University of Campinas

Institute of Computing

MO644 – Introduction for Parallel Programming

Parallelization of Modified Census Transform applied to Face Recognition

RA:
192617

Student:
Edgar Rodolfo Quispe Condori

June 26, 2017

1 Modified Census Transform Histogram

In face Recognition and in general in image processing when using supervised learning the pipeline follows 3 big steps: image preprocessing, feature extraction and feature classification (figure 1). Since the most common algorithms for classification are already parallelized, this project focus on the step of feature extraction. Parallelization of the algorithm of Modified Census Transform Histogram (Modified Centrist) is done.

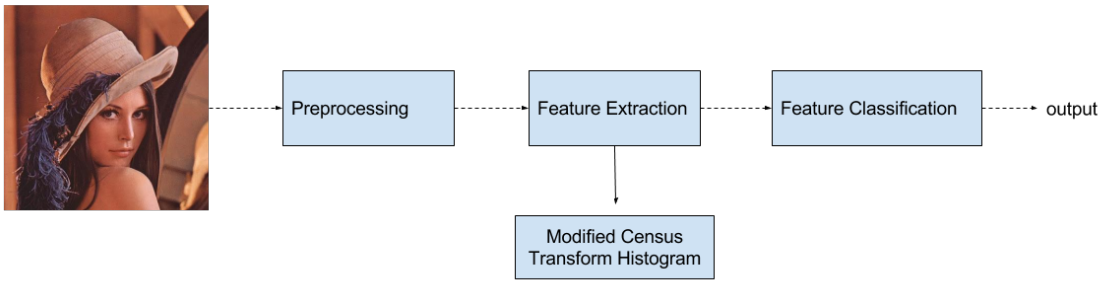

Figure 1: Classical pipeline for face recognition.

The modified Centrist is a descriptor that creates a histogram that represents the image texture (figure 2). It has a lot of application in problems like face detection, facial expressions analysis, hands posture recognition and general object detection.

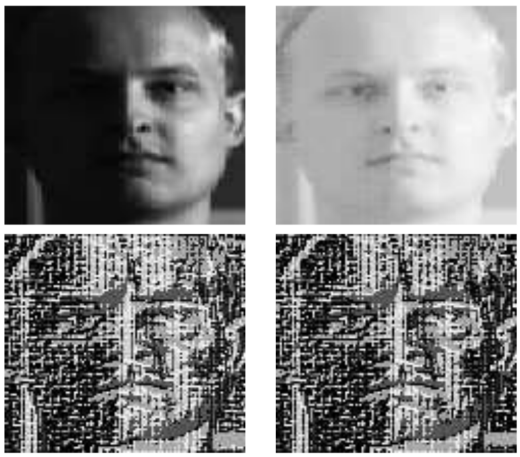

Figure 2: Modified Centrist applied to face detection.

1.1 Profiling

Doing a profile to the sequential implementation of the modified Centrist, it is perceived that the function $mod\_CENTRIST$ takes almost all the time of the algorithm. The input for this function is the original image and its ouput is the histogram of texture and Cencus Transform image.

```
Flat profile:
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
100.11      5.38     5.38        1     5.38     5.38  mod_CENTRIST
  0.00      5.38     0.00        2     0.00     0.00  readPPM
  0.00      5.38     0.00        2     0.00     0.00  rtclock

Call graph
granularity: each sample hit covers 2 byte(s) for 0.19% of 5.38 seconds
index % time    self  children    called     name
                5.38    0.00       1/1       main [2]
[1]    100.0    5.38    0.00       1         mod_CENTRIST [1]
-------------------------------------------------- <spontaneous>
[2]    100.0    0.00    5.38                 main [2]
                5.38    0.00       1/1       mod_CENTRIST [1]
                0.00    0.00       2/2       readPPM [3]
                0.00    0.00       2/2       rtclock [4]
-----------------------------------------------
                0.00    0.00       2/2       main [2]
[3]      0.0    0.00    0.00       2         readPPM [3]
-----------------------------------------------
                0.00    0.00       2/2       main [2]
[4]      0.0    0.00    0.00       2         rtclock [4]
-----------------------------------------------
```

This function is:

```
1  void mod_CENTRIST(PPMImage *image, PPMImage *image_copy, float *hist) {
2      int i, j, x, y;
3      //convert to grayscale
4      for(i = 0; i < image_copy -> y; i++)
5        for(j = 0; j < image_copy -> x; j++)
6          int grayscale = fromRGBtotGrayscale(i, j);
7          image_copy -> data[(i * image_copy->x) + j].red = grayscale;
8      for(i = 0; i < 512; i++) hist[i] = 0;
9      image->x-=2, image->y-=2;
10     //compute Census transform
11     for(i = 0; i < image -> y; i++)
12       for(j = 0; j < image -> x; j++){
13         float mean = 0.0;
14         for(y = i; y <= i + 2; y++) for(x = j; x <= j + 2; x++)
15           mean += image_copy -> data[(y * image_copy->x) + x].red;
16
17         mean /= 9.0;
18         int value = 0, k = 8;
19         for(y = i; y <= i+2; y++) for(x = j; x <= j+2; x++){
```

```
20            if(image_copy->data[(y*image_copy->x)+x].red >= mean)
21              value |= 1<<k;
22            k--;
23          }
24          image -> data[(i * image->x) + j].red = value;
25          image -> data[(i * image->x) + j].green = value;
26          image -> data[(i * image->x) + j].blue = value;
27          //compute histogram
28          hist[value]++;
29        }
30      for(i = 0; i < 512; i++) hist[i] /= (float)(image->x * image -> y);
31  }
```

2 Parallelization

Since the computation of grayscale (loop of lines 4, 5) and the Census Transform value (loop of lines 11, 12) of each pixel is independent, this loops can be parallelized assigning each pixel to a thread (figure 3). Note that everything thread will add a value to some position of the histogram, thus the access to this critical region must be controlled. Synchronization is important after the conversion to grayscale, because threads use this value to compute Census Transform. The loop of lines 8 and 30 can also be parallelize assigning a thread for each position of the array.

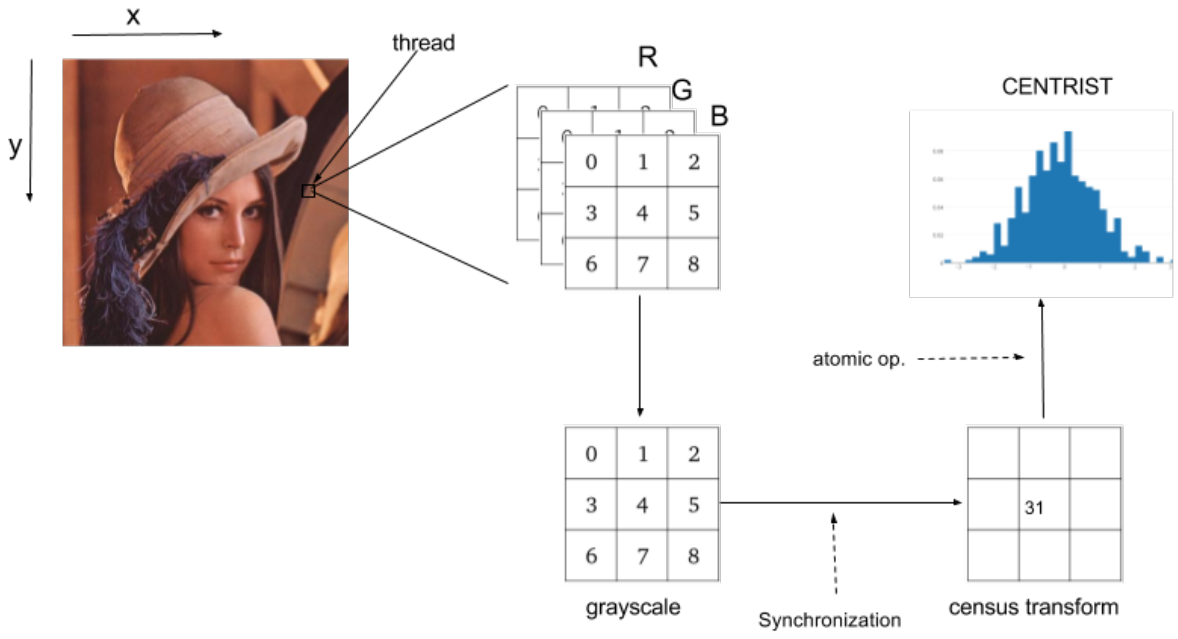

Figure 3: Parallelization of Modified Centrist.

The implementation in Cuda uses the shared memory, each block has a local histogram and also a copy of the original image. This is done in order to recover the values of the pixel in the neighborhood fastly and also reduce the number of threads that are waiting to write in the histogram. All this local histograms are joint to the final histogram before a synchronization of the Census Transform computation.

The implementation in Clang creates different kernels for the loops, this is done because we need to synchronize the loop that compute the grayscale and the Census Transform, and in the loop of lines 4, 5 only the variable *image_copy* is allocated in the GPU.

3 Results

The parallel implementations were tested in 5 images (table 1), all this images are in RGB and were previously transformed to *.ppm* format such they are manageable in *C* without any extra library.

Name	**Dimensions**
Image00	2048×1536
Image01	4360×2340
Image02	4014×3895
Image03	6000×4000
Image04	10000×5624

Table 1: Images used in the benchmark.

The results of the speed up obtained are shown in the figure 4. Cuda (blue bars), Clang with none optimization (red bars), Clang with tile optimization (pink bars), and Clang with vectorized optimization (gray bars) are compared.

The time of each implementation is obtained by the media of 5 executions. For the parallel implementations, the offload delay is also considered in the final time. The maximum speed up achieved with Cuda is 15.794962, with Clang using none optimization it is 5.296281, with Clang using tile optimization it is 5.197378 and with Clang using vectorized optimization it is 5.502497. If we compare the times of the Clang implementations, the optimizations used in Clang does not have a big impact in the final speed up.

The implementation in Cuda is faster than the Clang one because the construction of the histogram is sequential in Clang, this is because the there is not an *atomic* directive available that let synchronize the access to each position of the histogram. Besides this, there is an interesting speed up in the Clang implementation, this is because the conversion to grayscale and computation of Census Transform for each pixel were successfully parallelized.

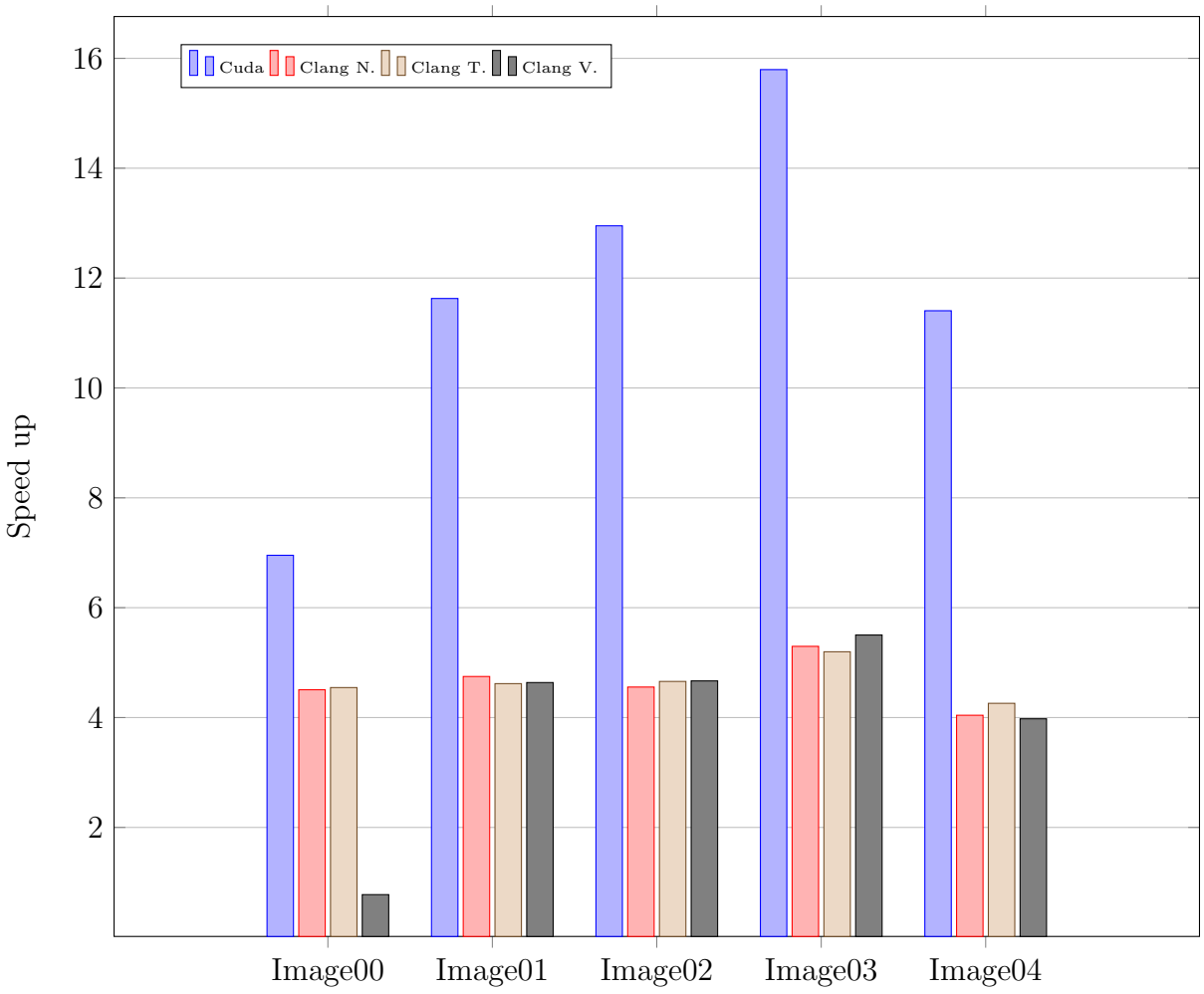

Figure 4: Speed up of the parallelizations.

The use of Modified Centrist let us get 95.2% of accuracy in the The ORL Database of Faces[1]

4 Difficulties

- There was a float point error generated in the GPU, the same operation returned different results in the CPU and GPU, this happened when the three RGB components had the same value.

```
int grayscale = (int)(red*0.299 + green*0.587 + blue*0.114);
cpu 252 252 252
cpu 252.000000000000000000000000000000000000000
cpu stored 252
gpu 252 252 252
```

[1] www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html

```
gpu 251.99999999999971578290569595992565155O3
gpu stored 251
```

- For the face recognition application algorithms like Support Vector Machines are needed, the library Sci-kit Learn have all this stuff available, but it is only available for python. We can use the *suprocces* library to use code implemented in C from python.

References

1. Wu, J., & Rehg, J. M. (2011). CENTRIST: A visual descriptor for scene categorization. IEEE transactions on pattern analysis and machine intelligence, 33(8), 1489-1501.

2. Marcel, S., Rodriguez, Y., & Heusch, G. (2007). On the recent use of local binary patterns for face authentication (No. LIDIAP-ARTICLE-2007-005).

3. Froba, B., & Ernst, A. (2004, May). Face detection with the modified census Transform. In Automatic Face and Gesture Recognition, 2004. Proceedings. Sixth IEEE International Conference on (pp. 91-96). IEEE.