0,:]
    Cn = Cn[0,:]
    Ey = Ey[0,:]
    Hy = Hy[0,:]

    return Ct,Cn,Ey,Hy


def get_image_dataframe(image_file, label):
    image = cv2.imread(image_file) #Read the image file RGB
    image_gray = cv2.imread(image_file,0)#open the image file as gray

    (u,std,skw,kirt) = get_image_cf(image)
```

```
(u_mf,std_mf,skw_mf,kirt_mf) = get_image_mf(image_gray)
(Ct,Cn,Ey,Hy) = get_image_tf(image_gray)

features = [[label,u[0],u[1],u[2],std[0],std[1],std[2],skw[0],skw[1],skw[2],kirt[0],kirt[1],kirt[2],
            u_mf,std_mf,skw_mf,kirt_mf,
            Ct[0],Ct[1],Ct[2],Ct[3],Cn[0],Cn[1],Cn[2],Cn[3],
            Ey[0],Ey[1],Ey[2],Ey[3],Hy[0],Hy[1],Hy[2],Hy[3]]]
'''col = ["label","ur","ug","ub","stdr","stdg","stdb","skwr","skwg","skwb","kirtr","kirtg","kirtb",
          "u_mf","std_mf","skw_mf","kirt_mf",
          "Cto","Ct45","Ct90","Ct135","Cno","Cn45","Cn90","Cn135",
          "Eyo","Ey45","Ey90","Ey135","Hyo","Hy45","Hy90","Hy135"]

df = pd.DataFrame(features,columns = col)'''
return features
```