Proc. of Int. Conf. on Current Trends in Eng., Science and Technology, ICCTEST

A Review on FPGA Implementation of Distributed Canny Edge Detector

Chandrashekar N.S [1] and Dr. K.R. Nataraj [2]
[1]Don Bosco Institute of Technology/Department of ECE, Bengaluru, India.
Email: chandrashekarns.dbit@gmail.com
[2]SJB Institute of Technology / Department of ECE, Bengaluru, India.
Email: nataraj.sjbit@gmail.com

***Abstract*—In this paper, we present a distributed Canny edge detection algorithm that results in significantly reduced memory requirements decreased latency and increased throughput with no loss in edge detection performance as compared to the original Canny algorithm. The new algorithm uses a low-complexity 8-bin non-uniform gradient magnitude histogram to compute block-based hysteresis thresholds that are used by the Canny edge detector. Furthermore, FPGA-based hardware architecture of our proposed algorithm is presented in this paper and the architecture is synthesized on the Xilinx Virtex 4 FPGA. The design development is done in VHDL and simulates the results in modelsim 6.3 using Xilinx 12.2. *Keywords*: Canny Edge detector, Distributed Processing, Non-uniform quantization, FPGA.**

I. INTRODUCTION

Edge detection is a very important first step in many algorithms used for segmentation, tracking and image/video coding. The Canny edge detector is predominantly used due to its ability to extract significant edges [1]. Edge detection, as a basic operation in image processing, has been researched extensively. A lot of edge detection algorithms, such as Robert detector, Prewitt detector, Kirsch detector, Gauss-Laplace detector and Canny detector have been proposed. Among these algorithms, Canny algorithm has been used widely in the field of image processing because of its good performance [2]. The Canny edge detector is predominantly used in many real-world applications due to its ability to extract significant edges with good detection and good localization performance. Unfortunately, the Canny edge detection algorithm contains extensive pre-processing and post-processing steps and is more computationally complex than other edge detection algorithms. Furthermore, it performs hysteresis thresholding which requires computing high and low thresholds based on the entire image statistics. This places heavy requirements on memory and results in large latency, hindering real-time implementation of the Canny edge detection algorithm [3]. Implementing image processing algorithms on reconfigurable hardware minimizes the time-to-market cost, enables rapid prototyping of complex algorithms and simplifies debugging and verification [4]. Edge detectors based on the first derivative do not guarantee to produce edge maps with continuous edge contours nor unwanted branches. Edge detectors based on the second derivatives, such as zero crossing; suffer from generating erroneous edges in textured images because of its high sensitivity to noise. Being an effective edge detector with single-pixel response, Canny operator has been widely used in accurately abstracting the edge information in image processing. However, taking its 4-step process into account, its real-time implementation based on CPU has become a significant problem, especially for the part of the edge tracing, which consumes a large amount of computing time. To solve the problem, GPU will be used for sake of its powerful ability of parallel processing while a new Canny operator is proposed with the introduction of parallel breakpoints detection and edge tracing without recursive operations [5] .The original Canny algorithm computes the higher and lower thresholds for edge detection based on the entire image statistics, which prevents the processing of blocks independent of each other [1]. In order to reduce memory requirements, decreased latency and increased throughput, a distributed canny edge detection algorithm is proposed in [1]. The hysteresis threshold calculation is a key element that greatly affects the edge detection results. In [3], it is proposed a new threshold selection algorithm based on the distribution of pixel gradients in a block of pixels to overcome the dependency between the blocks. However, in [1], the hysteresis thresholds calculation is based on a very finely and uniformly quantized 64-bin gradient magnitude histogram, which is computationally expensive and thereby, hinders the real-time implementation. In this paper, a method based on non-uniform and coarse quantization of the gradient magnitude histogram is proposed. In addition, the proposed algorithm is mapped onto reconfigurable hardware architecture. The threshold is calculated using the data of the histogram of gradient magnitude rather than is set manually in a failure-and-try fashion and can give quite good edge detection results without the intervening of an operator [6]. This improved new Canny algorithm is also implemented on FPGA (field programmable gate array) to meet the needs of real time processing. This paper is organized as follows: Section 2 gives a brief overview of the original Canny edge detector algorithm. Section 3 presents the proposed distributed Canny edge detection algorithm which includes a novel method for the hysteresis thresholds computation based on a non-uniform quantized gradient magnitude histogram. The proposed hardware architecture and FPGA implementation algorithm are described in Section 4. Simulation results are presented in Section 5. A conclusion is given in Section 6.

II. CANNY EDGE DETECTOR

The popular Canny edge detector uses the following steps to find contours presents in the image. The first stage is achieved using Gaussian smoothing. The resulting image is sent to the PC that sends it back to the gradient filter, but here we modified our gradient filter a bit because this time we don't only need the gradient magnitude that is given by our previous operator, but we need separately Gx and Gy. We also need the phase or orientation of our gradient which is obtained using the following formula: θ = arctan As we can see, this equation contains an arctan and a division. These operators are very difficult to implement using hardware. We also don't need a high precision. The final θ has to give only one of the four following possible directions, as we can see in Figure 1. The fourth direction is the horizontal direction with zero degrees, not indicated in the figure. Figure 1: Possible Directions for the Gradient Phase Arctan and the division can be eliminated by simply comparing Gx and Gy values. If they are of similar length, we will obtain a diagonal direction, if one is at least 2.5 times longer than the other, we will obtain a horizontal or vertical direction. After the edge directions are known, non-maximum suppression is applied. Nonmaximum suppression is used to trace pixels along the gradient in the edge direction and compare the values perpendicular to the gradient. Two perpendicular pixel values are compared with the value in the edge direction. If their value is lower than the pixel on the edge, then they are suppressed i.e. their pixel value is changed to 0, else the higher pixel value is set as the edge and the other two are suppressed with a pixel value of 0. Finally, hysteresis is used as a means of eliminating streaking. Streaking is the breaking up of an edge contour caused by the operator output fluctuating above and below the threshold. If a single threshold, T1 is applied to an image, and an edge has an average strength equal to T1, then due to noise, there will be instances where the edge dips below the threshold. Equally it will also extend above the threshold making an edge look like a dashed line. To avoid this, hysteresis uses 2 thresholds, a high and a low. Any pixel in the image that has a value greater than T1 is presumed to be an edge pixel, and is marked as such immediately. Then, any pixels that are connected to this edge pixel and that have a value greater than T2 are also selected as edge pixels. If you think of following an edge, you need a gradient of T2 to start but you don't stop till you hit a gradient below T1 [7]. The original Canny algorithm [3] shown in Fig 2, consists of the following steps executed sequentially:

1. Low pass filtering the image with a Gaussian mask.
2. Computing horizontal and vertical gradients at each pixel location.
3. Computing the gradient magnitude at each pixel location.
4. Computing a higher and lower threshold based on the histogram of the gradients of the entire image.

5. Suppressing non-maximal strong (NMS) edges.
6. Computing the hysteresis high and low thresholds based on the histogram of the magnitudes of the gradients of the entire image.
7. Performing hysteresis thresholding to determine the edge map.

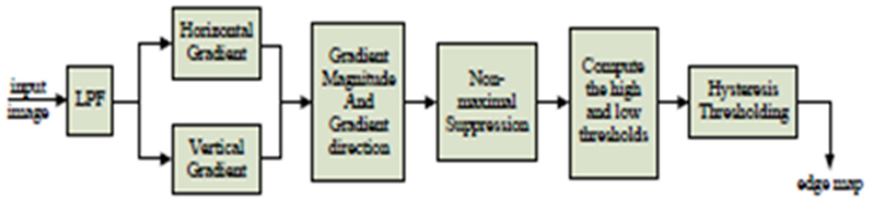

Figure 2: Block Diagram of the Canny Edge Detection

In our implementation, the architecture is synthesized on the Xilinx VIRTEX 4 FPGA. The results show that a 16-core architecture (3 X 3 block size for 256 X 256 image) leads to 16 times decrease in running time without performance degradation when compared with the original frame-based Canny algorithm.

III. Proposed Distributed Canny Edge Detection Algorithm

The superior performance of the frame-based Canny algorithm is due to the fact that it computes the gradient thresholds by analyzing the histogram of the gradients at all the pixel locations of an image. Though it is purely based on the statistical distribution of the gradient values, it works well on natural images which consist of a mix of smooth regions, texture regions and high-detailed regions [1]. Directly applying the frame-based Canny at a block-level would fail because such a mix of regions may not be available locally in every block of the frame. This would lead to excessive edges in texture regions and loss of significant edges in high detailed regions. The Canny edge detection algorithm operates on the whole image and has a latency that is proportional to the size of the image. While performing the original canny algorithm at the block-level would speed up the operations, it would result in loss of significant edges in high-detailed regions and excessive edges in texture regions. Natural images consist of a mix of smooth regions, texture regions and high-detailed regions and such a mix of regions may not be available locally in every block of the entire image. In [1], it is proposed a distributed Canny edge detection algorithm, which removes the inherent dependency between the various blocks so that the image can be divided into blocks and each block can be processed in parallel. The input image is divided into $m \times m$ overlapping blocks. The adjacent blocks overlap by $(L - 1)/2$ pixels for a $L \times L$ gradient mask. However, for each block, only edges in the central $n \times n$ (where $n = m + L - 1$) non-overlapping region are included in the final edge map. Steps 1 to 4 and Step 6 of the distributed Canny algorithm are the same as in the original Canny algorithm except that these are now applied at the block level. Step 5, which is the hysteresis high and low thresholds calculation, is modified to enable parallel processing. In [1], a parallel hysteresis thresholding algorithm was proposed based on the observation that a pixel with a gradient magnitude of 2, 4 and 6 corresponds to blurred edges, psycho visually significant edges and very sharp edges, respectively. In order to compute the high and low hysteresis thresholds, very finely and uniformly quantized 64-bin gradient magnitude histograms are computed over overlapped blocks. If the 64-bin uniform discrete histogram is used for the high threshold calculation, this entails performing 64 multiplications and $64 \times Np$ comparisons, where Np is the total number of pixels in an image. Therefore, it is necessary to find a good way to reduce the complexity of the histogram computation. As in [6], it was observed that the largest peak in the gradient magnitude histograms after NMS of the Gaussian smoothed natural images occurs near the origin and corresponds to low-frequency content, while edge pixels form a series of smaller peaks where each peak corresponds to a class of edges having similar gradient magnitudes. The proposed distributed thresholds selection algorithm is shown in Fig.3. Let Gt be the set of pixels with gradient magnitudes greater than a threshold t, and let NGt for t = 2, 4, 6, be the number of corresponding gradient elements in the set Gt. Using NGt, an intermediate classification threshold C is calculated to indicate whether the considered block is high-detailed, moderately edged, blurred or textured, as shown in Fig.3. Consequently, the set Gt = Gt=c can be selected for computing the high and low thresholds. The high threshold is calculated based on the histogram of the set Gc such that 20% of the total pixels of the block would be identified as strong edges. The lower threshold is the 40% percentage of the higher threshold as in the original Canny algorithm.

```
Let  $Gt$: set of pixels with gradient magnitudes greater
          than a threshold $t$
     $N_{Gt}$: the number of elements in the set $Gt$

Step 1:  Determine $Gt$ for $t$ = 2, 4, and 6 and $N_{Gt}$ for $t$ =2, 4, 6
Step 2:  If ($N_{G4}$ > 0.25 * Total_block_pels)
                 $C = 6$                  /*High-detailed*/
         else If ($N_{G4}$ > 0.05 * Total_block_pels)
                 $C = 4$                  /*moderately-edged*/
         else If ($N_{G2}$ < 0.25 * Total_block_pels)
                 $C = 2$                  /*blurred*/
          Else
                 Exit;                    /*textured*/
Step 3: Compute the 8-bin non-uniform gradient magnitude
        histogram of $Gc$ and the corresponding cumulative
        distribution function F($Gc$).
Step 4: Compute High_threshold as F(High_threshold) = 0.8
Step 5: Compute Low_threshold  = 0.4*High_threshold
```

Figure 3: Pseudo-code of the proposed Distributed threshold Selection Scheme

We compared the high threshold value that is calculated using the proposed distributed algorithm based on an 8-bin non-uniform gradient magnitude histogram with the value obtained when using a 16-bin non-uniform gradient magnitude histogram. These two high thresholds have similar values. Therefore, we use the 8-bin non-uniform gradient magnitude histogram in our implementation.

IV. Implementation Of The Proposed Distributed Canny Algorithm

In this section, we describe the hardware implementation of our proposed distributed canny edge detection algorithm on the Xilinx VIRTEX 4 FPGA. We provide a high-level architecture diagram as follows. *Architecture:* Depending on the available FPGA resources, the image needs to be partitioned into q sub-images and each sub-image is further divided into p m×m blocks. The proposed architecture, shown in Fig. 4, consists of q processing units in the FPGA and some Static RAMs (SRAM) organized into q memory banks to store the image data, where q equals to the image size divided by the SRAM size.

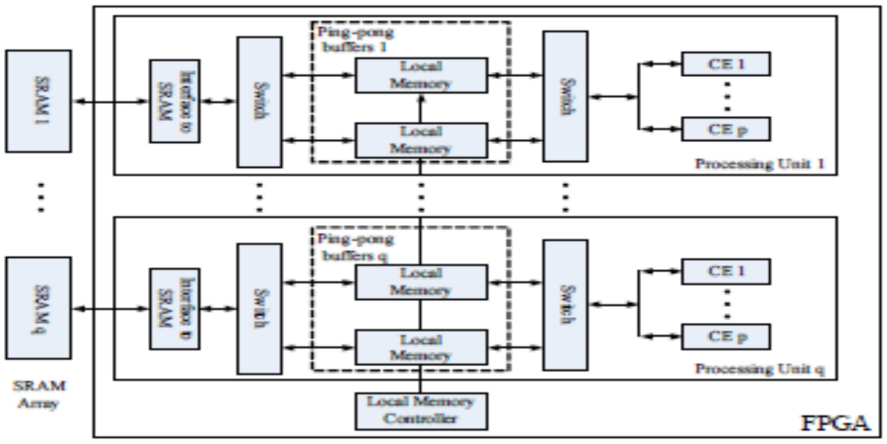

Figure 4: The Architecture of the proposed Distributed Canny Algorithm

Each processing unit processes a sub-image and reads/writes data from/to the SRAM through ping-pong buffers, which are implemented with dualport Block RAMs (BRAM) on the FPGA. As shown in Fig.4, each processing unit (PU) consists of p computing engines (CE), where each CE detects the edge map of an m×m block image. Thus, p×q blocks can be processed at the same time and the processing time for an N×N image is reduced, in the best case, by a factor of p×q. The specific values of p and q depend on the processing time of each PE, the data loading time from the SRAM to the local memory and the interface between FPGA and SRAM, such as total pins on the FPGA, the data bus width, the address bus width and the maximum system clock of the SRAM. In our application, we choose p = 2 and q = 8. In the proposed architecture, each CE consists of the following 6 units, as shown in Fig.5: 1. Smoothening unit using Gaussian filter. 2. Vertical and horizontal gradient calculation unit. 3. Magnitude calculation unit. 4. Directional non-maximum suppression unit. 5. High and low threshold Calculation unit. 6. Thresholding with hysteresis unit.

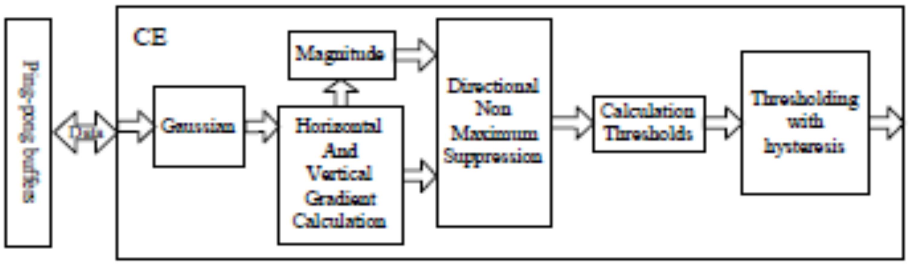
Ping-pong buffers
Data
CE
Gaussian
Magnitude
Horizontal And Vertical Gradient Calculation
Directional Non Maximum Suppression
Calculation Thresholds
Thresholding with hysteresis

Figure 5: Block diagram of the CE (compute engine) for the proposed Distributed Canny edge Detection

V. Simulation Results And Analysis

Figure 6.shows the implementation software result and the FPGA implementation generated result. For the 256 X 256 camaramen image using the proposed distributed Canny edge detector with block size of 3 X 3 and a 3 × 3 gradient mask. The FPGA result is obtained using ModelSim.

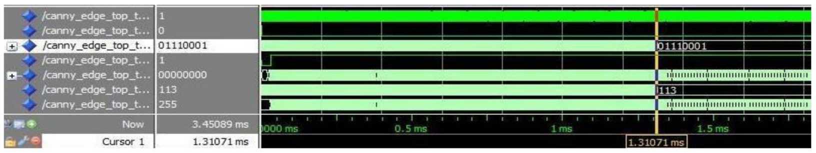

Figure 6: Simulation waveform for canny Edge Detection system

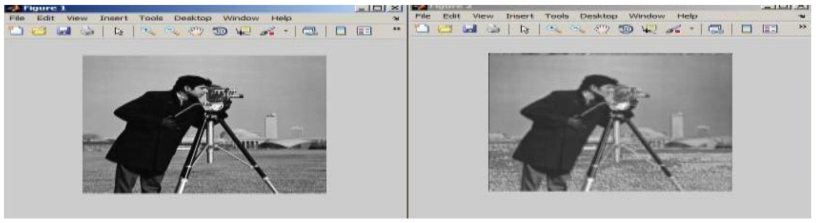

Figure 7: Comparision results of 256 X 256 camaramen image (a) Original image (b) Bluring image

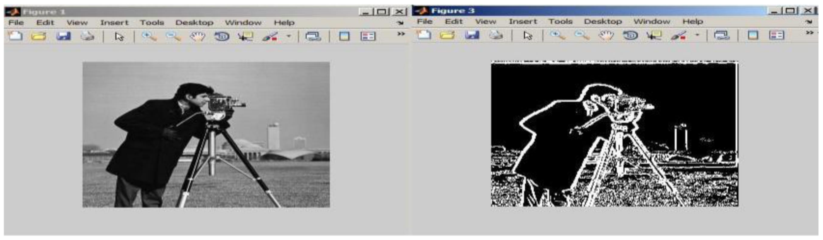

Figure 8: Comparision results of 256 X 256 camaramen image (a) original image and (b) Edge map image

VI. Conclusion

We proposed a novel nonuniform quantized histogram calculation method in order to reduce the computational cost of the hysteresis threshold selection. As a result, the computational cost of the proposed algorithm is very low compared to the original Canny edge detection algorithm. The algorithm is mapped to onto a Xilinx Virtex-4 FPGA platform and tested using ModelSim. It is capable of supporting fast real-time edge detection for images and videos with various spatial and temporal resolutions including full-HD content. For a 100 MHz clock rate, the total processing running time using the FPGA implementation is 0.655 ms for a 256 X 256 image.

REFERENCES

[1] S. Varadarajan, C. Chakrabarti, L. J. Karam, and J. M.Bauza, “A Distributed psycho-visually motivated canny edge detector,” IEEE ICASSP, pp. 822 –825, Mar. 2010.

[2] L. Torres, M. Robert, E. Bourennane, and M. Paindavoine, “Implementation of a recursive real time edge detector using Retiming techniques,” VLSI, pp. 811 –816, Aug. 1995.

[3] Qian Xu, Chaitali Chakrabarti and Lina J. Karam, “A Distributed Canny Edge Detector and Its Implementation On FPGA”, Tempe, AZ.

[4] D. V. Rao and M. Venkatesan, “An efficient reconfigurable Architecture and implementation of edge detection algorithm Using Handle-C,” ITCC, vol. 2, pp. 843 – 847, Apr. 2004.

[5] Shengxiao Niu, Jingjing Yang, Sheng Wang, Gengsheng Chen,”Improvement and Parallel Implementation of Canny Edge Detection Algorithm Based on GPU”.

[6] W. He and K. Yuan, “An improved Canny edge detector and its Realization on FPGA,” WCICA, pp. 6561 –6564, Jun. 2008.

[7] J. Canny, “A computational approach to edge detection,”IEEE Trans. PAMI, vol. 8, no. 6, pp. 679 –698, Nov. 1986.