CUDA Acceleration for Edge Detection

ECE 5720

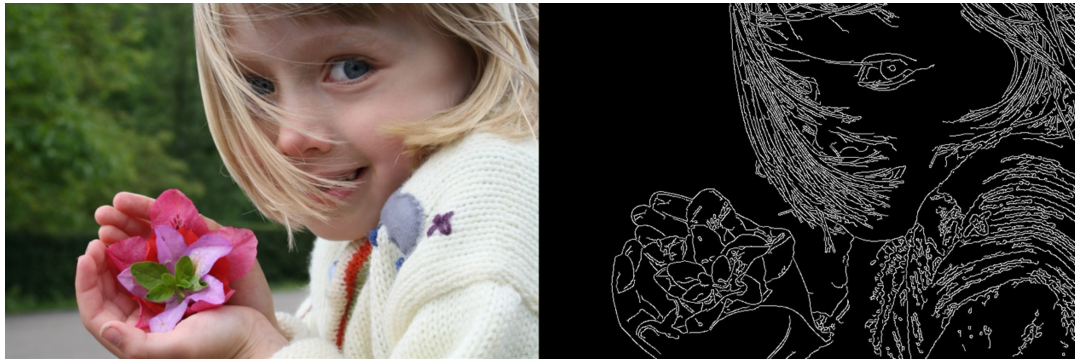
Jingkai Zhang (jz544)
Chen-Tung Chu (cc2396)

Motivation

- Edge detection is commonly implemented in various fields
  - Computer vision
  - Machine learning
- Applications
  - Depth perception
  - Object detection
    - Fingerprint recognition
    - License plate detection
- Computation speed is crucial for such implementation
  - Lower the high-resolution digital images for faster convolution operations.

Edge detection

- What is edge detection?
  - Simplest definition: Sharp changes in the image brightness
    - The points where the image brightness varies sharply are called **the edges** of the image

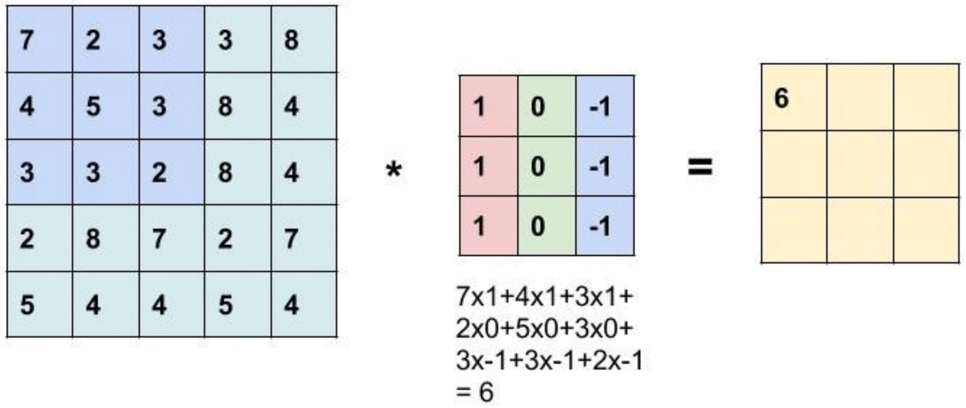


Source: Wiki

- Three major edge detection methods
  - Sobel, Canny, Fuzzy logic

Edge detection (cont.)

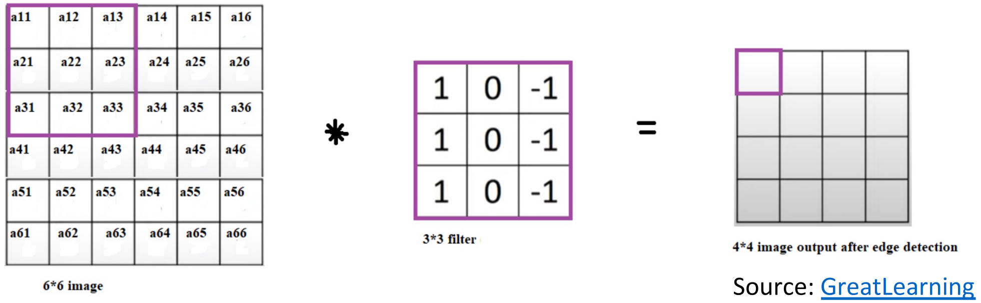
- How to process the high-resolution digital image?
  - Convolution comes to the rescue



$$\begin{bmatrix} 7 & 2 & 3 & 3 & 8 \\ 4 & 5 & 3 & 8 & 4 \\ 3 & 3 & 2 & 8 & 4 \\ 2 & 8 & 7 & 2 & 7 \\ 5 & 4 & 4 & 5 & 4 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 6 & & \\ & & \\ & & \end{bmatrix}$$

7x1+4x1+3x1+
2x0+5x0+3x0+
3x-1+3x-1+2x-1
= 6

Source: medium



$$\begin{bmatrix} a11 & a12 & a13 & a14 & a15 & a16 \\ a21 & a22 & a23 & a24 & a25 & a26 \\ a31 & a32 & a33 & a34 & a35 & a36 \\ a41 & a42 & a43 & a44 & a45 & a46 \\ a51 & a52 & a53 & a54 & a55 & a56 \\ a61 & a62 & a63 & a64 & a65 & a66 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} =$$

Source: GreatLearning

- For the purple square:
  - $output =$

$$a_{11} * 1 + a_{12} * 0 + a_{13} * (-1) + a_{21} * 1 + a_{22} * 0 + a_{23} * (-1) + a_{31} * 1 + a_{32} * 0 + a_{33} * (-1)$$

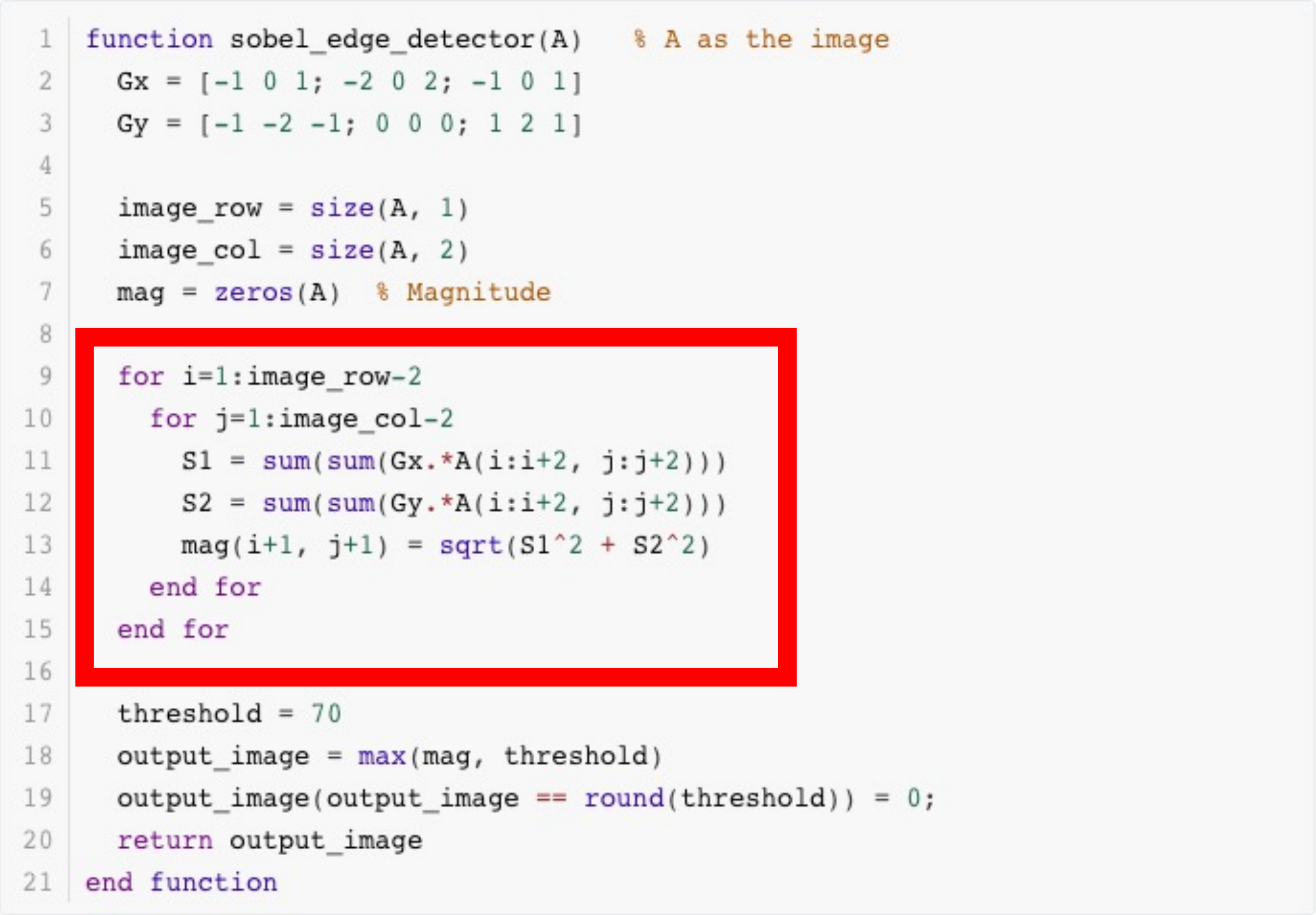
Sobel Edge detection

- Detect the horizontal and vertical edges in images
- Filters:
  - $G_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} * A$ (vertical edge detection)
  - $G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & 2 & -1 \end{bmatrix} * A$ (horizontal edge detection)

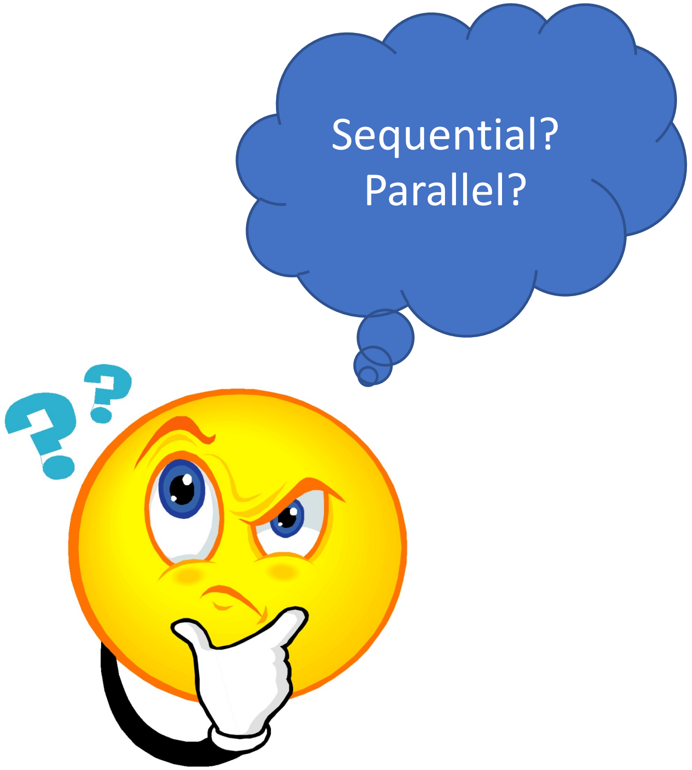
Sobel Edge detection (cont.)

- The Sobel kernels can compute the gradient with smoothing
- $G_x = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} * ([+1 \;\; 0 \;\; -1] * A) \qquad \Longrightarrow \text{increasing the right direction}$
- $G_y = \begin{bmatrix} +1 \\ 0 \\ -1 \end{bmatrix} * ([1 \;\; 2 \;\; 1] * A) \qquad \Longrightarrow \text{increasing the down direction}$
- For each point in the image, the gradient approximation can be combined as the gradient magnitude:
  - $G = \sqrt{G_x^2 + G_y^2}$

Implementation

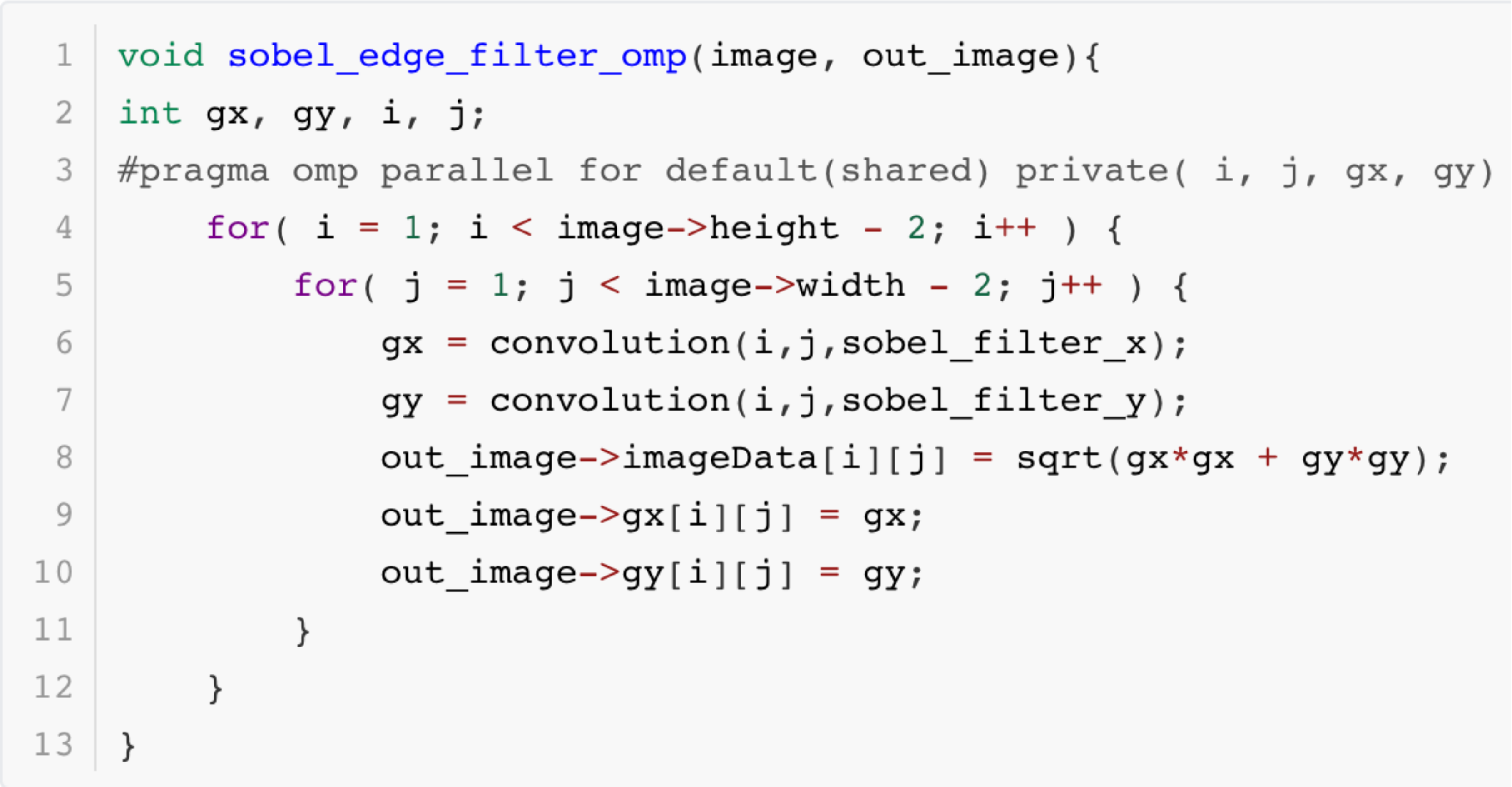


```
function sobel_edge_detector(A)   % A as the image
  Gx = [-1 0 1; -2 0 2; -1 0 1]
  Gy = [-1 -2 -1; 0 0 0; 1 2 1]

  image_row = size(A, 1)
  image_col = size(A, 2)
  mag = zeros(A)  % Magnitude

  for i=1:image_row-2
    for j=1:image_col-2
      S1 = sum(sum(Gx.*A(i:i+2, j:j+2)))
      S2 = sum(sum(Gy.*A(i:i+2, j:j+2)))
      mag(i+1, j+1) = sqrt(S1^2 + S2^2)
    end for
  end for

  threshold = 70
  output_image = max(mag, threshold)
  output_image(output_image == round(threshold)) = 0;
  return output_image
end function
```



Parallel approaches

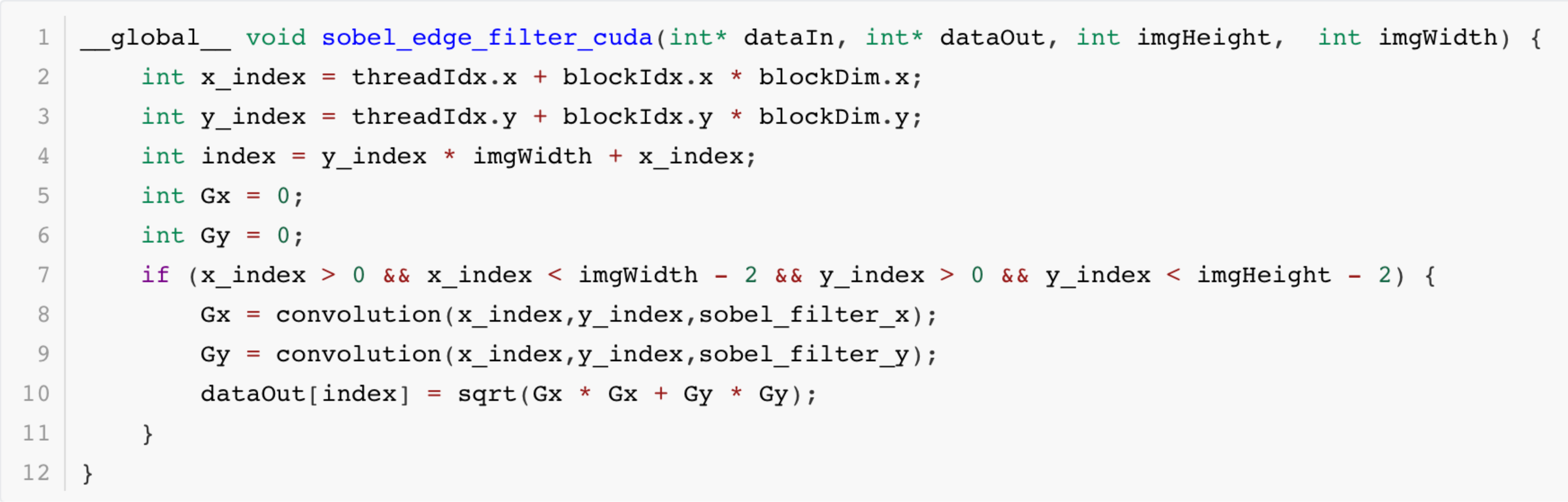
- OpenMP



```
void sobel_edge_filter_omp(image, out_image){
int gx, gy, i, j;
#pragma omp parallel for default(shared) private( i, j, gx, gy)
    for( i = 1; i < image->height - 2; i++ ) {
        for( j = 1; j < image->width - 2; j++ ) {
            gx = convolution(i,j,sobel_filter_x);
            gy = convolution(i,j,sobel_filter_y);
            out_image->imageData[i][j] = sqrt(gx*gx + gy*gy);
            out_image->gx[i][j] = gx;
            out_image->gy[i][j] = gy;
        }
    }
}
```

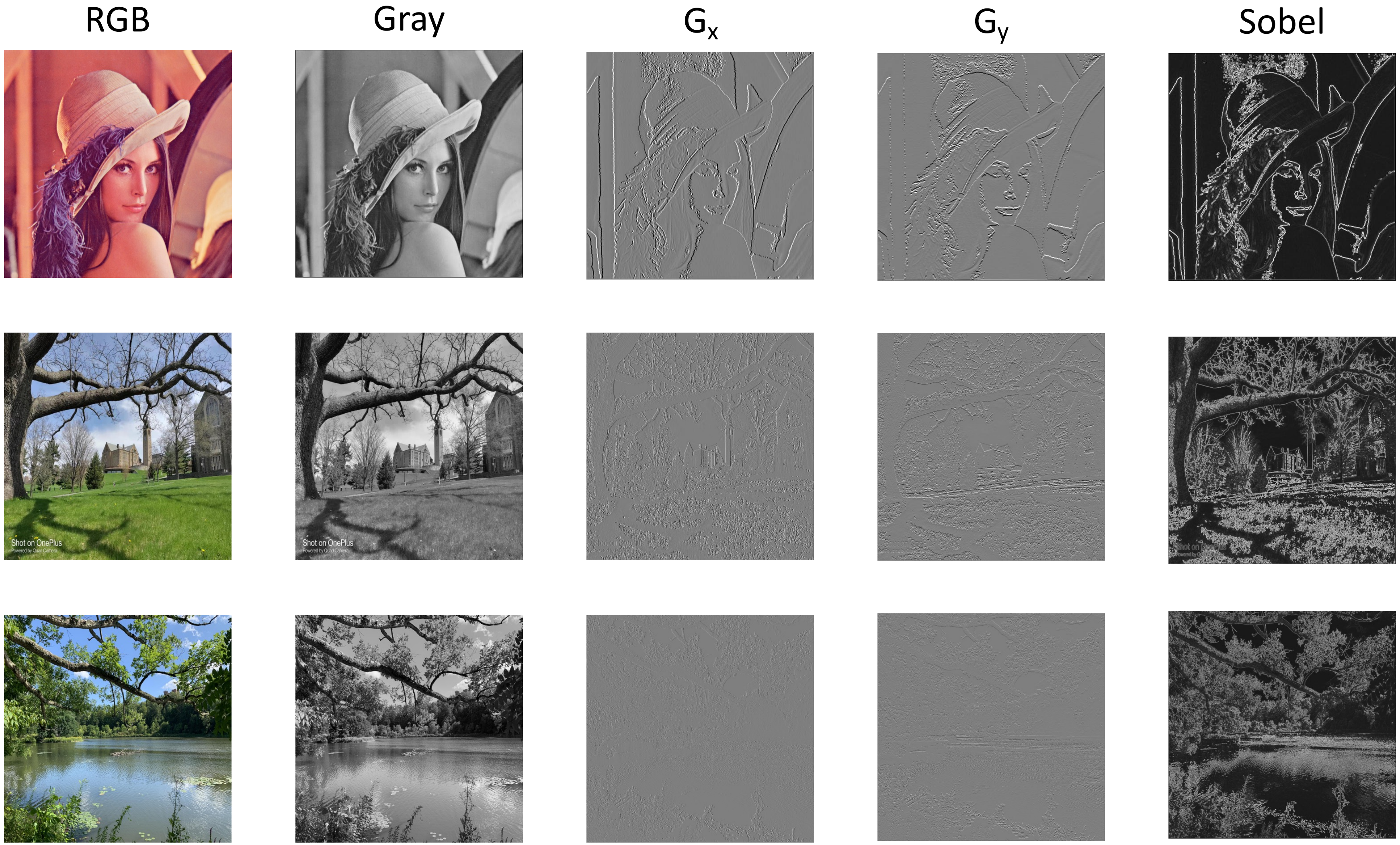
Parallel approaches

- Cuda



```
__global__ void sobel_edge_filter_cuda(int* dataIn, int* dataOut, int imgHeight,  int imgWidth) {
    int x_index = threadIdx.x + blockIdx.x * blockDim.x;
    int y_index = threadIdx.y + blockIdx.y * blockDim.y;
    int index = y_index * imgWidth + x_index;
    int Gx = 0;
    int Gy = 0;
    if (x_index > 0 && x_index < imgWidth - 2 && y_index > 0 && y_index < imgHeight - 2) {
        Gx = convolution(x_index,y_index,sobel_filter_x);
        Gy = convolution(x_index,y_index,sobel_filter_y);
        dataOut[index] = sqrt(Gx * Gx + Gy * Gy);
    }
}
```

Results



RGB

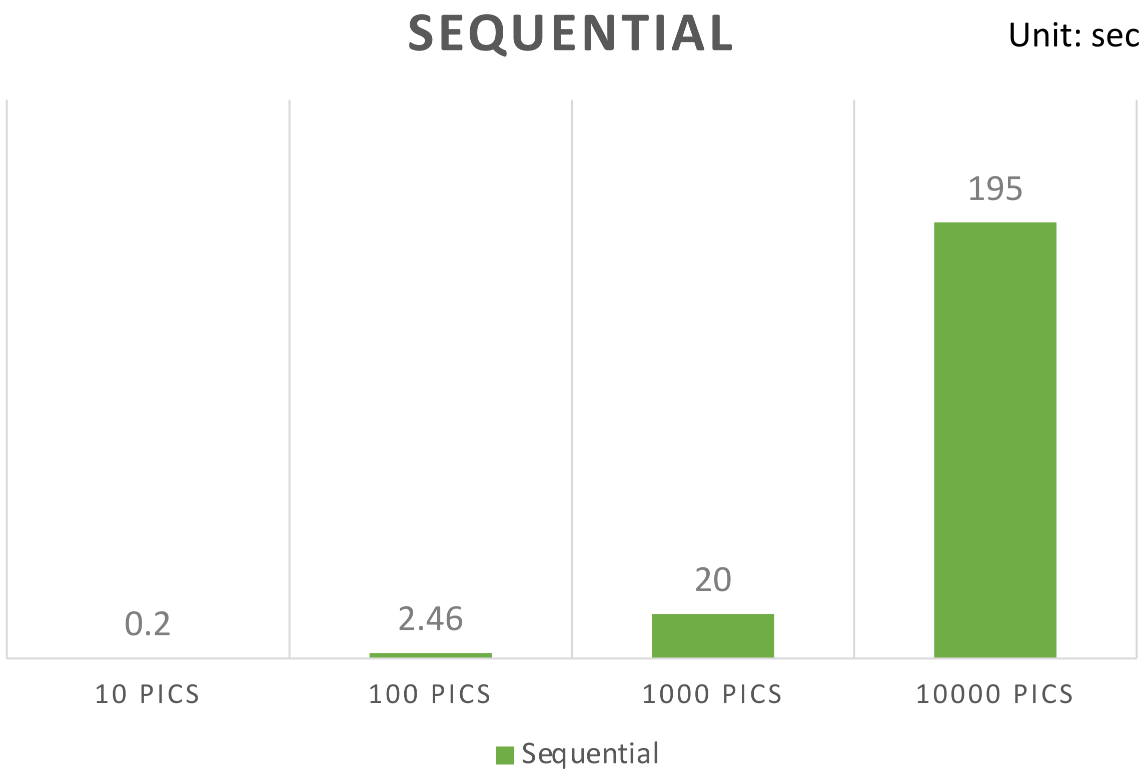
Gray

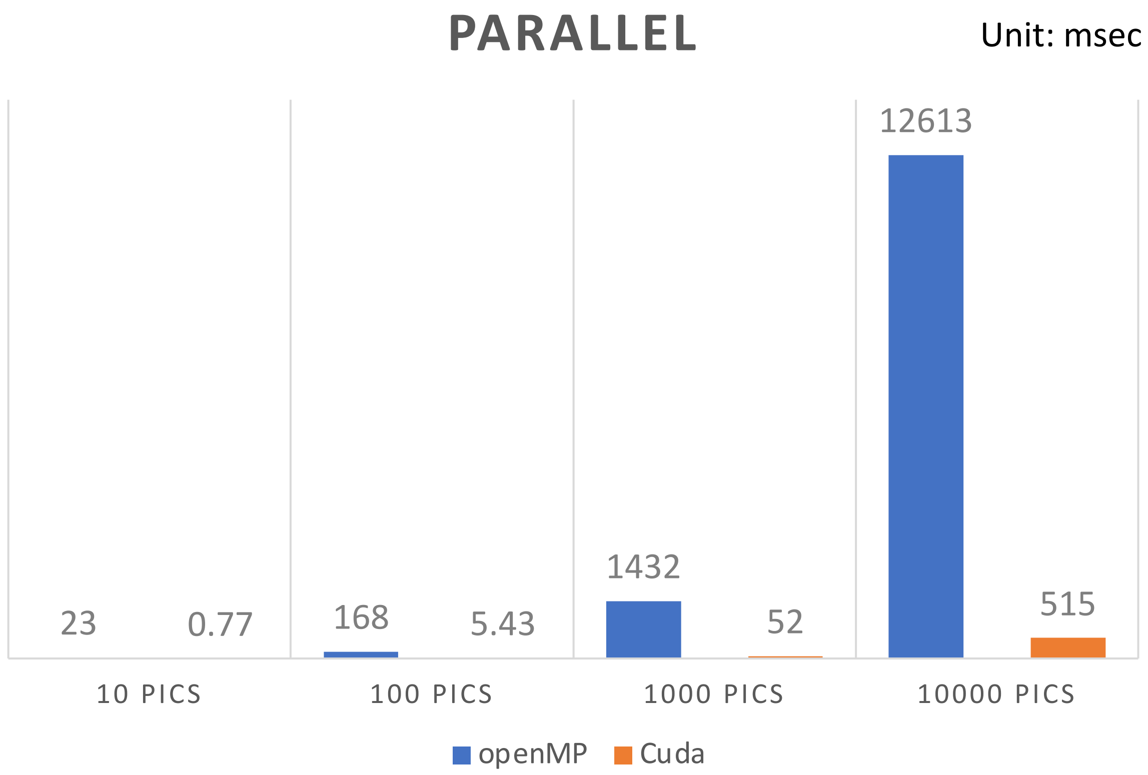
G_x

G_y

Sobel

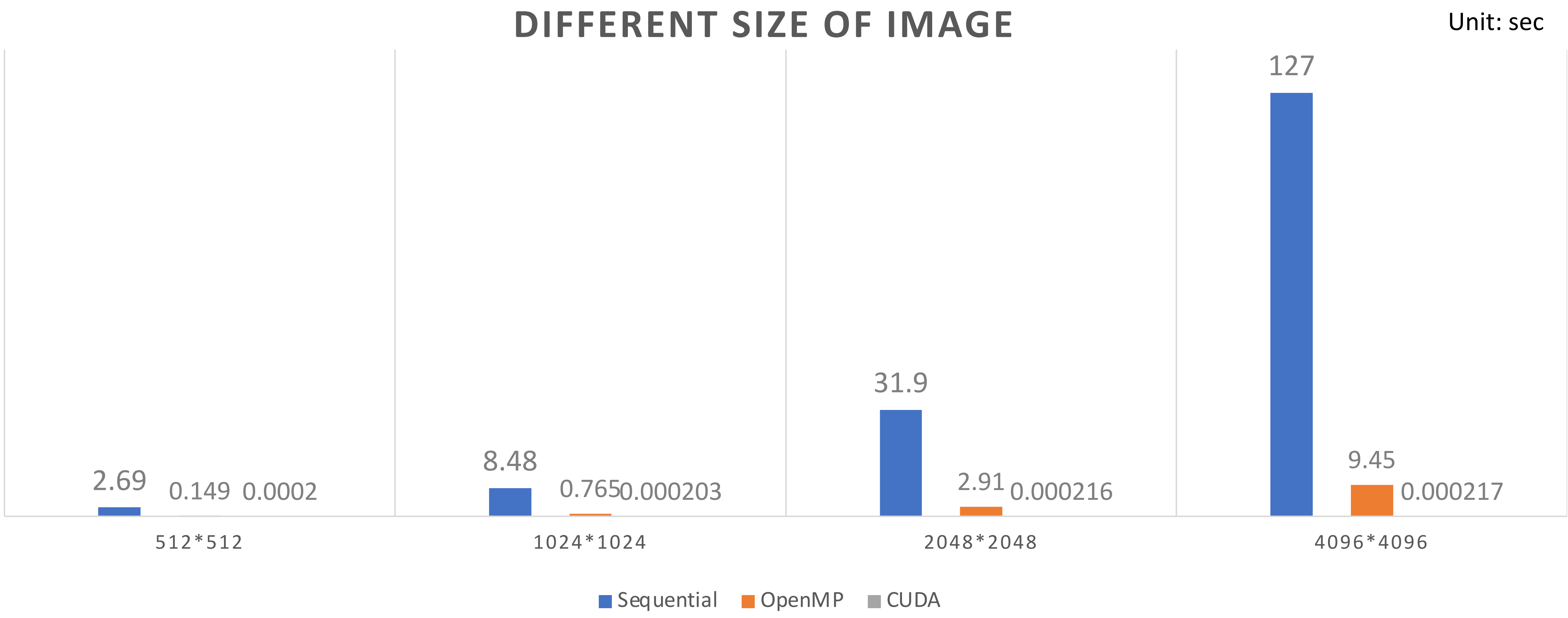
Comparisons



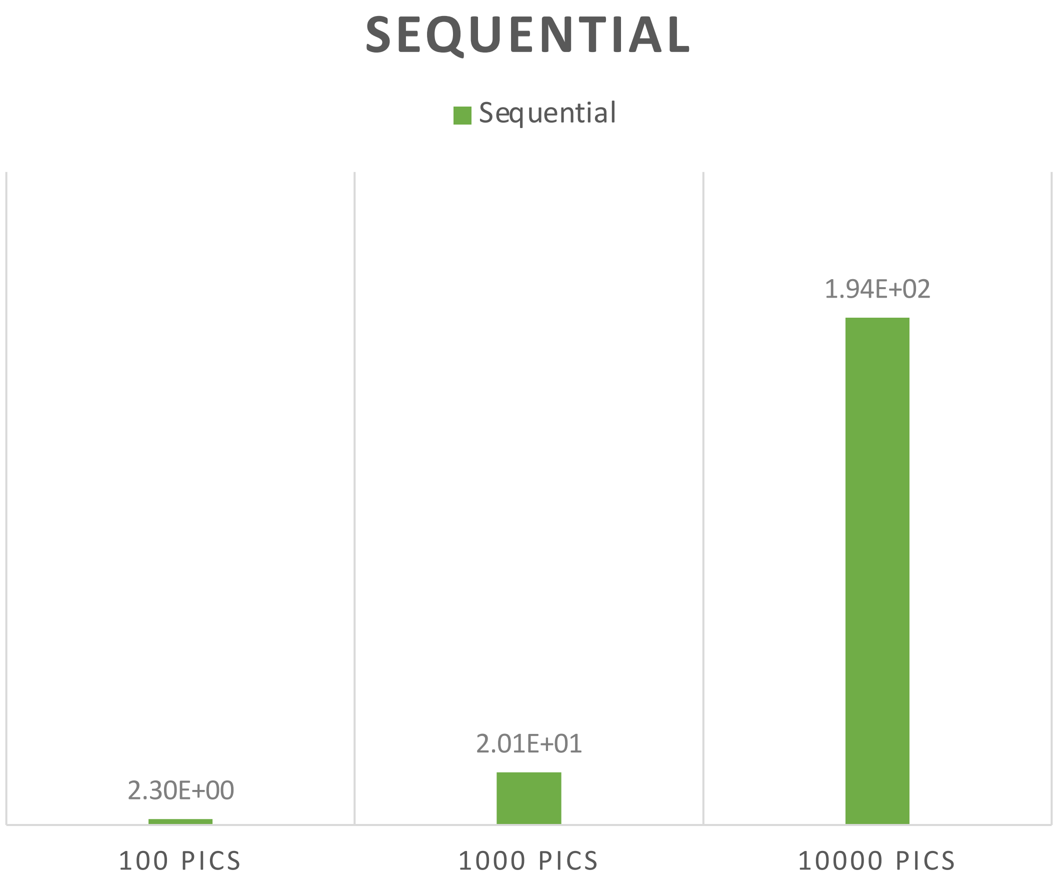


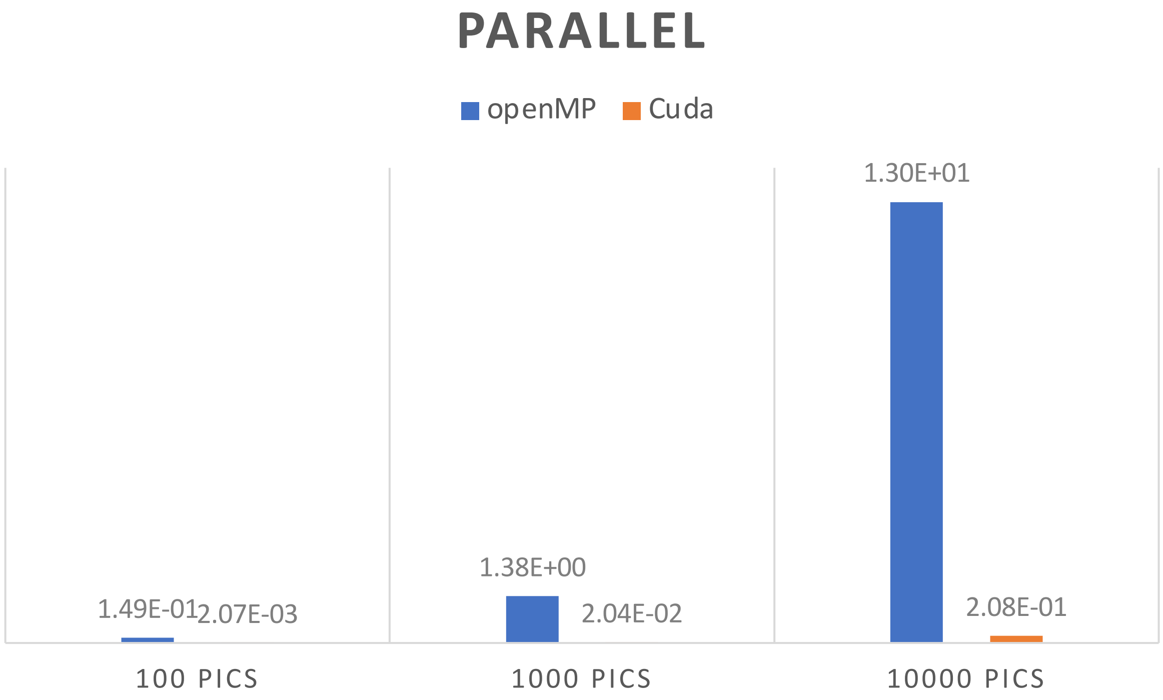
# images: 100
# threads in openMP: 10
Cuda size: grid(30, 30), block(300, 300)

Comparisons



Comparisons (plus loading image time)


SEQUENTIAL
🟩 Sequential
2.30E+00
2.01E+01
1.94E+02
100 PICS
1000 PICS
10000 PICS


PARALLEL
🟦 openMP
🟧 Cuda
1.49E-01
2.07E-03
1.38E+00
2.04E-02
1.30E+01
2.08E-01
100 PICS
1000 PICS
10000 PICS