Problem Statement

Recently, almost of parallel calculations including neural networks are made of GPUs. However, in machine learning using neural networks, especially deep learning, it is necessary to use other resource for other processing like preprocessing of images to maximize the learning speed by conserving the GPU's resources as much as possible, as it consumes hours to days depending on the state of the GPU.

I. Background and Motivation

The Canny edge detector is an edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images. It was developed by John F. Canny in 1986. It was a popular algorithm for finding edge. Recently, however, by applying neural networks to image processing, feature extraction from unstructured data has become a trend in image processing. This trend makes image processing put original or resized images into the learning device and obtain weights for feature extraction.

In certain studies, algorithms such as lane centerline detection also exist, where near-original images are not necessarily required when extracting features from images. However, neural networks are essential for algorithms in autonomous vehicles because variables such as light, weather, and road conditions also exist. In examples such as the previous algorithm, the camera takes the original image as it is, but the neural network does not have to contain the original or resized image. This is due to primary image processing filtering, such as Canny edge, which reduces the burden on GPUs as the index of features is highly compressed in the original image. However, since Canny edge is also an algorithm that uses multiple image processing filters, it can be difficult to use for video normally unless the computational speed of the computer is high. At that time, a useful use is MPI parallel operation using the CPU. As a result, we hope to receive video data with fast primary filtering to reduce the burden on GPUs.

II. 1-core Canny edge detection Model

Canny edge detection algorithm generated as Fig. First, Gaussian filter blur Grayscale image to reduce noise. Second, Sobel mask generates approximate edge magnitude data and direction of edge. Third, non-maxima suppression makes connected line from magnitude data by selecting middle value which is biggest one at 3 values in direction line. Last, Hysteresis Thresholding filters low value data and medium data which is not connected with high value.

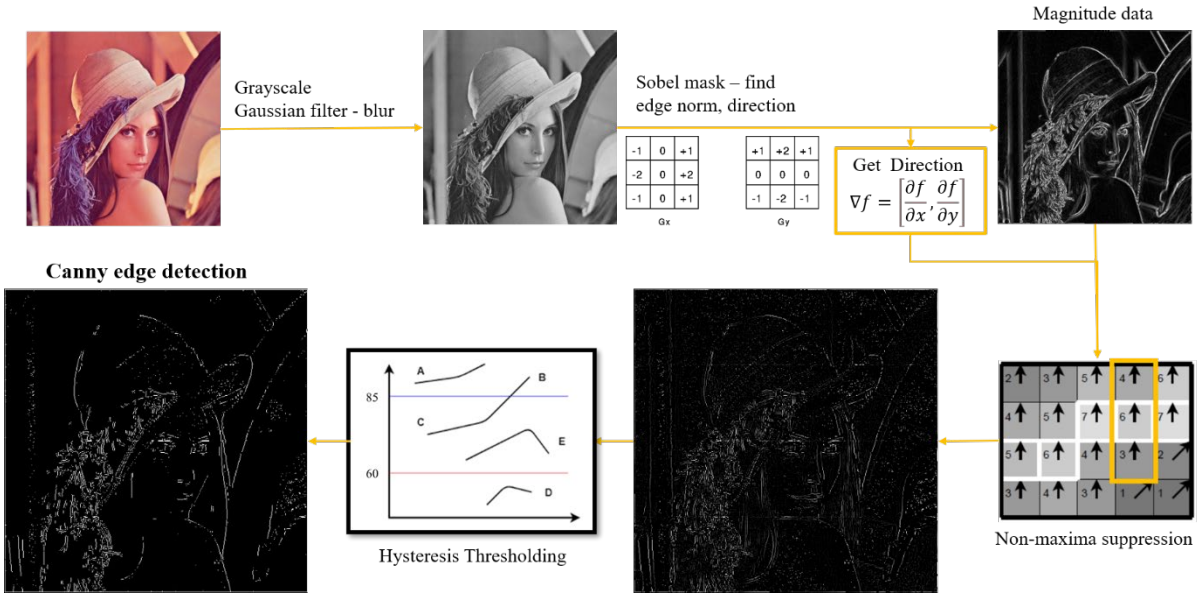


Fig. 1 Flow of Canny edge detection algorithm

III. Parallel Canny edge detection model

Parallel canny edge detection begins after the image is loaded and grayscale work is done. Therefore, the Gaussian filter and the Sobel mask filter are used for convolution of image at parallel from. Convolution is an algorithm that multiplies the opposite value, such as Fig. 2, and the convolution in matrix is also derived by multiplying the value of the opposite position of the matrix, such as (1).

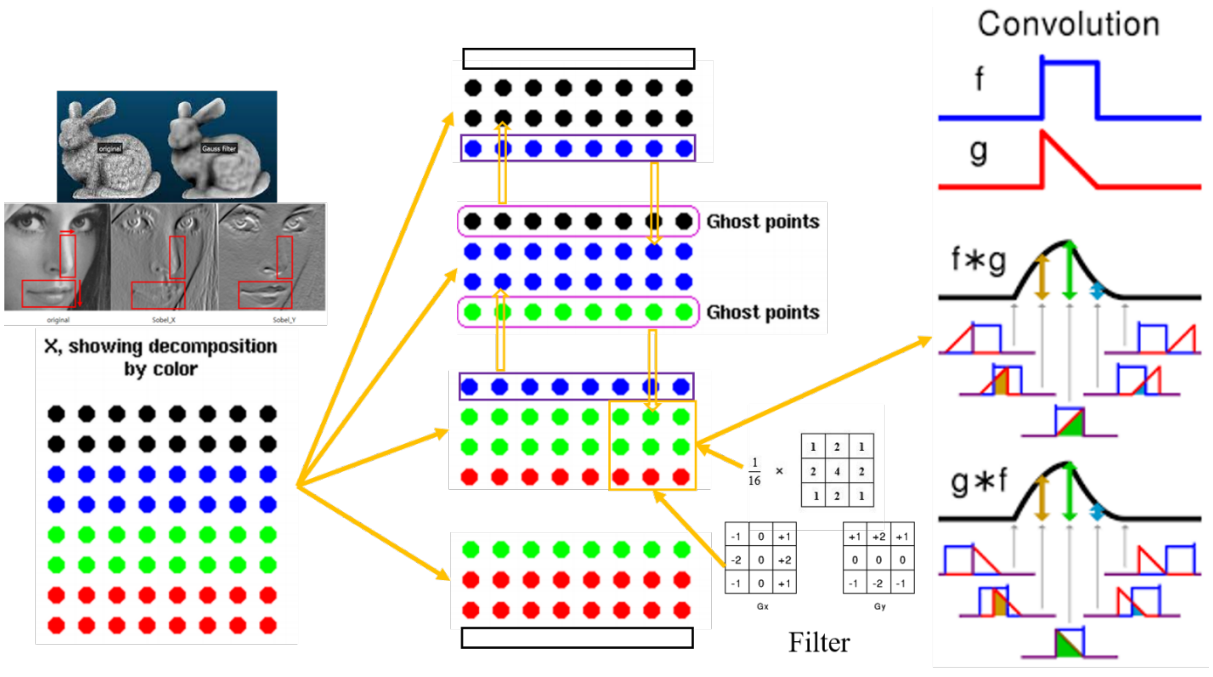


Fig. 2 Parallel convolution algorithm

$$\begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{bmatrix} * \begin{bmatrix} y_{1,1} & y_{12} & \cdots & y_{1n} \\ y_{21} & y_{22} & \cdots & y_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ y_{m1} & y_{m2} & \cdots & y_{mn} \end{bmatrix} = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} x_{(m-i)(n-j)} y_{(1+i)(1+j)} \quad (1)$$

Parallel convolution places ghost points on outer place of each edge to update the value of the edges which is divided by the number of cores, as purple box in Fig. 2. However, due to the nature of the convolution, the position of the first and last edges does not have a number to be convolved. By compensate algorithm, it can be inserted 0 or reduced by one point in the output image. In here, reducing one point way is chosen by placing a black box on the edge in Fig. 2. After convolution happens, in Fig 3, values are exchanged at process 3 and 5 because only values except ghost points are updated, and they are updated by exchanging values to renew them as convolved values.

Fig. 3 is scheme of Schematic of parallel canny edge detection algorithm as 4 cores. To put it in order,

1. make basis image Grayscale and scatter at each core with image/core+2 size array.

   (First and last one has image/core+1)

2. convolve 3*3 Gaussian blur for each image cell without edge position pixel.

3. Update ghost point by exchanging value of close core

4, 5. Make same progress as 2, 3 by sobel mask filter to get magnitude value and direction value

6. Make Non-maxima suppression using 3*3 cells with direction value from sobel mask

7. Make hysteresis thresholding using 3*3 cells with optimized value and gather separated image.

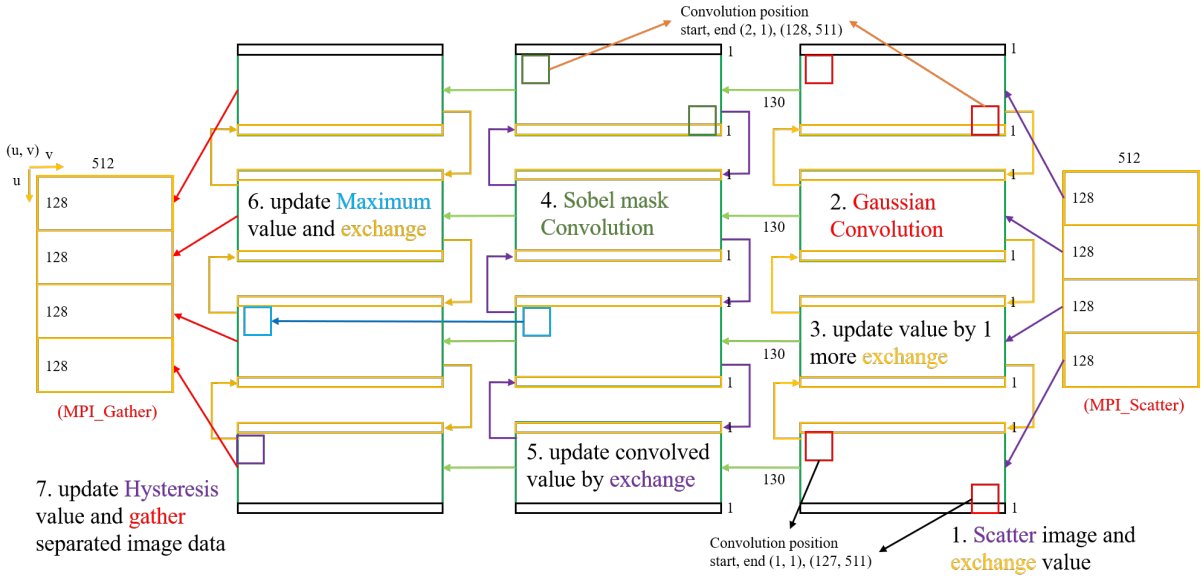
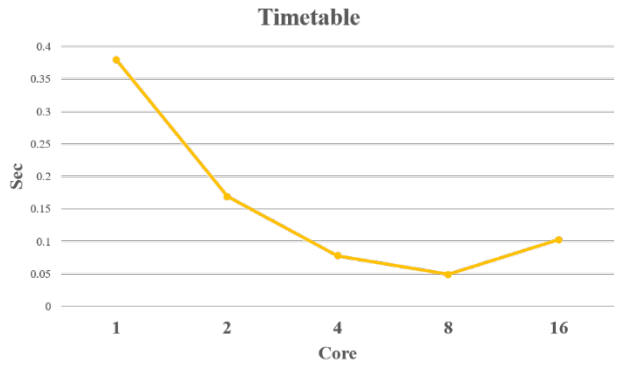


Fig. 3 Schematic of parallel canny edge detection algorithm

At last, the speed of algorithm is needed to be analyzed. In algorithm, time has measured after grayscale image, and it ended at time of gathering image by using “MPI_Wtime” function. As I expected, Time goes lower by the number of cores until 8 number of physical cores but after physical number of cores, calculation time get increased.



Core	1	2	4	8	16
try 1	0.389	0.155021	0.07833	0.050662	0.097252
try 2	0.38	0.202241	0.078741	0.048344	0.093488
try 3	0.369	0.149224	0.076382	0.047567	0.117237
Average(sec)	0.379333	0.168829	0.077818	0.048858	0.102659

Fig. 4 Average process speed per core

Result