FPGA Implementation of Object Recognition Processor for HDTV Resolution Video Using Sparse FIND Feature

Yuri Nishizumi1, Go Matsukawa1, Koichi Kajihara1, Taisuke Kodama1, Shintaro Izumi1, Hiroshi Kawaguchi1, Chikako Nakanishi2, Toshio Goto3, Takeo Kato4 and Masahiko Yoshimoto1
1 Graduate School of System Informatics, Kobe University, Kobe, 657-8501 Japan
2 Osaka Institute of Technology, Osaka, 535-8585 Japan
3 Electronics advanced development department, Toyota Motor Corporation, Toyota, 471-8571 Japan
4 Toyota Central R&D Lab., Inc., Nagakute, 480-1192 Japan
nishizumi.yuri@cs28.cs.kobe-u.ac.jp

Abstract—This paper describes FPGA implementation of object recognition processor for HDTV resolution 30 fps video using the Sparse FIND feature. Two-stage feature extraction processing by HOG and Sparse FIND, a highly parallel classification in the support vector machine (SVM), and a block-parallel processing for RAM access cycle reduction are proposed to perform a real time object recognition with enormous computational complexity. From implementation of the proposed architecture in the FPGA, it was confirmed that detection using the Sparse FIND feature was performed for HDTV images at 47.63 fps, on average, at 90 MHz. The recognition accuracy degradation from the original Sparse FIND-base object detection algorithm implemented on software was 0.5%, which shows that the FPGA system provides sufficient accuracy for practical use.

Keywords—FPGA; Sparse FIND; HDTV resolution

I. INTRODUCTION

Real-time object recognition is a fundamentally important technology for surveillance cameras, robots, and automobile applications. Particularly as a means to prevent traffic accidents, driving support systems that use computer vision technology are effective at making drivers aware of danger. To detect a specific object from an input image, techniques for efficiently extracting feature information are important. The Histograms of Oriented Gradients (HOG) method [1] has been used for conventional object detection. This feature descriptor, which is robust to luminance change, is independent of the object texture. Features that have higher dimensions than those obtained in HOG are proposed for more accurate object detection. Among them, Sparse Feature Interaction Descriptor (FIND) [2] shows higher recognition accuracy than either HOG or Co-occurrence Histograms of Oriented Gradients (CoHOG) [3]. The processing capability of general purpose processors has improved along with development of semiconductor technology in recent years. Using general purpose processors with high performance or GPUs that can accommodate highly parallel processing, it has become possible to perform object recognition requiring extensive computation in real time. However, general purpose processors and GPUs have the shortcoming of high power consumption. For that reason, they are unsuitable for mobile systems under restrictions of battery capacity and thermal design. Furthermore, for applications requiring distant object detection such as in-vehicle systems, it is necessary to handle high-resolution images such as HDTV resolution (1920 × 1080 pixels). The high-resolution image makes it possible to capture detailed shapes of objects with a wide field angle. However, increasing computational complexity and memory capacity necessary for the real time processing are problems that increase power consumption. Therefore, we developed an FPGA accelerator to execute real-time object recognition employing Sparse FIND feature for HDTV resolution video.

The Sparse FIND algorithm and issues related to hardware implementation are described in Chapter 2. The proposed design techniques used for the object detection processor developed are explained in Chapter 3. Chapter 4 explains the processor architecture. Chapter 5 describes the evaluation results. Chapter 6 summarizes the conclusions of this paper.

II. OUTLINE OF SPARSE FIND ALGORITHM

Sparse FIND is an image feature extraction algorithm [2] that reduces the calculation costs of the FIND [4] feature created using the HOG feature. The FIND feature is calculated by obtaining the correlation using all HOG feature elements and performing normalization of the correlation. Because the object shape can be expressed more finely, the detection accuracy is better than that of the HOG feature. However, many calculation costs arise. Therefore, among the elements of the HOG feature, only elements with high validity in identification are extracted (sparsified). Correlation is used to reduce the number of dimensions. In addition, by obtaining the dimensionless coefficients beforehand and accumulating them at the time of calculating the feature, the normalization which was necessary for post-processing is eliminated. Through these improvements, calculation costs are reduced while maintaining detection performance using FIND features.

Sparse FIND feature calculation method will be described in order. First, the gradient strength and gradient direction are
calculated from the luminance value of each pixel of the input image. Using these, the luminance gradient histogram in the direction \(d\) in a cell of \(p \times p\) pixels are created. \(q \times q\) cells are defined as one block. The luminance gradient histograms for one block in a detection window are arranged in a row to create HOG feature vectors of

\[ H = (h_1, \ldots, h_m), \]

as depicted in Fig. 1. The number of dimensions per block is \(m = d \times q \times q\).

Next, the sparsification threshold \(th\) and the dimensionless coefficient \(a\) are calculated from the HOG feature. Using the sparsification threshold \(th\), elements with high validity for identification are obtained from the HOG feature. The sparsification threshold \(th\) and the dimensionless coefficient \(a\) are calculated using the following expressions.

\[
th = k \cdot \frac{1}{m} \sum_{h_i \in H} h_i \tag{2}
\]

\[
a = \frac{1}{\sum_{h_i \in H} |h_i|^2} \tag{3}
\]

Coefficient \(k\) in equation (2) is a sparsification rate that is a constant necessary for determining the number of real elements for the number of dimensions of the feature vector. The dimensionless coefficient in equation (3) makes normalization processing unnecessary. Sparse FIND feature in equation (4) is calculated by taking correlation with only elements with high validity for identification, as shown in Fig. 2.

\[
f(h_i, h_j) = a \cdot h_i \cdot h_j \mid h_i \in H_D \land h_j \in H_D \land i \neq j \tag{4}
\]

In that equation, \(H_D\) is defined as \(H_D = \{h_i \mid h_i > th\}\).

Figure 3 [2] is cited as data showing the extraction accuracy of HOG, FIND, and Sparse FIND. The sparsification rate \(k\) is used from 0.5 to 2.0 in 0.5 increments. The Sparse FIND shows better characteristics than those of HOG.

The features of Sparse FIND are fewer than those of FIND, but the problem remains that the total calculation amount is 161 GOPS. The number of processing cycles is also large. Solutions against these issues are described in Section III.

In the FPGA implementation, the processor was designed with \(p=4, d=8, q=2,\) and \(k=1.0\).
Intermediate result RAM." When calculating the block of the next column, the SVM score is read out to the MAC module at the first row of the next column. Finally, when the SVM operation of the block 74 of a certain detection window is performed, processing of the detection window is ended. The intermediate result of the SVM operation of the window is stored in the RAM and is invoked from the RAM at any time when the data of the block belonging to the window is inputted if the calculation of all features in the window is not completed and data of a new block belonging to the window is not input. In this way, high-speed operation is realized by executing the SVM operation at a maximum 75 parallels. Using this method, it became possible to reduce the processing cycle of the SVM operation using HOG feature by 98.6% compared with the case without highly parallel processing.

C. Block-parallel processing for RAM access cycle reduction

The third one is block-parallel processing for RAM access cycle reduction. In SVM operation, to calculate the sum of products with the Sparse FIND feature, the SVM coefficient is accessed and acquired from RAM. The tendency of features excluded from computation by the sparsifying process is random for each block and has no regularity. In the feature calculation for each block and the SVM operation, randomness causes a bias in the number of accesses for each RAM block, so that efficient computation cannot be performed. However, features for two blocks are calculated concurrently to reduce variations in the number of RAM accesses in the proposed scheme, with only a small increase of the logic gate count in the control circuits. An example is portrayed in Fig. 7. When the block A and B are processed consecutively, the access cycle is the sum of the maximum number of RAM accesses of each block: if the maximum access number of the block A is 5 and the maximum number of accesses of the block B is 6, then 11 cycles in all are necessary. The access cycle is the maximum sum of the number of RAM accesses if two blocks A and B are executed simultaneously: 9 cycles are necessary in the case portrayed in Fig. 7. Consequently, the maximum access cycle for two blocks is reduced from the sum of the maximum access cycle of two blocks to the maximum sum of access cycle of 2 block processing. Using this method, it became possible to reduce the average access cycle of SVM coefficient to RAM in the SVM operation using Sparse FIND feature by 28.5%.

B. High parallelization of computing for SVM

The second one is a highly parallel processing of SVM operation. The SVM operation is performed on a block basis. Because the detection window consists of 75 blocks: one block belongs to a maximum of 75 detection windows. The SVM classification circuit is depicted in Fig. 6. The circuit comprises five SVM operation cores and comparators, an SRAM for intermediate classification results, and an SRAM for SVM coefficients. The SVM operation core comprises 15 MAC modules, and carries out multiple accumulation operations for 15 blocks: The MAC module in the SVM classification circuit has 75 parallel configurations (5 cores × 15). This number is equal to the total number of blocks per detection window. Each time the SVM calculation of the block is completed, the SVM score obtained from the block is added cumulatively to the SVM score sent from the MAC module at the preceding row and is transferred to the MAC module at the next row. At that time, the SVM score output from the MAC module at the last row in the "SVM calculation core" is written to "SVM calculation core".
IV. ARCHITECTURE

A. FPGA system

The overall system configuration is presented in Fig. 8. The Sparse FIND processor developed this time is a shadowed portion. The functions of the other blocks are the following.

- External interface: Interface for data communication between PC and FPGA
- Memory interface: Interface for data communication between FPGA and DDR3 memory
- Control register: Register file group for controlling operation of Sparse FIND processor
- Circuit for data transfer control (DMA-Ctrl): Circuit for controlling input / output data with Sparse FIND processor

B. Sparse FIND processor

The architecture of the Sparse FIND processor is presented in Fig. 9. It comprises a common processing block of HOG/Sparse FIND, a HOG processing block and a Sparse FIND processing block. The color utilized in Fig. 9 corresponds to Fig. 4. In the common processing block, the luminance gradient strength and luminance gradient direction are calculated, and a cell histogram and a dimensionless coefficient are obtained. In the HOG processing block, HOG features are calculated and identification using HOG is performed. In the Sparse FIND processing block, Sparse FIND features are calculated for the block identified as an object by HOG. Then classification using Sparse FIND is performed.

V. EVALUATION

To evaluate differences of detection accuracies caused by implementations, with software and with an FPGA, pedestrian detection was performed using actual road environment images captured by an in-vehicle camera during daytime. We used 4,024 test images from the Caltech Dataset [5]. Images were preprocessed to HDTV resolution from VGA resolutions using the Lanczos 3 method. Pyramid images shrunk from an input image were scanned sequentially using a beforehand trained pedestrian classifier. The 24 × 64 pixel scanning window was...
shifted every 4 pixels in each image. The detection accuracy was evaluated using a tool attached to the Caltech Pedestrian Dataset, although the accuracy shown in this manuscript is not comparable with that reported in Caltech Pedestrian Benchmarks [6] because of differences of evaluation conditions such as the resolutions of images and the sliding stride of a scanning window. The FPGA clock frequency was set to 90 MHz in this evaluation.

In this design, the CORDIC method was used for the arc tangent and square root operation in the histogram generation. For the square root division, the NEWTON method was used. The number of steps in the CORDIC method and the NEWTON method was determined by evaluating the influence of the number of steps on accuracy. Figure 10 presents the normalized miss rate at each false positive when the number of CORDIC method steps is changed. Here, the miss rate obtained in the FPGA system was normalized by that in the software implementation using C++ original code. Figure 11 shows the normalized miss rate when the number of NEWTON method steps is changed.

From the viewpoint of average miss rate over Fig. 10, '11' was chosen as the number of steps of the CORDIC method in the hardware implementation. 'Four' was selected as the number of steps of the NEWTON method from Fig. 11.

Table I presents the resource utilization on the FPGA of the hardware circuit developed this time. The tool used to create this table is Vivado 2016.3. Its operating frequency is 90 MHz. Here, BRAM is the memory region allocated initially in FPGA.

Figure 12 presents results of accuracy comparison between C++ original code and FPGA (VU440). The average error of the undetected rate in the section from $10^{-2}$ to $10^0$ is 0.5%, which shows that the FPGA system provides sufficient accuracy for practical use.

Table I presents the resource utilization on the FPGA of the hardware circuit developed this time. The tool used to create this table is Vivado 2016.3. Its operating frequency is 90 MHz. Here, BRAM is the memory region allocated initially in FPGA.

Figure 13 shows the usage ratio of the circuit. The left and right pie charts respectively show the usage ratio of FF and LUT in the FPGA. More than half of FF and LUT are used in SVM classification part of Sparse FIND. In addition, two-step processing by HOG and Sparse FIND generated additional circuits. With this additional circuit implementation, the circuit quantity increases by 21% for FF and 24% for LUT.

Figure 14 shows the frame rate distribution attained at 90 MHz operation when the rejection threshold $\alpha$ for HOG processing results is -0.68. The rejection threshold $\alpha$ was determined when learning was executed by the C++ original code. The frame rate represents the number of frames processed per second during object detection. Frames with a higher frame rate can be processed faster. The average of the frame rate at 90 MHz is 47.63 fps. It also operates at 46.15 fps at the slowest case. Therefore, it is apparent that real-time (30 fps) processing for the HDTV resolution video is possible.
Figure 15 portrays the demonstration system. The PC converts one frame of HDTV image into 25 grayscale pyramidal images and then transfers them to the FPGA board. Conversely, when the object detection is achieved using Sparse FIND feature, the 64 bit data as described in Section A of Chapter IV are generated. The PC detects the end of one image when the EOF flag of the output data enable (10 in binary number). Furthermore, using this output data, an image in which the detection frame is superimposed on the input image like that of Fig. 16 is created.

VI. CONCLUSION

For this study, an FPGA system was implemented as dedicated hardware to execute the human detection algorithm using Sparse FIND feature in real time (30 fps) for HDTV images. Recognition accuracy degradation from the original Sparse FIND-base object detection algorithm implemented on software was 0.5%, which demonstrates that the FPGA system provides sufficient accuracy for practical use. The processing speed was 46.15 fps in the worst case, which satisfied the real-time execution requirement (30 fps) at the operating frequency of 90 MHz.

REFERENCES