IPSJ Transactions on System LSI Design Methodology Vol.6 42–51 (Feb. 2013)

[DOI: 10.2197/ipsjtsldm.6.42]

Regular Paper

An FPGA Implementation of a HOG-based Object Detection Processor

Kosuke Mizuno[1,a) Yosuke Terachi[1] Kenta Takagi[1] Shintaro Izumi[1]
Hiroshi Kawaguchi[1] Masahiko Yoshimoto[1]

Received: May 25, 2012, Revised: August 31, 2012,
Accepted: October 30, 2012, Released: February 15, 2013

Abstract: This paper describes a Histogram of Oriented Gradients (HOG)-based object detection processor. It features a simplified HOG algorithm with cell-based scanning and simultaneous Support Vector Machine (SVM) calculation, cell-based pipeline architecture, and parallelized modules. To evaluate the effectiveness of our approach, the proposed architecture is implemented onto a FPGA prototyping board. Results show that the proposed architecture can generate HOG features and detect objects with 40 MHz for SVGA resolution video (800 × 600 pixels) at 72 frames per second (fps).

Keywords: image recognition, histograms of oriented gradients, support vector machine, hardware implementation

1. Introduction

In recent years, detecting objects in visual images has posed a challenging problem in a wide range of application domains such as surveillance, entertainment, automotive systems, and robotics. A high-accuracy algorithm used in object detection systems, Histogram of Oriented Gradients (HOG) [1], is robust against changes in illumination and also attains high computational accuracy in detection of variously textured objects.

Recent high-performance general-purpose processors can achieve real-time HOG-based object detection in spite of a heavy computational cost. However, these processors suffer from high power consumption and are therefore unsuitable for mobile systems under limited battery conditions. Consequently, a low-power and high-performance HOG feature extraction processor is necessary to widen the range of applications.

Figure 1 presents the image resolution versus frame rate for several published descriptions of HOG hardware. Zhang et al. [2] proposed efficient object detection using GPGPU. Some FPGA implementations [3], [4], [5], [6], [7] and an FPGA-GPU architecture [8] have been proposed for real-time applications. Cao et al. [9] realized the best-performing FPGA implementation when compared with other implementations. However, this study has been specified to stop-sign detection. In order to make HOG algorithm adaptable to a wide variety of applications, next-generation HOG feature extraction processors is required to provide greater expandability and higher performance.

Most conventional processors employ a window-based approach. For the window-based approach, an amount of required computations of 89.2 GOPS and memory bandwidth of 10.9 Gbps are required for SVGA resolution because of repetitive computations. The amount of required computations and memory bandwidth are greatly reduced by reusing calculated data or adopting efficient computation. However, data reuse causes an increase of memory capacity and circuit area. Consequently, a cooperative design among algorithm, architecture, and circuit is necessary.

To achieve real-time and low-power HOG feature extraction for SVGA resolution video, we propose the following three techniques.

- A simplified HOG algorithm with cell-based scanning and simultaneous Support Vector Machine (SVM) calculation to reduce amount of required computations.
- A cell-based algorithm and architecture for memory bandwidth reduction.
- Parallelized architectures for cell histogram generation, histogram normalization, and SVM classification to reduce the necessary cycle count.

As described in this paper, details of the simplified HOG al-

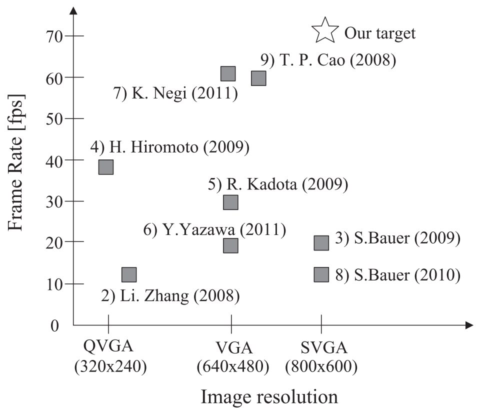


Fig. 1 Previous works of HOG feature extraction processor.

1 Graduate School of Engineering, Kobe University, Kobe, Hyogo 657–8501, Japan
a) mi-no@cs28.cs.kobe-u.ac.jp

gorithm are described in Section 2. The proposed architecture is addressed in Section 3. Then, these are followed by FPGA implementation in Section 4. Section 5 concludes this paper.

2. Algorithm

2.1 Original HOG Algorithm

Figure 2 portrays a flow diagram of object detection using the original HOG algorithm [1]. Scanning on the input image is based on the detection window. The window is divided into cells, and each cell accumulates a histogram of gradient orientations over the pixels of the cell. For better invariance to illumination, histogram normalization can be done by accumulating a measure of the local histogram energy over blocks and using the results to normalize all cells in the block. The normalized histograms (HOG features) are collected over the detection window. The collected features are fed to a linear SVM for object/non-object classification.

2.2 Simplified HOG Algorithm for Hardware Implementation

A simplified HOG algorithm for hardware implementation is

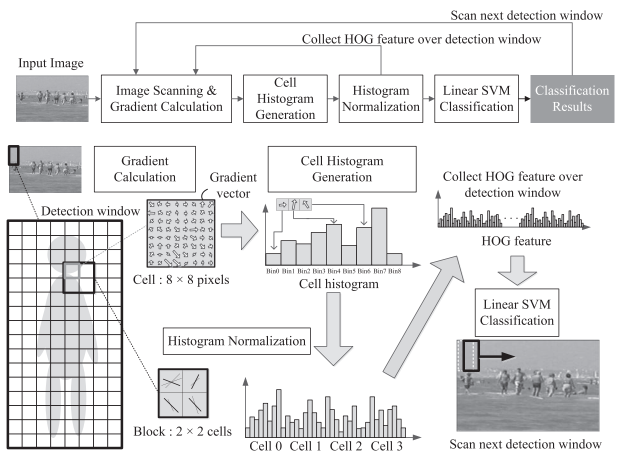

Scan next detection window
Collect HOG feature over detection window
Input Image
Image Scanning & Gradient Calculation
Cell Histogram Generation
Histogram Normalization
Linear SVM Classification
Classification Results
Gradient Calculation
Gradient vector
Cell Histogram Generation
Collect HOG feature over detection window
Detection window
HOG feature
Cell : 8 × 8 pixels
Bin0 Bin1 Bin2 Bin3 Bin4 Bin5 Bin6 Bin7 Bin8
Cell histogram
Linear SVM Classification
Histogram Normalization
Block : 2 × 2 cells
Cell 0 Cell 1 Cell 2 Cell 3
Scan next detection window

Fig. 2 Original HOG algorithm flow.

introduced in this subsection. **Figure 3** shows a flow diagram of object detection using the simplified HOG algorithm. This flow is modified from the original flow using the following six techniques.

1. Cell-based scanning (Section 2.2.1)
2. Gradient calculation using CORDIC [10]
3. Approximation of weighted voting for spatial and orientation anti-aliasing (Section 2.2.2)
4. Newton method with approximated initial value (Section 2.2.3)
5. Simultaneous SVM calculation (Section 2.2.4)
6. Parameter optimization (Section 2.3)

Figure 4 portrays the amount of required computations for HOG-based object detection. Assuming the window-based approach, an amount of required computations of 89.2 GOPS is necessary for SVGA resolution because of repetitive computations. On the other hand, the simplified HOG algorithm with cell-based scanning and simultaneous SVM calculation reduces the amount of required computations to 2.25 GOPS. However, 2.25 GOPS is still heavy for a processor with low operating frequency. To accommodate the amount of required computations in real time, the proposed architecture has parallelized modules for cell histogram generation, histogram normalization, and SVM classification.

2.2.1 Cell-based Scanning Method

Object detection with HOG features is executed by the scanning detection window on an input image, as presented in **Fig. 5** left. When one window is finished, the next window is scanned using an offset of 1 cell. The memory bandwidth is increased by reloading input pixels for the next window. Consequently, extensive data reuse is desirable for memory bandwidth reduction.

Figure 5 right shows a cell-based scanning approach. HOG features are extracted from cell-based calculations. No cell overlaps with another cell. Consequently, sharing and reuse of a cell have a great impact on memory bandwidth reduction.

2.2.2 Anti-aliasing in Histogram Generation

Gradient vectors within a local region called a cell (8 × 8 pix-

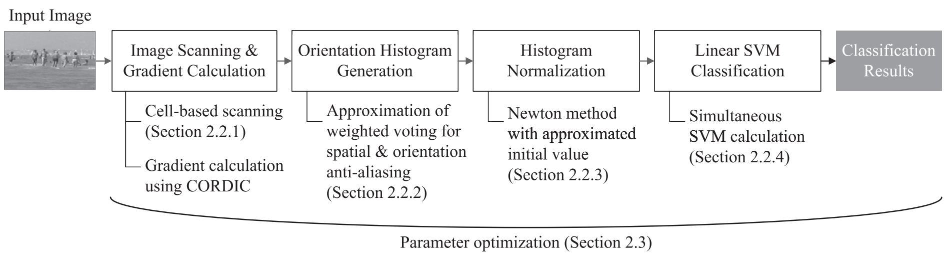
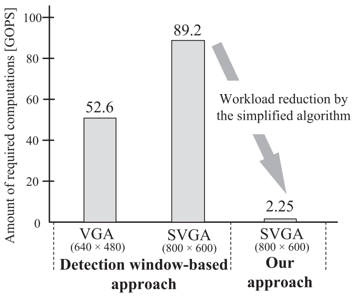

Input Image
Image Scanning & Gradient Calculation
Orientation Histogram Generation
Histogram Normalization
Linear SVM Classification
Classification Results
Cell-based scanning (Section 2.2.1)
Gradient calculation using CORDIC
Approximation of weighted voting for spatial & orientation anti-aliasing (Section 2.2.2)
Newton method with approximated initial value (Section 2.2.3)
Simultaneous SVM calculation (Section 2.2.4)
Parameter optimization (Section 2.3)

Fig. 3 Simplified HOG algorithm flow.



Detection window-based approach

Gradient calc. + Magnitude & Orientation calc. + Histogram generation
+ Histogram normalization + Classification
= 2.7 + (46.6 + 31.5) + 1.1 + 6.0 + 1.3
= 89.2 [GOPs]

Our approach

Gradient calc. + Magnitude & Orientation calc. + Histogram generation
+ Histogram normalization + Classification
= 0.03 + (0.49 + 0.33) + 0.06 + 0.08 + 1.26
= 2.25 [GOPs]

Fig. 4 Amount of required computations.

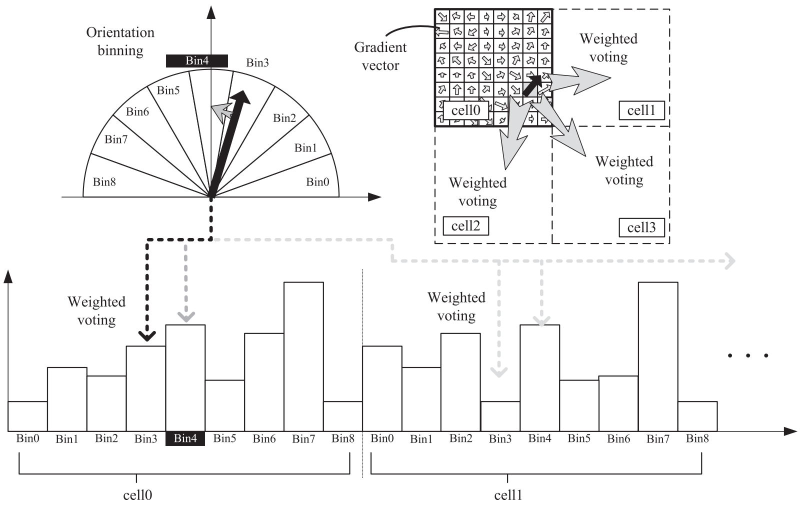


Fig. 6 Anti-aliasing for spatial/orientation bin.

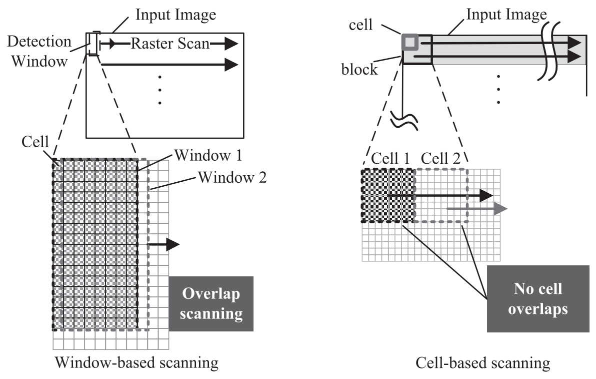


Fig. 5 Image scanning methods.

els) are classified into nine bins according to its orientation, as shown in **Fig. 6**. A cell histogram is generated by adding the magnitude of the gradient vector into the corresponding bin. To prevent aliasing from occurring around the border between each of the bins, the original algorithm votes the magnitude into the corresponding bin and the adjacent bin. The magnitude of the adjacent bin is in proportion to the orientation. However, the proposed algorithm adopts a fixed coefficient expressed as a power of two for the magnitude calculation. The proposed method suppresses a hardware cost increase.

Another type of aliasing occurs around the borders between each of the cells. Strong edges around the borders between each of the cells degrade the accuracy of feature vectors. To avoid the phenomenon described above, gradient vectors in a cell are voted into other cells. The original algorithm adopts a bi-linear interpolation method and Gaussian weighting method for calculating weighted magnitude. However, the proposed algorithm uses a coefficient table because the weighting coefficient at one location is a fixed value. To reduce multiplication for weighting, the weighting coefficients are approximated to bit shift operation.

2.2.3 Histogram Normalization

The cell histograms within a region called a block (2 × 2 cells) are concatenated and normalized. The normalization is performed by dividing a concatenated cell histogram v by its L2 norm. The values of normalized histograms are clipped to a limit number and then re-normalized, which requires square root calculation and division. To avoid these burdensome calculations, the inverse of normalization divisor d is approximated as

$$d = \frac{1}{\sqrt{\|v\|_2^2 + \varepsilon^2}} \tag{1}$$

then d is multiplied by v, where ε is a small constant to avoid zero division.

To obtain d, an approximation method with bit shift operation is proposed in Ref. [6]. However, this simplification is unsuitable for re-normalization and causes normalization errors. Therefore, we take a more accurate approach with additional iterative calculations. The inverse d is calculated iteratively using the Newton method as follows.

$$d_{i+1} = d_i \cdot (3 - d_i^2 \cdot \|v\|_2^2)/2 \tag{2}$$

One iteration is composed of three multiplications, a subtraction and a bit shift. Although this iteration converges quadratically, the initial value d_0 is important to reduce the number of iterations. To obtain d_0, we use an approximation method proposed in an earlier report [6]. If the sum of squares of v satisfies the following inequality,

$$2^n - \varepsilon^2 \leq \|v\|_2^2 < 2^{n+1} - \varepsilon^2 \tag{3}$$

then d_0 is given as follows.

$$d_0 = 2^{-\frac{n+1}{2}} \tag{4}$$

d_0 is expressed as a binary number instantly if n is an odd number. d_0 is given by multiplying additionally if n is an even number.

2.2.4 Simultaneous SVM Calculation

In the window-based approach, HOG features from 105 blocks are collected. Then the features are multiplied by SVM coefficients corresponding to one window. However, the cell-based approach provides partial HOG features after normalization for one block; after which the features are multiplied by SVM coefficients corresponding to the 105 windows.

Figure 7 presents simultaneous SVM calculations for cell-based processing. Partial HOG features belong to 105 windows maximally and are located at different positions in each window.

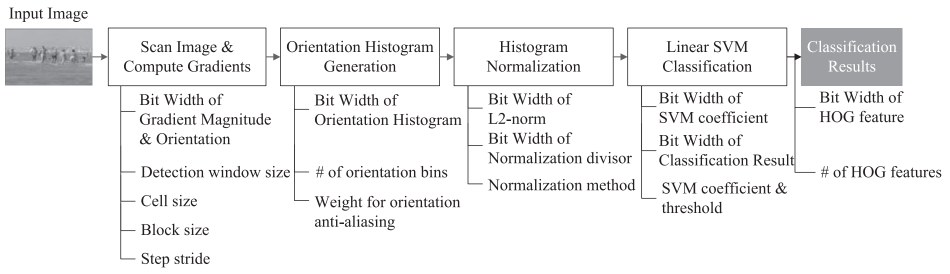

Input Image
Scan Image & Compute Gradients
Orientation Histogram Generation
Histogram Normalization
Linear SVM Classification
Classification Results
Bit Width of Gradient Magnitude & Orientation
Detection window size
Cell size
Block size
Step stride
Bit Width of Orientation Histogram
# of orientation bins
Weight for orientation anti-aliasing
Bit Width of L2-norm
Bit Width of Normalization divisor
Normalization method
Bit Width of SVM coefficient
Bit Width of Classification Result
SVM coefficient & threshold
Bit Width of HOG feature
# of HOG features

Fig. 8 HOG algorithm parameters.

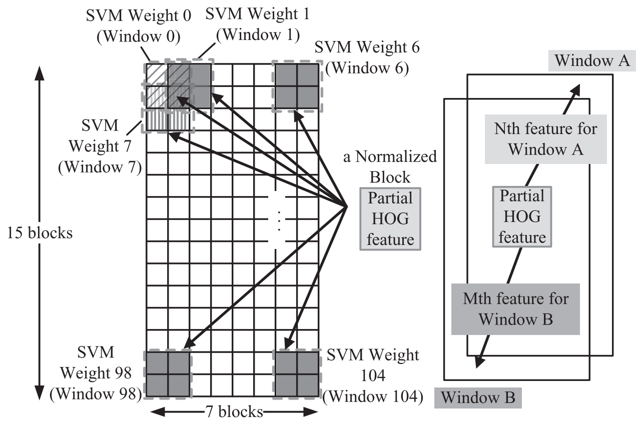


Fig. 7 Simultaneous SVM classification.

Table 1 Optimized bit width.

Parameter	Bit width [bit]		
	Sign	Integer	Fractional
Gradient magnitude	1	9	0
Gradient orientation	1	3	3
Orientation histogram	0	11	0
1st L2-norm	0	25	0
1st normalization divisor	0	0	11
2nd L2-norm	0	0	14
2nd normalization divisor	0	3	7
HOG feature	0	0	4
SVM coefficient	1	3	7
Classification buffer	1	4	10

Partial HOG features are multiplied and accumulated by the SVM coefficients of each window. The accumulation result is stored and reused in the subsequent SVM calculation. Simultaneous SVM calculation is suitable for parallel computing in hardware.

2.3 Parameter Optimization

Figure 8 shows the parameters of each process in object detection using HOG features. In general, software implementations employ floating-point calculations to provide high accuracy. On the other hand, if we implement hardware floating-point unit, it requires hardware resources to a great degree. Therefore, fixed-point operation is often used in hardware implementations. The accuracy of a fixed-point operation depends on the bit width itself, although the bit width affects the memory capacity and the circuit area. Optimized parameters provide reasonable classification accuracy and minimize hardware costs. **Table 1** presents the results of parameter adjustment. The other parameters for the HOG algorithm are shown in **Table 2**.

Table 2 Other parameters.

Detection window size	64 × 128 pixels
Cell size	8 × 8 pixels
Block size	2 × 2 cells
Step stride	8 pixels, vertically and horizontally
# of orientation bins	9 bins (0-180)
Weight for orientation anti-aliasing	0.25
Normalization method	L2-norm
SVM coefficients and threshold	Default value in OpenCV
# of HOG features	3780 = (7 × 15 blocks) • (2 × 2 cells) • (9 bins)

2.4 Simulation Results

Performance and accuracy degradation were evaluated by software simulation for the simplified algorithm. The test data used was INRIA Person Dataset [11] which includes several people in various backgrounds. The simplified algorithm is compared with the original linear rectangular HOG (Lin. R-HOG) [1]. **Figure 9** presents a graph of false positives per window (FPPW) versus the miss rate. In Fig. 9, simulation results with the original Lin. R-HOG show higher miss rate in comparison with that in the Ref. [1] owing to the difference of test condition including the number of test samples. The simulation results with the simplified algorithm and the optimized bit width show that the miss rate degradation is 3% at 0.0001 FPPW. It provides sufficient performance for general-purpose applications.

3. Architecture

3.1 Cell-based Pipeline Architecture

Figure 10 depicts a block diagram of the cell-based pipeline architecture and external peripherals for the demonstration system detailed in Section 4. The proposed architecture comprises a controller, a cell histogram generation module, a histogram normalization module, an SVM classification module, SRAMs for several image data, a CPU interface, and a memory interface. The HOG feature extraction processor is controlled by an external CPU, and the input grayscale image is loaded to a cell-line buffer from an external SRAM via a memory interface. The internal datapath modules process the input image and output the detection result. The CPU receives the detection result from the HOG feature extraction processor and then draws the result on an LCD display.

The proposed architecture adopts a cell-based pipeline flow, as presented in **Fig. 11**. Figure 11 above shows a relation between cells, blocks, windows, and the frame. Cell-based pipeline processing is conducted as follows:

1. A cell histogram is generated with cell-based scanning.
2. When the process described above reaches the block level, the block-level cell histogram is normalized and the block-

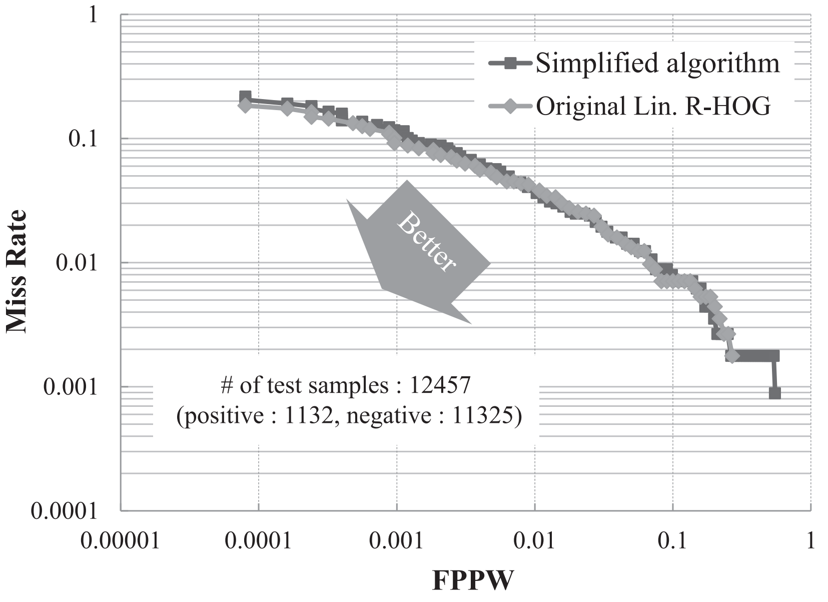


Fig. 9 Accuracy degradation by the simplified algorithm.

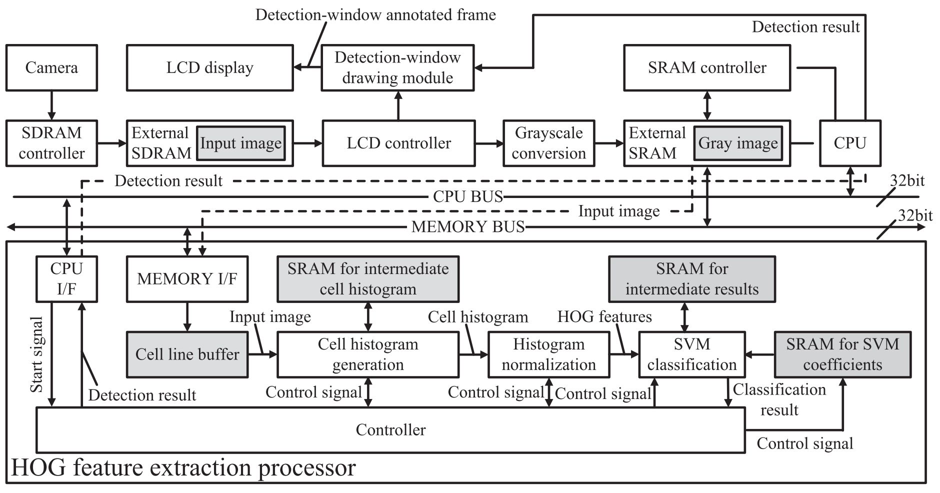


Fig. 10 HOG feature extraction architecture.

level HOG feature is extracted.
3. Block-level HOG features and SVM coefficients corresponding to each window are multiplied and accumulated.
4. The accumulation result of the window level is compared with the SVM threshold. Then the detection result is obtained.

The cell-based pipeline architecture greatly reduces the memory bandwidth because it prevents reloading of input pixels in different detection windows.

Figure 12 describes the memory bandwidth analysis of HOG-based object detection. The window-based approach for SVGA resolution requires a memory bandwidth of 10.9 Gbps. In general, mobile systems under limited battery conditions adopt a lower operating frequency. Therefore, the memory bandwidth must be reduced as low as possible for low-power and real-time operation. Our approach adopts a cell-based algorithm and architecture to reduce the memory bandwidth to 0.116 Gbps.

3.2 Cell Histogram Generation

In cell histogram generation, magnitude and orientation of a given pixel gradient are calculated; then a weighted magnitude is voted into a bin corresponding to its orientation. **Figure 13** presents the architecture for cell histogram generation. Four-way architecture is adopted because one cell is commonly used for four blocks maximally. Block 0 and 2 load an initial value from the neighboring block. Block 1 loads an initial value from SRAM. Block 3 loads zero for its initial value. When cell histogram generation is finished, Block 0 outputs a cell histogram to the histogram normalization module. **Figure 14** illustrates a block diagram of the processing element (PE). One PE executes weighted voting and binning to generate a histogram of one cell. Spatial anti-aliasing is conducted in four processing elements corresponding to one block.

3.3 Histogram Normalization

Figure 15 presents the architecture for histogram normalization. The architecture consists of two stages to implement L2-Hys normalization. The first stage includes four Cell MAC modules, an approximation module, a Newton method module, and a threshold module. The second stage comprises four Cell MAC

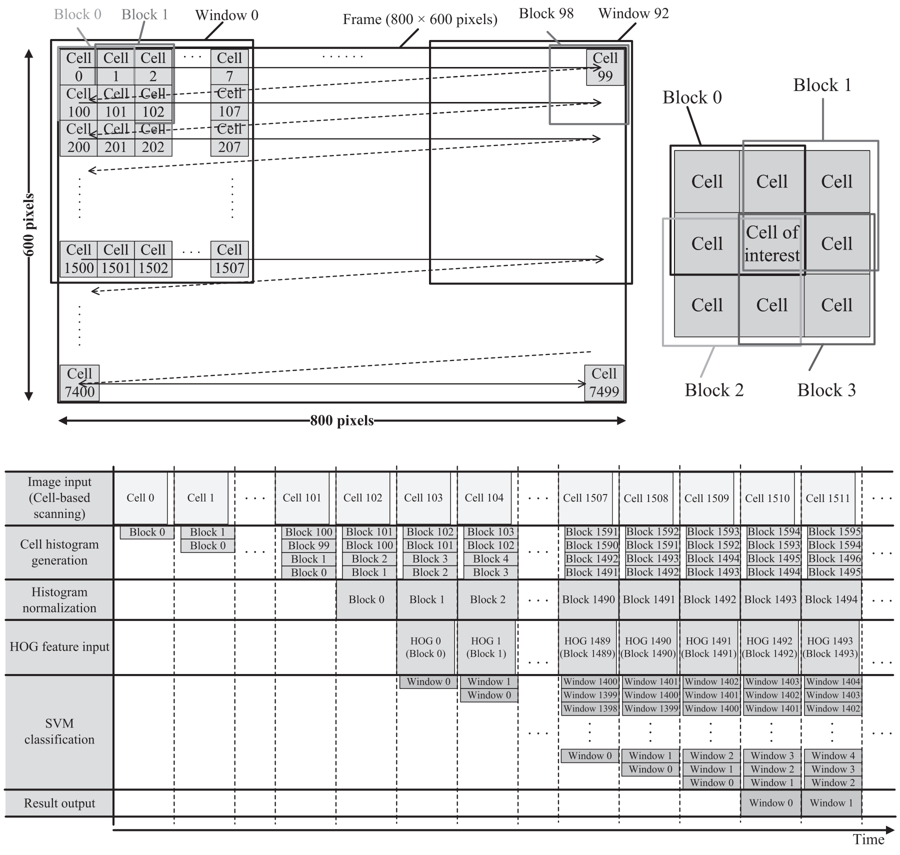


Fig. 11 Cell-based pipeline flow.

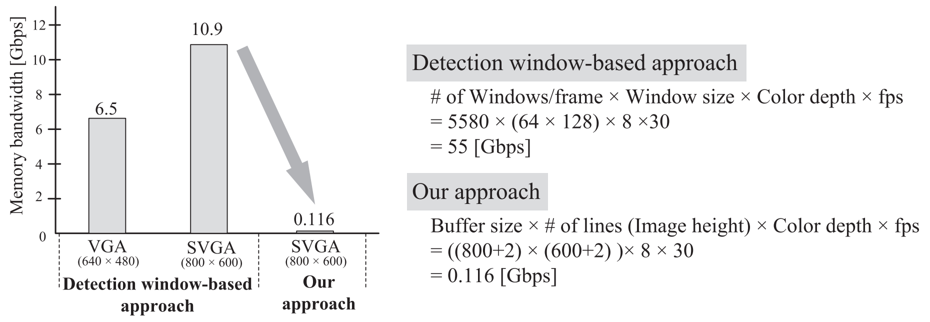


Fig. 12 Memory bandwidth analysis.

modules and a Newton method module.

In the first stage, Cell MAC modules first calculate the sum of squares of the input cell histogram. Secondly, an initial value for the Newton method is approximated to bit shift operation. Thirdly, the Newton method calculates an inverse number of square roots. Furthermore, the Cell MAC modules normalize the cell histogram. Finally, the normalized cell histogram is compared with a threshold and is fed to the second stage.

In the second stage, the sum of squares and an inverse number of square roots are calculated as in the first stage, after which Cell MAC modules normalize the cell histogram and extract 36-dimension HOG features.

3.4 SVM Classification

In SVM classification, extracted features and SVM coefficients are multiplied and accumulated until the operations reach window level. Then the accumulation result is compared with an SVM threshold to judge whether the window includes a target object. **Figure 16** shows a block diagram for simultaneous SVM classification. This architecture includes 15 classification cores. One classification core manages seven blocks of MAC operations. Consequently, the architecture can accommodate 105 blocks corresponding to one detection window. Sufficient parallelism reduces the required cycle count to manage the amount of required computations of 10.6 GOPS.

3.5 Performance Evaluation

The number of cycle counts was estimated using a Verilog-HDL simulator. The proposed architecture was compared with architecture without parallelization and without a pipeline. Estimation results are presented in **Fig. 17**, which demonstrates the superiority of the proposed architecture for SVGA resolution. The parallelization in the cell histogram generation and histogram normalization contributes to a reduction in cycle counts. Intro-

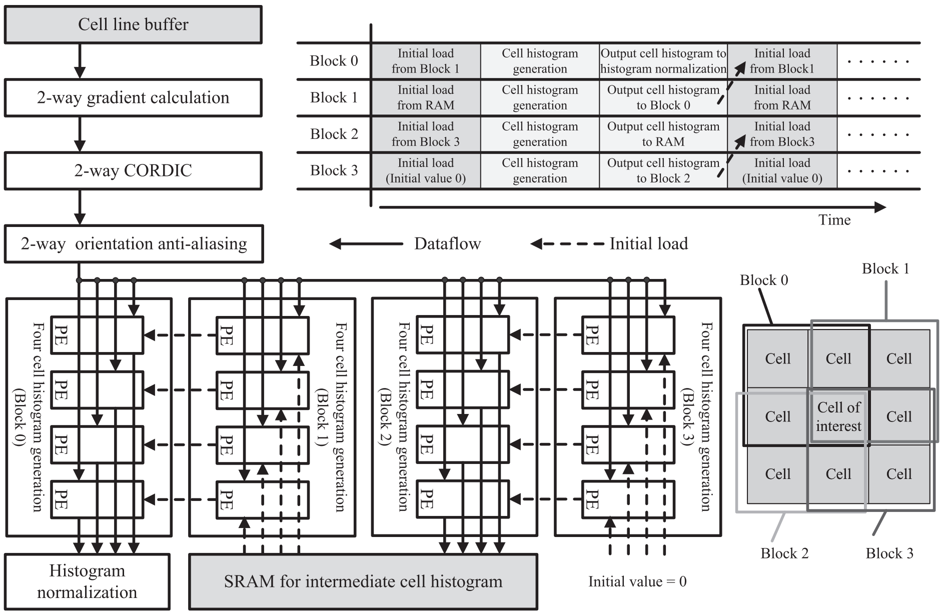

Cell line buffer
2-way gradient calculation
2-way CORDIC
2-way orientation anti-aliasing
Block 0
Block 1
Block 2
Block 3
Initial load from Block 1
Initial load from RAM
Initial load from Block 3
Initial load (Initial value 0)
Cell histogram generation
Output cell histogram to histogram normalization
Output cell histogram to Block 0
Output cell histogram to RAM
Output cell histogram to Block 2
Initial load from Block1
Initial load from RAM
Initial load from Block3
Initial load (Initial value 0)
Time
Dataflow
Initial load
PE
Four cell histogram generation (Block 0)
Four cell histogram generation (Block 1)
Four cell histogram generation (Block 2)
Four cell histogram generation (Block 3)
Histogram normalization
SRAM for intermediate cell histogram
Initial value = 0
Cell
Cell of interest

Fig. 13 Block diagram and processing flow of cell histogram generation.

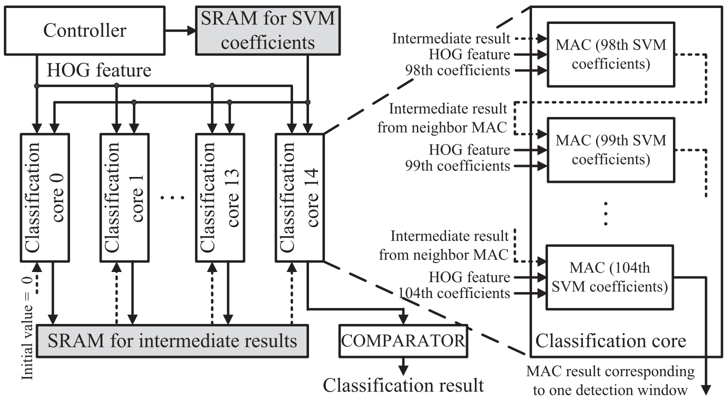


Fig. 16 Block diagram of SVM classification module.

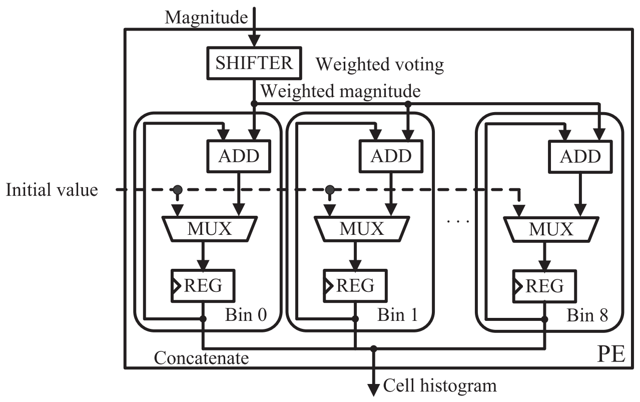


Fig. 14 Block diagram of a processing element.

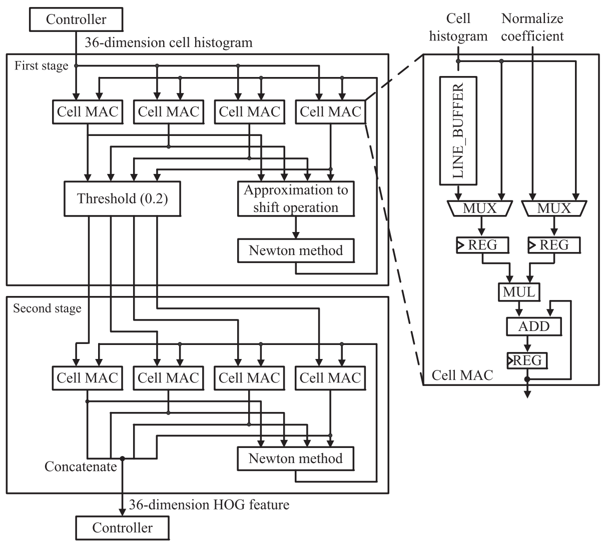


Fig. 15 Block diagram of histogram normalization.

duction of the proposed simultaneous SVM calculation architecture enables the reuse of intermediate results, allowing further cycle count reduction. Results show that the number of cycle counts in cell histogram generation, histogram normalization, and SVM classification are reduced by 85%, 65%, and 99%, respectively, compared with the number of cycle counts of architecture without parallelization and without a pipeline. In the proposed architecture, the overall process requires 0.56×10^6 cycles per frame. Therefore, it is inferred that the proposed architecture can accommodate SVGA resolution video at 72 fps with 40 MHz.

The proposed architecture shows superior performance in comparison with previous FPGA implementations [3], [4], [5], [6],

Table 3 FPGA Implementation Results.

	[3]	[4]	[5]	[6]	[7]	[9]	Ours
FPGA	Spartan 3	Virtex-5	Stratix II	Cyclone III	Virtex-5	Virtex-4	Cyclone IV
# of LUTs	42,435	28,495	37,940	34,838	17,383	8,921	34,403
# of registers	N/A	5,980	66,990	22,612	2,181	4,221	23,247
# of DSP blocks	18	2	120	N/A	N/A	3	68
Working memory (Mbits)	1.08	2.196	N/A	2.094	1.296	1.584	0.34
Resolution	800×600	320×240	640×480	640×480	640×480	752×480	800×600
Frame rate (fps)	20	38	30	20	62.5	60	72
Operating frequency (MHz)	63	167	127.49	70	44.85	N/A	40
Image scanning method in HOG feature extraction	Window-based				Cell-based	Window-based	Cell-based
Image scanning method in classification	Window-based						

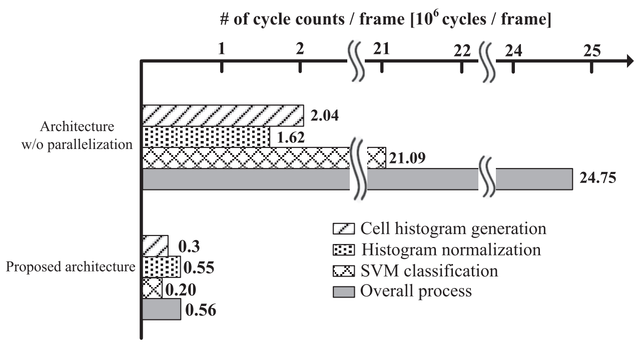


Fig. 17 Reduction of cycle count.

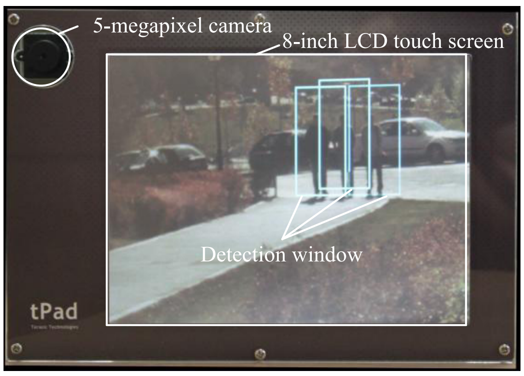


Fig. 18 Architecture verification by FPGA implementation.

[7], [9]. Working memory is reduced by 68%, 84%, 83%, 73% and 78%, respectively, compared with that in Refs. [3], [4], [6], [7] and [9]. Therefore, the proposed architecture is implementable even on FPGA with limited block memory. The proposed architecture can process in parallel about ten times as many detection windows as the architecture in Ref. [5], and it can process about fifty times as many detection windows as the architecture in Ref. [6]. Though the cell-base method for HOG feature extraction was taken in Ref. [7], the architecture in Ref. [7] requires larger memory than our approach. The classification module in Ref. [7] adopts no cell-based approach causing overhead of processing speed.

4. FPGA Implementation

To evaluate the effectiveness of our approach, we implemented the proposed architecture onto a prototyping board (tPad Multimedia Development Kit; Terasic Technologies Inc.). The board has DE2-115 with Cyclone IV EP4CE115 (Altera Corp.), a 5-megapixel digital image sensor module, and an 8-inch LCD touch screen module. **Figure 18** portrays a demonstration system of real-time object detection to verify the proposed technique.

Resource utilization and comparison to conventional FPGA implementations are presented in **Table 3**. Our FPGA implementation can generate HOG features and detect objects with 40 MHz for SVGA resolution video at 72 fps. The FPGA resource utilizations are as follows: 34,403 LEs, 68 embedded multipliers, and 0.34 Mbit block RAMs. Our implementation shows the best performance with minimum memory usage and minimum operating frequency.

5. Conclusion

This paper presents a proposal of a novel architecture of real-time HOG feature extraction for SVGA resolution video. The proposed scheme has a simplified HOG algorithm with cell-based scanning, simultaneous SVM calculation, cell-based pipeline architecture, and parallelized modules. The simplified algorithm contributes to a reduction of the amount of required computations from 89.2 GOPS to 2.25 GOPS with 3% accuracy degradation. The cell-based algorithm and pipeline architecture provide a memory bandwidth of 0.116 Gbps at SVGA resolution, which can be handled by a 32-bit memory bus with a reasonably low operating frequency. Parallelized modules greatly accelerate HOG feature extraction and object detection. The proposed architecture on a FPGA prototyping board showed the best performance with minimum memory usage and minimum operating frequency when compared with the performance of conventional processors.

Acknowledgments This work was supported by VLSI Design and Education Center (VDEC), The University of Tokyo, in collaboration with Cadence Design Systems, Mentor Graphics Corp. and Synopsys, Inc.

References

[1] Dalal, N. and Triggs, B.: Histograms of Oriented Gradients for Human Detection, *Proc. 2005 International Conference on Computer Vision and Pattern Recognition*, Vol.2, pp.886–893, Washington, DC, USA: IEEE Computer Society (2005).

[2] Zhang, L. and Nevatia, R.: Efficient Scan-Window Based Object Detection using GPGPU, IEEE, CVPRW (2008).

[3] Bauer, S., Brusmann, U. and Schlotterbeck-Macht, S.: FPGA Implementation of a HOG-based Pedestrian Recognition System, *MPC-Workshop* (July, 2009).

[4] Hiromoto, M. and Miyamoto, R.: Hardware Architecture for High-Accuracy Real-Time Pedestrian Detection with CoHOG Features, *IEEE ICCVW* (2009).

[5] Kadota, R., Sugano, H., Hiromoto, M., Ochi, H., Miyamoto, R. and

Nakamura, Y.: Hardware Architecture for HOG Feature Extraction, *Proc. 2009 International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, pp.1330–1333, Washington DC, USA: IEEE Computer Society (2009).

[6] Yazawa, Y., Yoshimi, T., Tsuzuki, T., Dohi, T. and Fujiyoshi, H.: FPGA Hardware with Target-Reconfigurable Object Detector by Joint-HOG, *Proc. SSII*, Yokohama, Japan (2011).

[7] Negi, K., Dohi, K., Shibata, Y. and Oguri, K.: Deep pipelined one-chip FPGA implementation of a real-time image-based human detection algorithm, *IEEE FPT* (2011).

[8] Bauer, S., Kohler, S., Doll, K. and Brunsmann, U.: FPGA-GPU Architecture for Kernel SVM Pedestrian Detection, *IEEE CVPRW* (2010).

[9] Cao, T.P. and Deng, G.: Real-Time Vision-Based Stop Sign Detection System on FPGA, *Proc. Digital Image Computing: Techniques and Applications*, pp.465–471, Los Alamitos, CA, USA: IEEE Computer Society (2008).

[10] Volder, J.E.: The CORDIC Trigonometric Computing Technique, *IRE Trans. Electron. Comput.*, EC-8:330-334 (1959).

[11] INRIA Person Dataset, available from ⟨http://pascal.inrialpes.fr/data/human/⟩.

Kosuke Mizuno received his B.E. and M.E. degrees in Computer Science and Systems Engineering from Kobe University, Kobe, Japan in 2008 and 2010, respectively, where he is currently pursuing a Ph.D. degree in Engineering. His current research interests include high-performance and low-power multimedia VLSI designs. He is a student member of IEICE and IEEE.

Yosuke Terachi received his B.E. and M.E. degrees in Computer Science and Systems Engineering from Kobe University, Kobe, Japan in 2010 and 2012, respectively. Since 2009, he has been involved in the research and development of low-power image recognition VLSI designs.

Kenta Takagi received his B.E. degree in Computer Science and Systems Engineering from Kobe University, Kobe, Japan in 2012. He is currently on the master course at Kobe University. Since 2011, he has been involved in the research and development of low-power image recognition VLSI designs.

Shintaro Izumi respectively received his B.Eng. and M.Eng. degrees in Computer Science and Systems Engineering from Kobe University, Hyogo, Japan, in 2007 and 2008. He received his Ph.D. degree in Engineering from Kobe University in 2011. He was a JSPS research fellow at Kobe University from 2009 to 2011. Since 2011, he has been an Assistant Professor in the Organization of Advanced Science and Technology at Kobe University. His current research interests include biomedical signal processing, communication protocols, low-power VLSI design, and sensor networks. He is a member of IEEE, IEICE, and IPSJ.

Hiroshi Kawaguchi received his B.Eng. and M.Eng. degrees in Electronic Engineering from Chiba University, Chiba, Japan, in 1991 and 1993, respectively, and earned a Ph.D. degree in Electronic Engineering from the University of Tokyo, Tokyo, Japan, in 2006. He joined Konami Corporation, Kobe, Japan, in 1993, where he developed arcade entertainment systems. He moved to the Institute of Industrial Science, the University of Tokyo, as a Technical Associate in 1996, and was appointed as a Research Associate in 2003. In 2005, he moved to Kobe University, Kobe, Japan. Since 2007, he has been an Associate Professor with the Department of Information Science at that university. He is also a Collaborative Researcher with the Institute of Industrial Science, the University of Tokyo. His current research interests include low-voltage SRAM, RF circuits, and ubiquitous sensor networks. Dr. Kawaguchi was a recipient of the IEEE ISSCC 2004 Takuo Sugano Outstanding Paper Award and the IEEE Kansai Section 2006 Gold Award. He has served as a Design and Implementation of Signal Processing Systems (DISPS) Technical Committee Member for IEEE Signal Processing Society, as a Program Committee Member for IEEE Custom Integrated Circuits Conference (CICC) and IEEE Symposium on Low-Power and High-Speed Chips (COOL Chips), and as an Associate Editor of IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences and IPSJ Transactions on System LSI Design Methodology (TSLDM). He is a member of IEEE, ACM, IEICE, and IPSJ.

Masahiko Yoshimoto received his B.S. degree in Electronic Engineering from Nagoya Institute of Technology, Nagoya, Japan, in 1975, and M.S. degree in Electronic Engineering from Nagoya University, Nagoya, Japan, in 1977. He received a Ph.D. degree in Electrical Engineering from Nagoya University, Nagoya, Japan in 1998. He joined the LSI Laboratory, Mitsubishi Electric Corp., Itami, Japan, in April 1977. From 1978 to 1983 he was engaged in the design of NMOS and CMOS static RAM including a 64 K full CMOS RAM with the world's first divided-wordline structure. From 1984, he was involved in research and development of multimedia ULSI systems for digital broadcasting and digital communication systems based on MPEG2 and MPEG4 Codec LSI core technology. Since 2000, he has been a Professor of the Department of Electrical and Electronic Systems Engineering at Kanazawa University, Japan. Since 2004, he has been a Professor of the Department of Computer and Systems Engineering at Kobe University, Japan. His current activity is focused on research and development of multimedia and ubiquitous media VLSI systems including an ultra-low-power image compression processor and a low power wireless interface circuit. He holds 70 registered patents. He served on the Program Committee of the IEEE International Solid State Circuit Conference from 1991 to 1993. In addition, he has served as a Guest Editor for special issues on Low-Power System LSI, IP, and Related Technologies of IEICE Transactions in 2004. He received the R&D100 awards from R&D Magazine for development of the DISP and development of a real-time MPEG2 video encoder chipset in 1990 and 1996, respectively.

(Recommended by Associate Editor: *Takashi Miyamori*)