

# Local Binary Pattern

---

Giacomo Ravara, Simone Casini

February 27, 2020

Università degli Studi di Firenze

[giacomo.ravara@stud.unifi.it](mailto:giacomo.ravara@stud.unifi.it)

[simone.casini3@stud.unifi.it](mailto:simone.casini3@stud.unifi.it)

# Overview

Sequential LBP

Parallel

CUDA

Results

## Sequential LBP

---

# LBP

- The Local Binary Pattern is an image descriptor
- Used mainly in texture analysis
- Computational simplicity

# LBP

- For each pixel define a neighborhood centered in the current pixel
- Compare pixels' values
- Define a new binary value for the current pixel

# Implementation

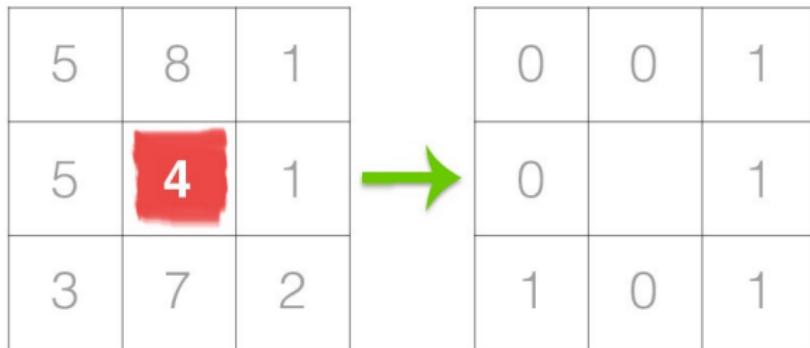
The LBP descriptor for every pixel is given as:

$$LBP(P, R) = \sum_{p=0}^{P-1} f(g_c - g_p)2^p$$

where  $g_c$  and  $g_p$  denote the intensity of the current and neighboring pixel, respectively. To the negative differences is assigned 0, to the positives is assigned 1.

The parameters of the algorithm are the number P of pixels to use for the LBP descriptor, R the radius of the neighborhood.

# LBP



Example of LBP step

## MultiBlock

- Different approach for facial image representation
- The image is divided into local regions
- Use local descriptions for each region and combine them into a global description

# Implementation

---

## Algorithm 1 LBP Sequential

---

```
1: for each pixel  $i$  in  $I$  do
2:   DEFINENEIGHBORHOOD( $i, P, R$ );
3:   COMPUTELBP( $i, P$ )
4:   UPDATEHISTOGRAM
```

---

# Parallel

---

- Use multiblock approach
- Divide between threads the input
- Each thread computes LBP on a part of input pixels
- No synchronization needed

# Implementation

---

## Algorithm 2 Parallel LBP

---

```
1: for  $t < num\_threads$  do
2:   LBPTHREAD ( $P, R$ )
3:   COMPUTEHISTOGRAM
```

---

---

## Algorithm 3 LBP Thread

---

```
1: for each pixel  $i$  do
2:   DEFINENEIGHBORHOOD( $i, P, R$ );
3:   COMPUTELBP( $i, P$ )
4:   UPDATEHISTOGRAM
```

---

# CUDA

---

- The multi-block approach exploits CUDA architecture
- Each block of threads corresponds to a sub-region of the image
- Each block computes LBP for its corresponding region

# CUDA

- Each thread computes LBP for a single pixel
- Then updates the local histogram
- **AtomicAdd** is used to avoid race condition

# Implementation

---

## Algorithm 4 Kernel LBP

---

```
1: for each thread in block do
2:   position ← threadId + blockId * blockEdge
3:   pixel ← GETPIXEL(position)
4:   vectorP ← LBPPVALUE(pixel)
5:   ATOMICADD(histogram, vectorP)
```

---

# Results

---

## Context

- The performances are measured by evaluating the speedup
- Changing the parameters and using different input sizes
- Using Microsoft Azure Virtual Machine, 2.60 GHz with 6 cores and NVIDIA Tesla M60

# Results

Size	Sequential	CUDA	SpeedUp
(750x562)	109 ms	5 ms	21.80
(1000x750)	193 ms	7 ms	24.13
(1500x1125)	441 ms	16 ms	27.56
(2000x1500)	774 ms	27 ms	28.67
(3000x2000)	1720 ms	57 ms	30.18
(4000x3000)	3234 ms	98 ms	33.45

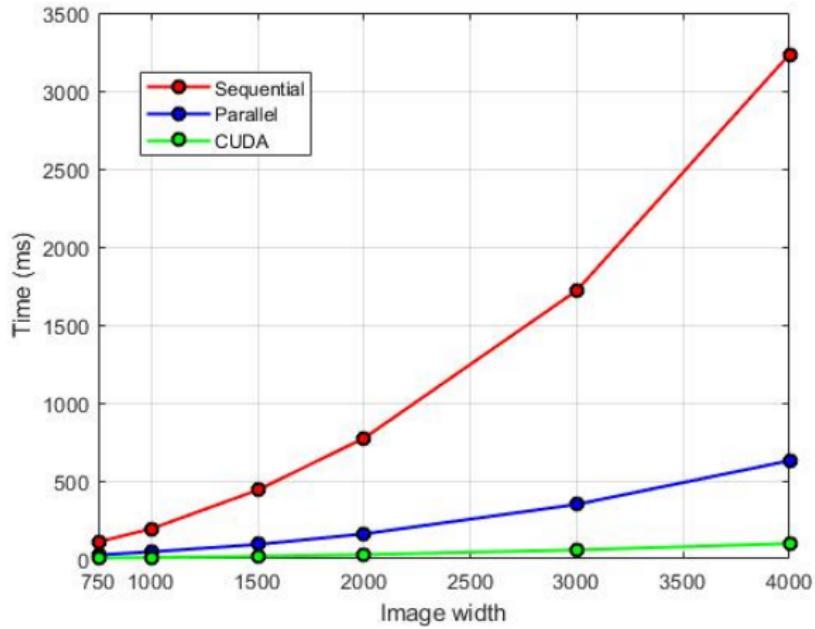
Tabelle 1: Comparison between sequential and CUDA

# Results

Size	Parallel	CUDA	SpeedUp
(750x562)	25 ms	5 ms	5.00
(1000x750)	45 ms	8 ms	5.62
(1500x1125)	92 ms	16 ms	5.75
(2000x1500)	160 ms	27 ms	5.93
(3000x2000)	349 ms	57 ms	6,13
(4000x3000)	629ms	98 ms	6,41

Tabelle 2: Comparison between parallel and CUDA.

# Results



Performance comparison with different image sizes

# Results

Parameters	Sequential	CUDA	SpeedUp
P=15, R=2, e=32	5069 ms	1004 ms	5.05
P=14, R=2, e=32	3885 ms	514 ms	7.56
P=13, R=2, e=32	3237 ms	264 ms	12.26
P=12, R=2, e=32	2866 ms	139 ms	20.62
P=11, R=2, e=32	2552 ms	76 ms	33.58
P=10, R=2, e=32	2299 ms	45 ms	51.09
P=9, R=2, e=32	2084 ms	27 ms	77.19
P=8, R=2, e=32	1926 ms	18 ms	107.00
P=6, R=2, e=32	1592 ms	10 ms	159.20
P=4, R=2, e=32	1297 ms	7 ms	185.29

Tabelle 3: Comparison between sequential and CUDA, size (3840x2160)

# Results

Parameters	Parallel	CUDA	SpeedUp
P=15, R=2, e=32	2468 ms	1004 ms	2.24
P=14, R=2, e=32	1442 ms	514 ms	2.81
P=13, R=2, e=32	907 ms	264 ms	3.44
P=12, R=2, e=32	646 ms	139 ms	4.65
P=11, R=2, e=32	502 ms	76 ms	6.61
P=10, R=2, e=32	408 ms	45 ms	9.07
P=9, R=2, e=32	349 ms	27 ms	12.93
P=8, R=2, e=32	318 ms	18 ms	17.67
P=6, R=2, e=32	261 ms	10 ms	26.10
P=4, R=2, e=32	213 ms	7 ms	30.43

Tabelle 4: Comparison between parallel and CUDA, image size (3840x2160)

# Results



Example of LBP Descriptor

END