VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

Project Title

Facial Recognition

By

Saurabh Mishra-17BCI0033

…….

Abstract

We present the development of a novel high-performance face detection system using a neural network-based classification algorithm and an efficient parallelization with OpenMP. We discuss the design of the system in detail along with experimental assessment. Our parallelization strategy starts with one level of threads and moves to the exploitation of nested parallel regions in order to further improve, by up to 19%, the image-processing capability. The presented system is able to process images in real time (38 images/sec) by sustaining almost linear speedups on a system with a quad-core processor and a particular OpenMP runtime library.

Keywords

Facial Recognition, OpenMP, OpenCV ,Parallelization.

Literature survey

Convolutional neural network architecture model

Convolutional neural networks are standard multilayer perceptrons, which make extensive use of convolutions with adjustable masks and subsampling operations in order to extract appropriate and robust features at the first layers. The last layers of network contain standard sigmoid neuron layers, which perform the actual classification and produce the network output. This class of neural network is particularly interesting for face detection as the preprocessing and feature set extraction are built-in in the network and not handcrafted Convolutional neural networks are standard multilayer perceptrons, which make extensive use of convolutions with adjustable masks and subsampling operations in order to extract appropriate and robust features at the first layers. The last layers of network contain standard sigmoid neuron layers, which perform the actual classification and produce the network output. This class of neural network is particularly interesting for face detection as the preprocessing and feature set extraction are built-in in the network and not handcrafted.

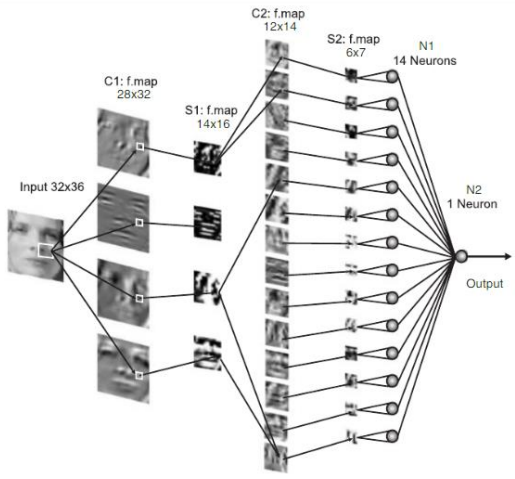


Figure 1. The convolutional neural network architecture.

Research Paper

No.	Paper Name	Remarks
1	ImageNet Classification with Deep Convolutional Neural Networks	Deep convolutional neural network to classify the 1.2 million high-resolution images in the ImageNet LSVRC-2010 contest into the 1000 different classes. The neural network, which has 60 million parameters and 650,000 neurons, consists of five convolutional layers, some of which are followed by max-pooling layers, and three fully-connected layers with a final 1000-way softmax.
2.	Parallel face Detection and Recognition on GPU	Proposed technique that process image for face detection and recognition in parallel on NVIDIA GeForce GTX 770 GPU which has compute Capability of 3.0. We have used Viola and Jones for face detection and PCA Eigenfaces algorithm for face recognition. The viola and jones shown result of processing the faces at 15 frame per second but our method shown approximately 109 frame per second in CUDA (Compute Unified Device Architecture) development framework. The algorithm are implemented to process data in parallel and made some practical optimization changes to work fast in C based programming model from NVIDIA .
3.	Facial recognition by a compact parallel optical correlator	The design procedure and fabrication process for an improved version of a second-generation compact parallel correlator (named COPaC II), the size of which is 20 × 24 × 43 cm3 and weight 6 kg. As a result, we obtained a low error rate of 0% as the false match rate and 0.3% as the false non-match rate, thus the COPaC II significant identification security level is sufficiently stable.
4.	Face recognition using line edge map	Faces can be recognized using a line edge map (LEM). The LEM, a compact face feature, is generated for face coding and recognition. A thorough investigation of the proposed concept is conducted which covers all aspects of human face recognition, i.e. face recognition under (1) controlled/ideal conditions and size variations, (2) varying lighting conditions, (3) varying facial expressions, and (4) varying pose. The system performance is also compared with the eigenface method, one of the best face recognition techniques, and with reported

		experimental results of other methods. A face pre-filtering technique is proposed to speed up the search process. It is a very encouraging to find that the proposed face recognition technique has performed better than the eigenface method in most of the comparison experiments.
5.	Parallel Processing for Multi Face Detection and Recognition	In this paper, a robust approach for real time face recognition where the images come from live video is proposed. To improve the algorithmic efficiency of face detection, we combine the eigenface method using Haar-like features to detect both of eyes and face, and Robert cross edge detector to locate the human face position. Robert Cross uses the integral image representation and simple rectangular features to eliminate the need of expensive calculation of multi-scale image pyramid. Moreover, In order to provide fast response in our system, we use Principal Component Analysis (PCA) to reduce the dimensionality of the training set, leaving only those features that are critical for face recognition. Eigendistance is used in face recognition to match the new face while it is projected on the face space. The matching is done when the variation difference between the new image and the stored image is below the threshold value. The experimental results demonstrate that the proposed scheme significantly improves the recognition performance. Overall, we find the system outperforms other techniques. Moreover, the proposed system can be used in different vision-based human computer interaction such as ATM, cell phone, intelligent buildings, etc.
6.	Real Time Application of Face Recognition Concept	Face Recognition concept is one of the successful and important applications of image analysis. It's a holistic approach towards the technology and have potential applications in various areas such as Biometrics, Information society, Law enforcement and Surveillance, Smart cards, Access control etc. This paper provides an overview of real time application of Face Recognition concept by generating a matlab code using image acquisition tool box. The basic approach used is Principal Component

		Analysis using Eigen faces, popularized by the seminal work of Turk and Pentland.
7.	Using Convolutional Neural Networks for Image Recognition	A neural network is a system of interconnected artificial “neurons” that exchange messages between each other. The connections have numeric weights that are tuned during the training process, so that a properly trained network will respond correctly when presented with an image or pattern to recognize. The network consists of multiple layers of feature-detecting “neurons”
8.	Parallel Implementation of Eigenfaces for Face Recognition on CUDA	One of the most successful template based techniques for face recognition is Principal Component Analysis (PCA) which is generally known as eigenface approach. It suffers from the disadvantage of higher computation cost, despite its better recognition rate. With the increase in number of images in training database and also the resolution of images, the computational cost also increases. Graphics Processing Unit (GPU) is the solution for fast and efficient computation. GPUs have massively parallel multi-threaded environment. With the use of GPU's parallel environment, a problem can be solved in parallel with much less time. NVIDIA has released a parallel programming framework CUDA (Compute Unified Device Architecture), which supports popular programming languages with CUDA extension for GPU programming. A parallel version of eigenface approach for face recognition is developed using CUDA framework.
9.	Optimized Parallel Implementation of Face Detection Based on Embedded Heterogeneous Many-Core Architecture	Computing performance is one of the key problems in embedded systems for high-resolution face detection applications. To improve the computing performance of embedded high-resolution face detection systems, a novel parallel implementation of embedded face detection system was established based on a low power CPU-Accelerator heterogeneous many-core architecture. First, a basic CPU version of face detection prototype was implemented based on the cascade classifier and Local Binary Patterns operator. Second, the prototype was extended to a specified embedded parallel computing platform that is called Parallella and consists of Xilinx Zynq

		and Adapteva Epiphany. Third, the face detection algorithm was optimized to adapt to the Parallella architecture to improve the detection speed and the utilization of computing resources. Finally, a face detection experiment was conducted to evaluate the computing performance of the proposal in this paper. The experimental results show that the proposed implementation obtained a very consistent accuracy as that of the dual-core ARM, and achieved 7.8 times speedup than that of the dual-core ARM. Experiment results prove that the proposed implementation has significant advantages on computing performance.
10.	A GPU based implementation of Robust Face Detection System	In our work, we have developed GPU based face detection system based on Viola Jones algorithm. To verify our work, we compared performance of our implementation with CPU based implementation at three stages of the algorithm i.e. image resizing, integral image calculation and cascade classification. We found that our GPU based implementation of Viola Jones face detection performed 5.41 to 19.75 times faster than its CPU implementation
11.	Very Low Resolution Face Recognition in Parallel Environment	This project tries to solve very low resolution (VLR) problem in face recognition where the resolution of the image to be recognized is lower than 16 × 16 pixels. With the increasing demand of surveillance camera-based applications, the VLR problem happens in many face application systems. Existing face recognition systems are not able to give satisfactory performance on the VLR face image. Based on this new approach, two constraints, namely, new data and discriminative constraints, are designed for good visual and face recognition applications under the VLR problem, respectively. Experimental results show that the proposed Feature extraction technique based on relationship learning and Euclidian distance outperforms the existing algorithms in face recognition from public face databases.
12.	Multi-algorithm-based face recognition system and method with optimal dataset partitioning for a cloud environment	A system and method of face recognition comprising multiple phases implemented in a parallel architecture. The first phase is a normalization phase whereby a captured image is normalized to the same size,

		orientation, and illumination of stored images in a preexisting database. The second phase is a feature extraction/distance matrix phase where a distance matrix is generated for the captured image. In a coarse recognition phase, the generated distance matrix is compared with distance matrices in the database using Euclidean distance matches to create candidate lists, and in a detailed recognition phase, multiple face recognition algorithms are applied to the candidate lists to produce a final result. The distance matrices in the normalized database may be broken into parallel lists for parallelization in the feature extraction/distance matrix phase, and the candidate lists may also be grouped according to a dissimilarity algorithm for parallel processing in the detailed recognition phase.
13.	A massively parallel face recognition system	We present methods for processing the LBPs (local binary patterns) with a massively parallel hardware, especially with CNN-UM (cellular nonlinear network-universal machine). In particular, we present a framework for implementing a massively parallel face recognition system, including a dedicated highly accurate algorithm suitable for various types of platforms (e.g., CNN-UM and digital FPGA). We study in detail a dedicated mixed-mode implementation of the algorithm and estimate its implementation cost in the view of its performance and accuracy restrictions.
14.	A distributed parallel system for face recognition	We present a distributed parallel system for face recognition. The face database in this system is so huge (more than 1,000 thousand faces) that matching and recognition processes cannot be carried out on only one computer. So cluster system must be used to improve the matching speed. But many current clusters existing have some problems and cannot fit our system. A special distributed parallel system was developed to complete face query and recognition. The concept of parallel virtual machine and a kind of linked table structure adopted in this system not only decreased lots of moving overhead of adding or deleting nodes in the array structure but also truly realized the infinite extensibility. Furthermore, key techniques such as distributed database,

		buffer and synchronization techniques in communication and multithreading in control flow were adopted to guarantee the normal running. Practical results proved that this parallel system could improve the matching speed for more than 4 times. Moreover, the greatest advantage of this system is not only increasing matching speed but also breaking the upper limit of face data capacity. Consequently, the face data capability of this system can be extended to arbitrary figure as bigger as possible.

Dataset description:

We have taken 323 images as our data set in which we have we have taken the photos of 19 different people. Each person has 18 different images of themselves. The images were taken in both dark and light so to get different types of images of the person. Then the image is converted into its pixel grayscale values and stored into notepad and that value is used to create a histogram.

Proposed system details and implementation:

At first all the images are converted into 2-Dimentional matrix and stored in a file for the use in the main program. This conversion of image into 2-D matrix is done using OpenCV library.

```
int main()
{
    int nrOfIds=20;
    int nrOfPhotosPerId=18;
    int num_rows=592;
    int num_cols=896;
    int histogramSize =256;
    double start_s=clock();
    int ***training_set=new int **[nrOfIds];
    for (int i = 0; i < nrOfIds; i++) {
        training_set[i] = alloc_2d_matrix(nrOfPhotosPerId,histogramSize);
    }
    for (int i = 0; i < nrOfIds; i++) { //initialize  training set to 0
        for (int j = 0; j < nrOfPhotosPerId; j++) {
            for (int e = 0; e < histogramSize; e++) {
                training_set[i][j][e] = 0;

            }
        }
    }
        string filename;
        for (int w = 1; w <= 20; w++) {
            for (int q = 1; q <= 18; q++) { //get all the file's names

                filename = "E:/PDC/LAB/Training/Person "+to_string(w) + "/image_0001 ("+ to_string(q)+").jpg";
                Mat person;
                person=imread(f,IMREAD_GRAYSCALE);
                int** data=alloc_2d_matrix(h,w);
                FILE *fe;
                std::string sn=f+".txt";
                fe=fopen(sn.c_str(),"w+");
                for (int y = 0; y < h; y++)
                {
                    for (int x = 0; x < w; x++)
                    {
                        int pixel = person.at<uchar>(x, y);
                        fprintf(fe,"%d\t",pixel);
                    }
                fprintf(fe,"\n");
                }
                fclose(fe);
                waitKey(0);
            }
        }
        double stop_s=clock();
        return 0;
    }
```

The Main program is divided into two parts (i) The training part (ii) The testing part. The paralysation is done using OpenMP.

(i) Training part:

- For each image in the dataset, the following steps are applied:
- For each pixel in an image, the pixel is compared with its 8 neighbours (on its top, bottom, left-top, left-middle, left-bottom, right-top, right-bottom, right-middle.).
- Where the centre pixel's value is less than the neighbour’s value, "1". Otherwise, "0".
- Value is converted into a decimal value.
- Histogram is computed, over the pixels, of the frequency of each "number" occurring (i.e., each combination of which pixels are smaller and which are greater than the centre).

```
void create_histogram(int *hist, int **img, int num_rows, int num_cols){
    int  smallMatrix[3][3];
    int i = 1;
    int decimal = 0;
    while ( i <= num_rows) {
        int j = 1;
        while ( j <= num_cols) {
            if (img[i][j] <= img[i - 1][j - 1]) {
                smallMatrix[0][0] = 0;

            }
            else{
                smallMatrix[0][0] = 1;

            }
            if (img[i][j] <= img[i - 1][j]) {
                smallMatrix[0][1] = 0;
            }
            else  {
                smallMatrix[0][1] = 1;

            }
            if (img[i][j] <= img[i - 1][j + 1]) {
                smallMatrix[0][2] = 0;
            }
            else {
                smallMatrix[0][2] = 1;
            }
            if (img[i][j] <= img[i][j - 1]) {
                smallMatrix[1][0] = 0;
            }
            else {
                smallMatrix[1][0] = 1;
            }
            if (img[i][j] <= img[i][j + 1]) {
                smallMatrix[1][2] = 0;
            }
            else  {
                smallMatrix[1][2] = 1;
            }
            if (img[i][j] <= img[i + 1][j - 1]) {
                smallMatrix[2][0] = 0;
            }
            else {
                smallMatrix[2][0] = 1;
```

```
            if (img[i][j] <= img[i + 1][j]) {
                smallMatrix[2][1] = 0;
            }
            else {
                smallMatrix[2][1] = 1;
            }
            if (img[i][j] <= img[i + 1][j + 1]) {
                smallMatrix[2][2] = 0;
            }
            else {
                smallMatrix[2][2] = 1;
            }
            decimal = smallMatrix[0][0] * int(pow(2, 7)) + smallMatrix[0][1] * int(pow(2, 6)) + smallMatrix[0][2] * int(pow(2, 5)) +
                      smallMatrix[1][2] * int(pow(2, 4)) +
                      smallMatrix[2][2] * int(pow(2, 3)) + smallMatrix[2][1] * int(pow(2, 2)) + smallMatrix[2][0] * int(pow(2, 1)) +
                      smallMatrix[1][0] * 1;

            hist[decimal]++;

            j++;
        }
        i++;
    }
```

(ii) Testing part:

- Histogram for the test image are generated as described in the previous part.
- Distance between test image's histogram and training histograms is computed.

```
double distance(int * a, int *b, int size) {
    double distance = 0;
  for (int i = 0; i < size; i ++) {
      if (a[i] + b[i] == 0) {
          distance += 0;
      }
      else {
          distance += 0.5 * pow ((a[i]- b[i]), 2) / (a[i] + b[i]);
      }
  }

    return  distance;
}
```

- Closest training histogram to determine the person is selected.

```
int find_closest(int ***training_set, int num_persons, int num_training, int size, int * test_image) {
    int i,j;
    double ** dist = new double * [num_persons]; //make an array which will store the comparison values
    for (i = 0; i < num_persons; i++) {
        dist[i] = new double [num_training];
    }

    #pragma omp parallel for private (j)
    for (i = 0;  i < num_persons; i++) { //populate the distance array
        for (j =0; j < num_training; j++) {
            dist[i][j] = distance(training_set[i][j], test_image, size);
        }
    }

    double closestValue = dist[0][0];
    int  closest = 0 ;

    #pragma omp parallel for private (j)
    for (i = 0;  i < num_persons; i++) {
        for (j =0; j < num_training; j++) {
            if (dist[i][j] < closestValue){
                closestValue = dist[i][j];
                closest = i;
        }
        }
    }

    #pragma omp parallel for
    for (int i = 0; i < num_persons; ++i) {
        delete dist[i];
    }
    delete []dist;
    return closest + 1;
}
```

- Results are displayed.

Evaluation results:

Parallel result:

```
**************************Welcome to the Facial Recognition**********************************
1.There are total of 323 images used for training.
2.There are 19 images is used for testing.One image for one person.
3.Every testing image will be compared with 323 image and the result will be told.
4.We will give two result one the predicated image and other actual result
5.Then we will calculate the accuracy of the program.
6.The program is parallelized and the time will be calculated and compared.
*********************************Software  used*****************************************************
1.Open cv.
2.Open mp.
The file is first converted into pixel using Opencv and then parallelized using Openmp.
================================================================================================

Started Scanning all the pixels.................
*************TAKING 18 IMAGE OF 1 PERSON FOR TESTING:*********************
Predicated Result           Actual Result
19                             1
*************TAKING 18 IMAGE OF 2 PERSON FOR TESTING:*********************
Predicated Result           Actual Result
2                              2
*************TAKING 18 IMAGE OF 3 PERSON FOR TESTING:*********************
Predicated Result           Actual Result
11                             3
*************TAKING 18 IMAGE OF 4 PERSON FOR TESTING:*********************
Predicated Result           Actual Result
4                              4
*************TAKING 18 IMAGE OF 5 PERSON FOR TESTING:*********************
Predicated Result           Actual Result
3                              5
*************TAKING 18 IMAGE OF 6 PERSON FOR TESTING:*********************
Predicated Result           Actual Result
7                              6
*************TAKING 18 IMAGE OF 7 PERSON FOR TESTING:*********************
Predicated Result           Actual Result
18                             7
*************TAKING 18 IMAGE OF 8 PERSON FOR TESTING:*********************
Predicated Result           Actual Result
15                             8
*************TAKING 18 IMAGE OF 9 PERSON FOR TESTING:*********************
Predicated Result           Actual Result
9                              9
*************TAKING 18 IMAGE OF 10 PERSON FOR TESTING:*********************
Predicated Result           Actual Result
5                              10
```

```
Predicated Result           Actual Result
1                              11
*************TAKING 18 IMAGE OF 12 PERSON FOR TESTING:*********************
Predicated Result           Actual Result
17                             12
*************TAKING 18 IMAGE OF 13 PERSON FOR TESTING:*********************
Predicated Result           Actual Result
13                             13
*************TAKING 18 IMAGE OF 14 PERSON FOR TESTING:*********************
Predicated Result           Actual Result
13                             14
*************TAKING 18 IMAGE OF 15 PERSON FOR TESTING:*********************
Predicated Result           Actual Result
15                             15
*************TAKING 18 IMAGE OF 16 PERSON FOR TESTING:*********************
Predicated Result           Actual Result
16                             16
*************TAKING 18 IMAGE OF 17 PERSON FOR TESTING:*********************
Predicated Result           Actual Result
17                             17
*************TAKING 18 IMAGE OF 18 PERSON FOR TESTING:*********************
Predicated Result           Actual Result
2                              18
*************TAKING 18 IMAGE OF 19 PERSON FOR TESTING:*********************
Predicated Result           Actual Result
10                             19
Accuracy: 7 correct answers for 19 tests
Parallel time: 127958 ms
```

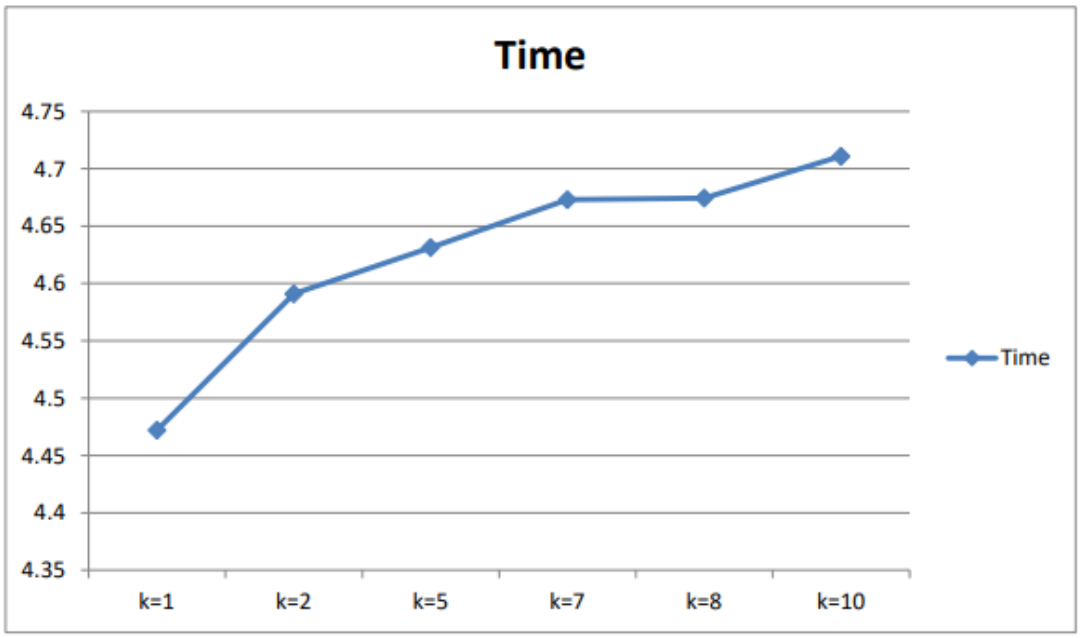
Serial results:

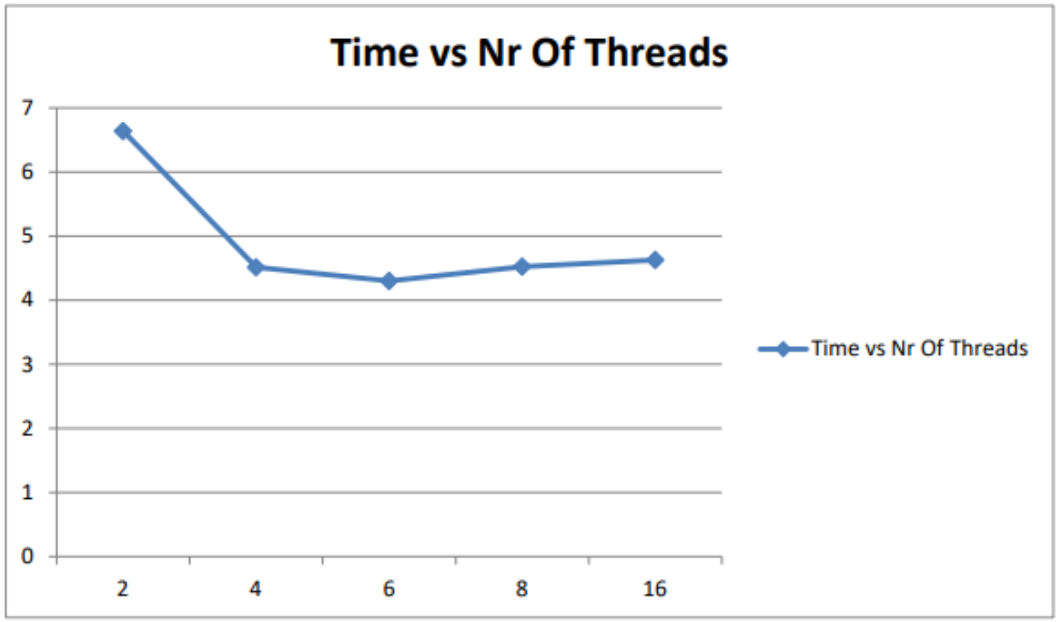
```
**************************Welcome to the Facial Recognition**********************************
1.There are total of 323 images used for training.
2.There are 19 images is used for testing.One image for one person.
3.Every testing image will be compared with 323 image and the result will be told.
4.We will give two result one the predicated image and other actual result
5.Then we will calculate the accuracy of the program.
6.The whole program will be parallelized and the time will be calculated and compared.
*********************************Software  used*****************************************************
1.Open cv.
2.Open mp.
The file is first converted into pixel using Opencv and then parallelized using Openmp.
================================================================================================

Started Scanning all the pixels.................
**************TAKING 18 IMAGE OF 1 PERSON FOR TESTING:**********************
Predicated Result           Actual Result
19                             1
**************TAKING 18 IMAGE OF 2 PERSON FOR TESTING:**********************
Predicated Result           Actual Result
2                              2
**************TAKING 18 IMAGE OF 3 PERSON FOR TESTING:**********************
Predicated Result           Actual Result
11                             3
**************TAKING 18 IMAGE OF 4 PERSON FOR TESTING:**********************
Predicated Result           Actual Result
4                              4
**************TAKING 18 IMAGE OF 5 PERSON FOR TESTING:**********************
Predicated Result           Actual Result
3                              5
**************TAKING 18 IMAGE OF 6 PERSON FOR TESTING:**********************
Predicated Result           Actual Result
7                              6
**************TAKING 18 IMAGE OF 7 PERSON FOR TESTING:**********************
Predicated Result           Actual Result
18                             7
**************TAKING 18 IMAGE OF 8 PERSON FOR TESTING:**********************
Predicated Result           Actual Result
15                             8
**************TAKING 18 IMAGE OF 9 PERSON FOR TESTING:**********************
Predicated Result           Actual Result
9                              9
**************TAKING 18 IMAGE OF 10 PERSON FOR TESTING:**********************
Predicated Result           Actual Result
5                              10
```

```
**************TAKING 18 IMAGE OF 11 PERSON FOR TESTING:**********************
Predicated Result           Actual Result
1                              11
**************TAKING 18 IMAGE OF 12 PERSON FOR TESTING:**********************
Predicated Result           Actual Result
17                             12
**************TAKING 18 IMAGE OF 13 PERSON FOR TESTING:**********************
Predicated Result           Actual Result
13                             13
**************TAKING 18 IMAGE OF 14 PERSON FOR TESTING:**********************
Predicated Result           Actual Result
13                             14
**************TAKING 18 IMAGE OF 15 PERSON FOR TESTING:**********************
Predicated Result           Actual Result
15                             15
**************TAKING 18 IMAGE OF 16 PERSON FOR TESTING:**********************
Predicated Result           Actual Result
16                             16
**************TAKING 18 IMAGE OF 17 PERSON FOR TESTING:**********************
Predicated Result           Actual Result
17                             17
**************TAKING 18 IMAGE OF 18 PERSON FOR TESTING:**********************
Predicated Result           Actual Result
2                              18
**************TAKING 18 IMAGE OF 19 PERSON FOR TESTING:**********************
Predicated Result           Actual Result
10                             19
Accuracy: 7 correct answers for 19 tests
167830
Sequential time: 167830 ms
```

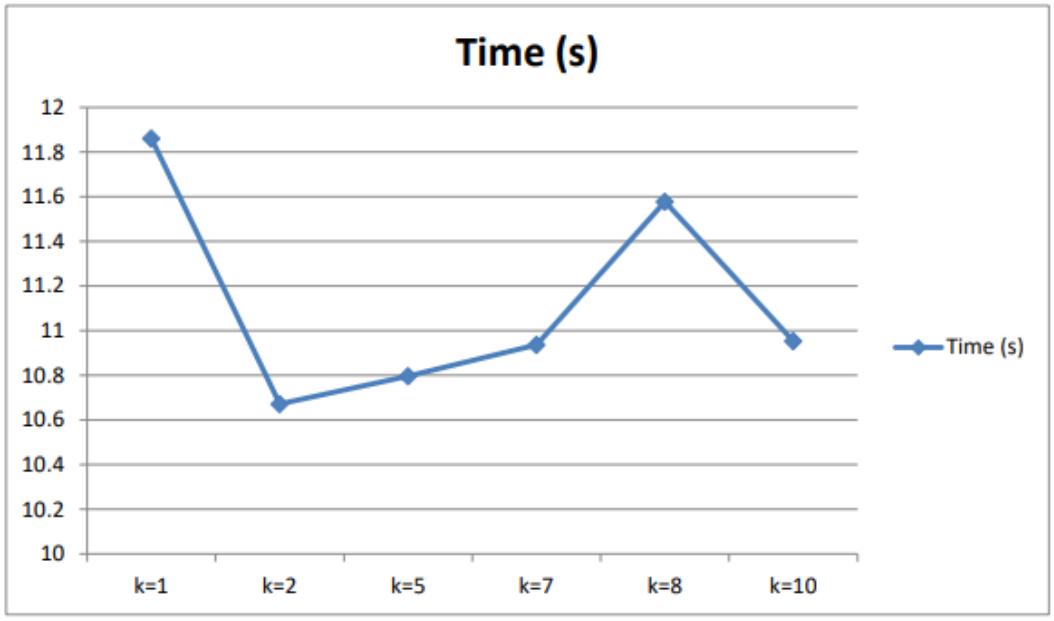
Graphs

Parallel Results





Serial Results



Explanation of Graph and Results

Serial :

As portrayed in the Errors graphs, at first for k = 1 there are 13 blunders and as k increases testing gives better outcomes (for k = 5 there are 3 blunders out of 270 tests). Moreover, for k >= 7 there are no blunders which shows that the preparation set is sufficient to decide precise results.

From the Time graph it very well may be found that the time variety for the sequential part is roughly 10-11 seconds with little fluctuation of 1-1.5 seconds (when k =1) . These contrast in results can be ascribed to correlation time (limited by number of tests) as it is diverse for various k, just as adjusting mistakes in the time work. Besides, the closeness in the outcomes can be credited to the time taken to figure the histograms, which requires dynamic allotment and correlations so as to get the decimals which are put inside the histograms (this procedure is the equivalent for all cases paying little mind to k).

Thinking about the consequences of the Graph, the accompanying can be said about the usage:

As expressed above from the diagram results, the time passed make histograms is the equivalent altogether cases. Solidly it requires some investment and is called 360 (18*20) to make the histograms. Total seconds comparing to it are 3.33. Calling primary which is done 1

time takes 4.06 combined seconds, in this manner 3.35 % of the time. The capacity for perusing from the documents into clusters is executed multiple times and takes 1.67 % of the absolute time which is 4.13 combined. Separation and power work whose calls shift for various nr of 4 have their claim part in the general time. Different capacities (deallocate, allocate, find nearest) don't take a huge amount of time, along these lines the issue is with the previously mentioned capacities. A good improvement would be in parallelising the piece of making histograms, contrasting, perusing the records and discovering separation between vectors.

Parallel:

As illustrated in the errors graph, the performance in terms of accuracy remains the same, the program runs correctly and the number of errors is the same as in the serial part. Regarding the Time graph, the time elapsed for different k's for the same nr of thread is observed. Concretely, results increasing slightly between 4.4-4.75 were obtained and as k increases the time elapsed increases as well. This can be attributed to a bigger number of training histograms each of which is to be compared with a test histogram, thus leading to a big number of computations. Logically being that less tests are done when k increases, the time should decrease, however the compare function does not take a considerable amount of time. Thus, this slight variation in time can be attributed to small errors in time calculating functions, the terminal and the pc. Regarding the graph Time vs Nr Of threads, these tests corresponds to k=1. Initially for 2 threads a bigger time is required for the program and as the number of threads increases, the time elapsed decreases (as shown in the graph). However, for a large nr of threads (i.e 14, 16) 0 1 2 3 4 5 6 7 2 4 6 8 16 Time vs Nr Of Threads Time vs Nr Of Threads a different behaviour was observed as the time elapsed increases slightly rather than being smaller than that for less nr of threads used. This can be due to rounding errors as well as the big number of threads is no longer required as the necessary optimisation is already done. To have a better insight on these results the following analysis of gproof and putting #pragma decisions is made: Being that the distance function is called in each comparison and it itself contains many calls of the power function which take a considerable time of time, I have inserted a parallelising statement before the for loop inside which the distance is computed for the histograms, thus making an improvement from the serial program. Furthermore, a pragma put for the for loop which initializes the 3D array where the histograms will be stored speeds up the initialization of the array elements with 0. Another important part is reading from files and creating the histograms, thus a pragma is inserted there to speed up the process of creating the histograms. Although allocation and deallocation of the 3D matrix do not require a significant amount of time, I have also parallelized that task. Regarding the proofs, I run for 2, 4,8,16 threads and obtained the corresponding outputs. Analysing the calls made to the functions, especially create_histogram and read_pgm_file, the results for different number of threads were varying in a not clear manner. Being that grove doesn't understand the intricacies of openmpi, these results can be attributed to this fact, as proof gives the results without the accuracy that they are being computed in omp. For instance, in the case of 4 threads each thread gets 90 pgm_files to open and populate the matrixes, however proof doesn't perceive well this behaviour, resulting to inaccurate results. However, by analysing the cumulative timing in proof txts for different threads, the time % taken by creating histograms reduces from more than 24% to less than 20% and reading the

files time also reduces slightly although its % is not significant. However, for thread Nr = 16 the proof timing results are slightly smaller than that of sequential. This can be also seen by the graphs where the timing decreases as the nr of threads increases except of when the nr of threads is much greater

Conclusion:

This project is a good example of paralysing a task with OpenMP. Reading, computing and comparing results with multiple data requires a high amount of time if the tasks are to be run serially. The serial version in this case requires about 1.5 times of the parallel version because in the serial version, the most time consuming tasks are reading from files and creating the histogram for each of 360 images, the runtime is reduced by inserting pragmas for the previously mentioned tasks. Regarding the time elapsed, the serial implementation takes about 160-170 seconds and the parallel one approximately 100-110 seconds. The improvement was obtained by looking at results for the serial part and optimizing it such that different number of threads take almost equal parts of the tasks and execute them in parallel, thus resulting in a better performance in terms of time.

References

(n.d.). Retrieved from https://ieeexplore.ieee.org/abstract/document/1301528.

(n.d.). Retrieved from https://ieeexplore.ieee.org/abstract/document/655647.

(n.d.). Retrieved from https://ieeexplore.ieee.org/abstract/document/6047293/.

Hirayama, T., Iwai, Y., & Yachida, M. (2002, November 21). Face Recognition Based on Efficient Facial Scale Estimation. Retrieved from https://link.springer.com/chapter/10.1007/3-540-36138-3_17.

Mckenna, S. J., Gong, S., & Raja, Y. (2001, June 7). MODELLING FACIAL COLOUR AND IDENTITY WITH GAUSSIAN MIXTURES. Retrieved from https://www.sciencedirect.com/science/article/abs/pii/S0031320398000661.

Yow, K. C., & Cipolla, R. (1998, May 19). Feature-based human face detection. Retrieved from https://www.sciencedirect.com/science/article/pii/S0262885697000036.

M.A. Turk and A.P. Pentland. “Face recognition using Eigenfaces”. In Proc. of Computer Vision and Pattern Recognition, pages 586-591. IEEE, June 1991b.

M.Turk and A. Pentland, "Eigenfaces for Recognition", Journal of Cognitive Neuroscience, March 1991.

L.I. Smith. “A tutorial on principal components analysis” Delac K., Grgic M., Grgic S., “Independent Comparative Study of PCA, ICA, and LDA on the FERET Data Set”, International Journal of Imaging Systems and Technology, Vol. 15, Issue 5, 2006, pp. 252-260.

H. Moon, P.J. Phillips, “Computational and Performance aspects of PCA-based Face Recognition Algorithms”, Perception, Vol. 30, 2001, pp. 303-321.