Segundo parcial Multiprocesadores

Hector Daniel Durán Herrera
Ricardo Hernández Morales

Comparación de Sistemas Operativos

Equipos/Especificaciones		
SO	macOS Catalina/Ubuntu16.04	Ubuntu 20.04
RAM	8GB	16GB
Procesador	2.9GHz IntelCore i5 Dos Núcleos	3.2GHz Ryzen 7, 8-Core, 12 hilos fisicos

Procesamiento a realizar sobre cada imagen

- Convertir a escala de grises
- Efecto espejo
- Desenfoque

The pixels in an image are represented as integers. After blurring each pixel 'x' of the resulting image has a value equal to the average of the pixels surrounding 'x' including 'x'. For example, consider a 3 * 3 image as

$$\text{image} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 7 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Then, the resulting image after blur is blurred_image = $[1]$

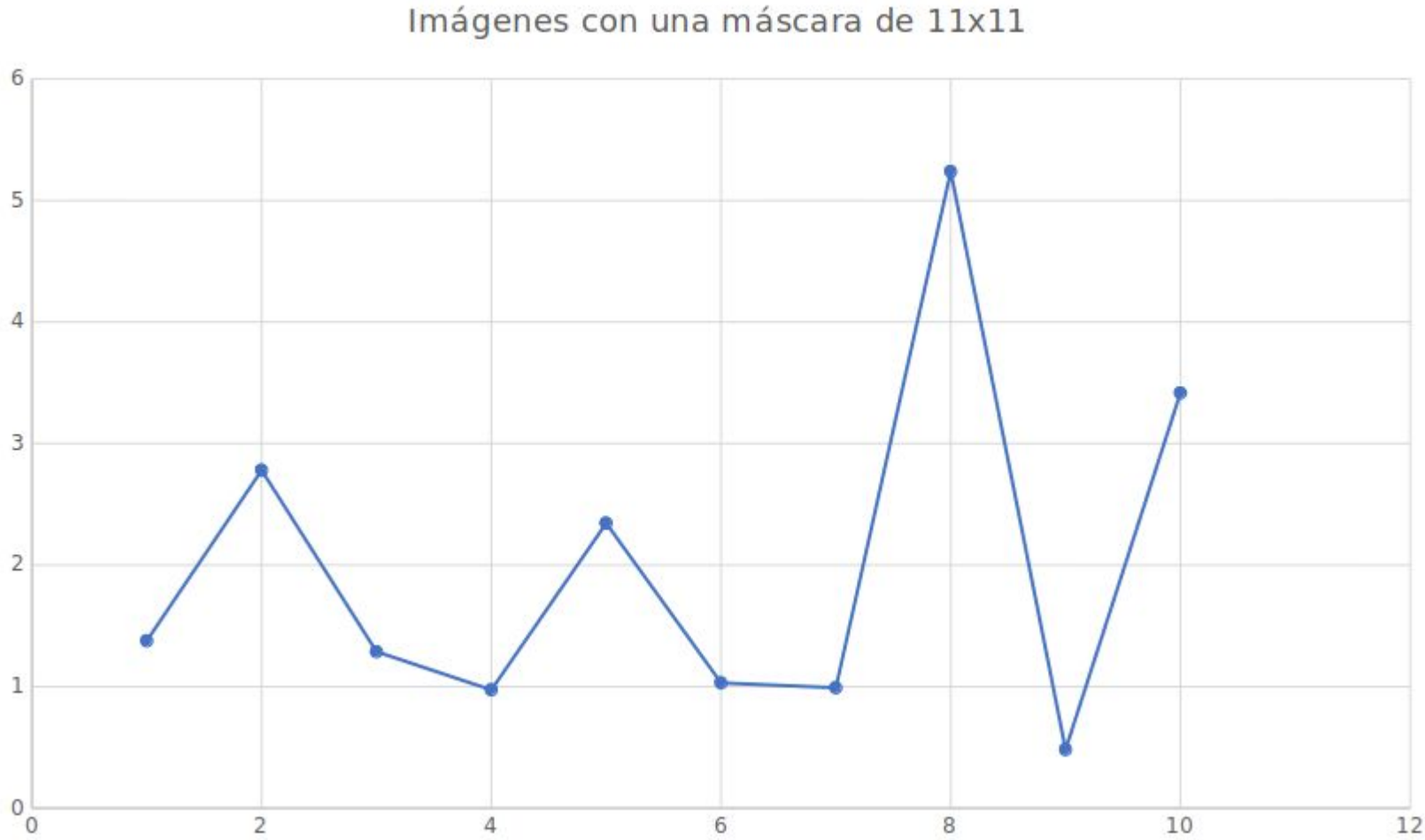
So, the pixel of blurred image is calculated as (1 + 1 + 1 + 1 + 7 + 1 + 1 + 1 + 1) / 9 = 1.66666 = 1

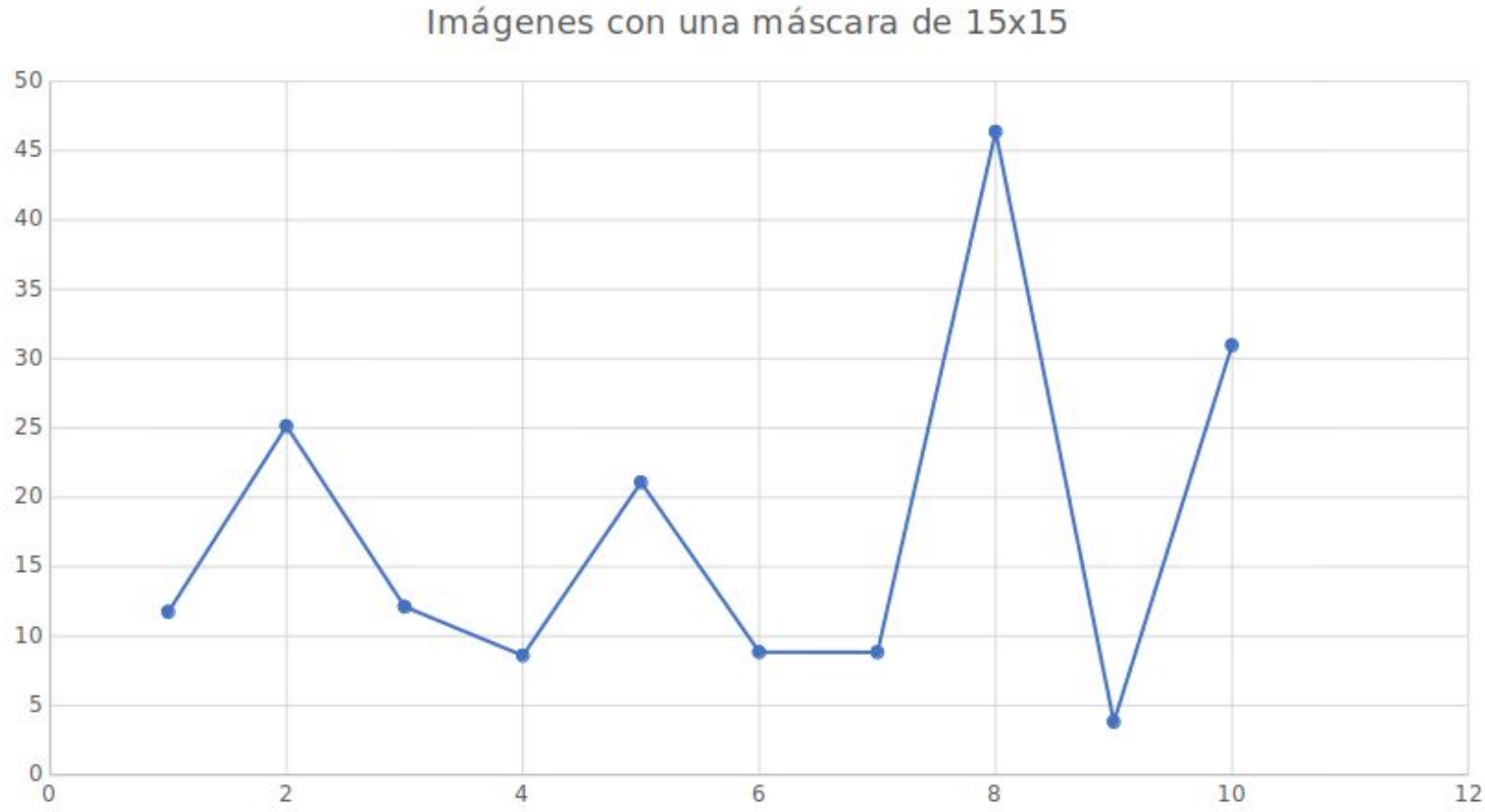
Schedule

```
  printf("FINISH GRAY CONVERT\n");
#pragma omp parallel for schedule(static)
  for (int i = 0; i < alto; i++)
  {
    for (int j = 0; j < (ancho * 3); j += 3)
    {
      *(arr_out + (i * ((ancho * 3) + n)) + j) = *(arr_in + (i * ((ancho * 3) + n)) + (ancho * 3) - j);
      *(arr_out + (i * ((ancho * 3) + n)) + j + 1) = *(arr_in + (i * ((ancho * 3) + n)) + (ancho * 3) - j - 1);
      *(arr_out + (i * ((ancho * 3) + n)) + j + 2) = *(arr_in + (i * ((ancho * 3) + n)) + (ancho * 3) - j - 2);
    }
  }
  aux = 0;
  int mask = 11;
  printf("FINISH SHIFT CONVERT\n");

#pragma omp parallel for schedule(guided)
  for (int i = 0; i < alto; i++)
  {
    for (int j = 0; j < (ancho * 3); j += 3)
    {
      aux = 0.0;
      for (int k = -(mask / 2); k <= mask / 2; k++)
      {
        for (int l = -(mask * 3 / 2); l <= mask * 3 / 2; l += 3)
        {
          aux += *(arr_out + ((i + k + 1) * ((ancho * 3) + n)) + j + l + 1) / (mask * mask);
        }
      }
      *(arr_blur + (i * ((ancho * 3) + n)) + j) = aux;
      *(arr_blur + (i * ((ancho * 3) + n)) + j + 1) = aux;
      *(arr_blur + (i * ((ancho * 3) + n)) + j + 2) = aux;
    }
  }

  printf("FINISH BLUR CONVERT\n");
  for (int i = 0; i < (ancho + n) * alto * 3; i++)
  {
```

Section

```
int altura_quad = alto / 4;
int altura_ultimo = altura_quad + alto % 4;
unsigned char *paux = arr_in;
unsigned char *q1 = malloc(altura_quad * (ancho + n) * 3 * sizeof(unsigned char));
unsigned char *q2 = malloc(altura_quad * (ancho + n) * 3 * sizeof(unsigned char));
unsigned char *q3 = malloc(altura_quad * (ancho + n) * 3 * sizeof(unsigned char));
unsigned char *q4 = malloc(altura_ultimo * (ancho + n) * 3 * sizeof(unsigned char));

memcpy(q1, paux, altura_quad * (ancho + n) * 3 * sizeof(unsigned char));
paux += altura_quad * (ancho + n) * 3;
memcpy(q2, paux, altura_quad * (ancho + n) * 3 * sizeof(unsigned char));
paux += altura_quad * (ancho + n) * 3;
memcpy(q3, paux, altura_quad * (ancho + n) * 3 * sizeof(unsigned char));
paux += altura_ultimo * (ancho + n) * 3;
memcpy(q4, paux, altura_quad * (ancho + n) * 3 * sizeof(unsigned char));

//printf("BEFORE FUNC\n");

#pragma omp parallel
{
    #pragma omp sections
    {
        #pragma omp section
        {
            unsigned char *grayRes;
            unsigned char *shiftRes;
            unsigned char *blurRes;

            grayRes = gray(q1, ancho, altura_quad, n);
            shiftRes = shift(grayRes, ancho, altura_quad, n);
            blurRes = blur(shiftRes, ancho, altura_quad, n, MASK);
            //blurRes = shiftRes;
            memset(q1, '\0', altura_quad * (ancho + n) * 3);

            memcpy(q1, blurRes, altura_quad * (ancho + n) * 3);
```


Imágenes con una máscara de 11x11
0
1
2
3
4
5
6
0
2
4
6
8
10
12


Imágenes con una máscara de 15x15
50
45
40
35
30
25
20
15
10
5
0
0
2
4
6
8
10
12

Resultados