Hardware Development of Edge-Preserving Bubble Image Conversion in High-level Synthesis

Jiang Qin

Akira Yamawaki

Kyushu Institute of Technology
1-1 Sensui, Tobata, Kitakyushu 804-8550, Japan
E-mail: qin.jiang816@mail.kyutech.jp, yama@ecs.kyutech.ac.jp

Abstract

The non-photorealistic rendering, NPR, is widely used in social networking service on the mobile device. To realize a real time NPR with low power making battery life of mobile device longer, we attempt to develop hardware module by using high-level synthesis, HLS, converting software to hardware automatically. This research focuses on edge-preserving bubble image, EPB, converting photos into image like filled with bubbles. We proposed a software description method for EPB algorithm so that HLS can generate a high-performance and low-power hardware module. Through the practical experiments, we show that our proposed description method can make HLS generate good hardware module improving the performance and power efficiency compared with the conventional method.

Keywords: Non-photorealistic, rendering, hardware, high-level synthesis, buffer, FPGA.

1. Introduction

In recent years the opportunity to use non-photorealistic rendering into a wide variety of images in social networking services has been increasing. The global market for embedded image processing systems has been expanding rapidly due to the development of such social networking services and measures taken by the Corona disaster to remotely control them and is expected to continue to increase in the future. In order to gain a large market share for embedded image processing systems, it is necessary to develop high-performance and power-efficient image processing system products and introduce them to the market as swiftly as possible.

There are two methods of realizing image processing: hardware and software. Hardware-based processing is desirable because of its higher performance and lower power consumption. However, hardware processing has the disadvantage of a long development period and a large design burden.

In order to solve this problem, it is essential to develop a software description method that can generate efficient hardware by using a technology called High-Level Synthesis (HLS)[1-4], which automatically converts software into hardware.

In this work, we focus on the edge-preserving bubble image, EPB, converting photos into image like filled with bubbles. The EPB is one of non-photorealistic image rendering processes[5], which are conventionally processed by software. We aim to develop a hardware image processing system that is faster than conventional methods by using HLS. This paper develops a software description method so that HLS can generate good hardware module.

2. Edge-preserving Bubble Image Conversion

Edge-preserving bubble transformation is a method to generate an edge-preserving bubble image called EPB (Edge-Preserving Bubble). Fig. 1 shows the overview of this algorithm.

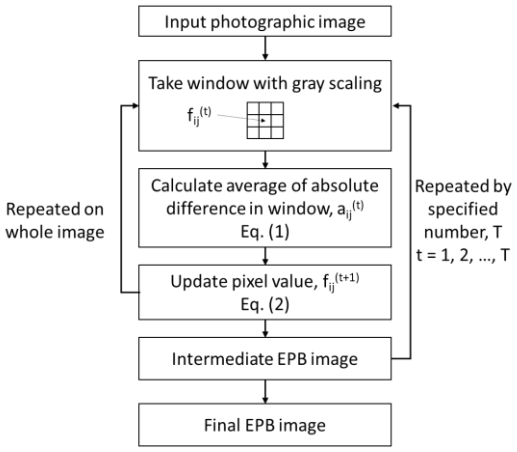

Fig. 1 EPB Algorithm Overview

This algorithm takes a window on the input image as well as spatial filters like mean, sobel, gaussian, and so on. The average absolute brightness difference between the opposite pixels of the processed image corresponds to the edge of the target window, and the transformation of the bubble image can be realized by repeatedly increasing or decreasing the brightness value of the edge equivalent part.

The EPB algorithm is implemented by an iterative process consisting of two steps. The first step is to compute the average absolute difference $a_{i,j}^{(t)}$ in the window by applying the target window of the processed image to Eq. (1).

$$a_{i,j}^{(t)} = \frac{\sum_{k=-1}^{1}\sum_{l=-1}^{1}|f_{i+k,j+l}^{(t)} - f_{i-k,j-l}^{(t)}|}{(2W+1)^2 - 1} \qquad (1)$$

Where, all images to be processed in this study are grayscale images. The $f_{i,j}^{(t)}$ is the luminance value of pixel (i,j), $t(=1,2\ldots)$ is the number of iterations. The k and l are the positions in the window. Then in step 2, all pixels in the process image are update according to Eq. (2).

$$f_{i,j}^{(t+1)} = \begin{cases} f_{i,j} - ba_{i,j}^{(t)} & (t\%2 = 0) \\ f_{i,j} - ba_{i,j}^{(t)} & (t\%2 = 1) \end{cases} \qquad (2)$$

Where, the b is a positive constant, and % is the remainder operator.

If $f_{i,j}^{(t+1)}$ is less than 0, $f_{i,j}^{(t+1)}$ must be set to $-f_{i,j}^{(t+1)}$, and furthermore, the pixel luminance value must be set to 255. If $f_{i,j}^{(t+1)}$ is greater than 255, then $f_{i,j}^{(t+1)}$ must be set to $255 + \frac{255+(255-f_{i,j}^{(t+1)})}{b}$.

The process of steps 1 and 2 is to be repeated T times for all pixels in the photographic image, and the image composed of the new pixels is the edge-preserving bubble image.

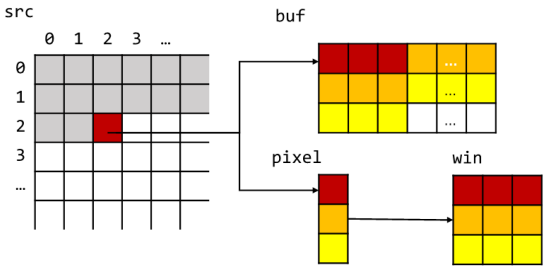

Fig. 2 Buffer Usage for Window Processing

3. Develop of Software Program for HLS

To create efficient hardware, each pixel of the processed image should be read and processed one by one, so that data is continuously fed into the system at each clock cycle, and high-speed processing is achieved through pipelining. To achieve this performing a window-based process, we use a buffer as shown in Fig. 2, and generate the ideal hardware using high-level synthesis. By using

```
for( i = 0 ; i < h; i++ ){
  for( j = 0; j < w; j++ ){
    pix = GetGray( src[i * w + j] ); // Gray scaling
    // Line buffer update
    for( k = 0; k < WIN_N - 1; k++ ){
      buf[k][j] = buf[k + 1][j];
      pixel [k] = buf[k    ][j];
    }
    pixel[WIN_N - 1] = buf[WIN_N - 1][j] = pix;

    // Window update
    for( k = 0; k < WIN_N; k ++ )
      for( l = 0; l < WIN_N - 1; l++ )
        win[k][l] = win[k][l+1];
    for( k = 0 ; k < WIN_N; k++ )
      win[k][WIN_N-1] = pixel[k];

    // Update brightness by Eq.(1) and Eq.(2)
    f = UpdateGrayFixed( win, t );

    // Output new brightness
    dst[i * w + j] = f << 16 ¦ f << 8 ¦ f;
  }
}
```

Fig. 3 Pseudo Source Code for HLS

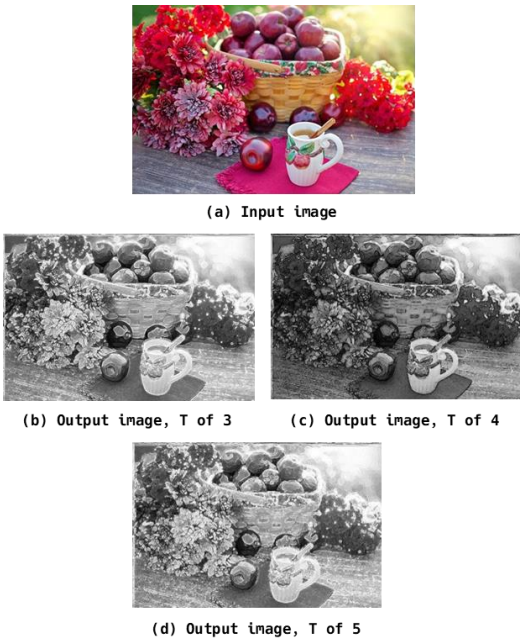

Fig. 4 Input and output Image

three buffers, the memory accesses are performed one by one pixel continuously while performing window processing. Fig. 3 shows the overview of the software programming code we developed for HLS.

4. Experiment and Discussion

4.1. Experimental Setup

We done the experiments on the real prototype system built on FPGA. We used HLS tool is Xilinx Vitis HLS 2021.1. The hardware generated is implemented on the ZYNQ FPGA. We measured the performance on ZYBO which is an FPGA board with ZYNQ FPGA.

The images used in the experiment are shown in Fig 4 (a). It is a BMP file image of size 427 in height and 460 in width. The output images are also shown in Fig. 4(b)-(d) with the iteration number of 3, 4 and 5.

As comparable evaluations, we prepared three versions: software execution on PC, software execution on embedded processor in FPGA, hardware execution in FPGA. The processor of PC is Core i5 at 3.7GHz. The embedded processor is Cortex A9 at 650MHz. The HLS hardware modules run at 100MHz in FPGA.

4.2. Effect of Software Restructuring

To clarify the effect of software restructure introduced to make memory access perform one pixel by one pixel, we

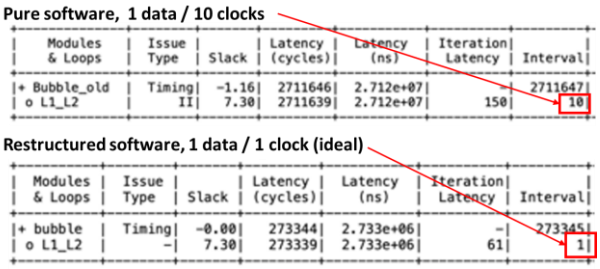

Fig. 5 HLS Result

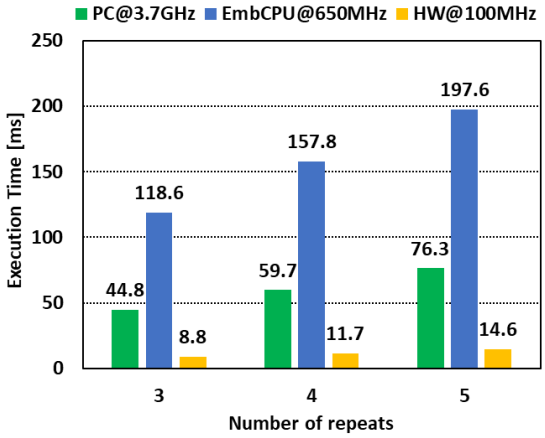

Fig. 6 Execution Time

compare the reports output by the HLS tools. Fig. 5 shows the results that HLS tool converted the pure software intuitively implementing the algorithm and the restructured software considering hardware characteristic shown in Fig. 3.

For pure software, the HLS generated a poor hardware module just produces 1 data per 10 clocks. It is not optimum pipelined data path. In contrast, the HLS can generate the well-organized hardware module with optimum pipelined data path outputting 1 data per 1 clock from the restructured software programming code.

4.3. Performance Evaluation

Fig. 6 shows the measured execution time of PC, embedded processor and FPGA respectively.

Based on Fig. 6, we confirmed that the hardware execution time was reduced by a factor of about 5 and 13 compared to the software execution time on the computer and that on the embedded processor respectively. Our hardware module generated by HLS from software code restricted can achieve a significant performance improvement compared with the software execution.

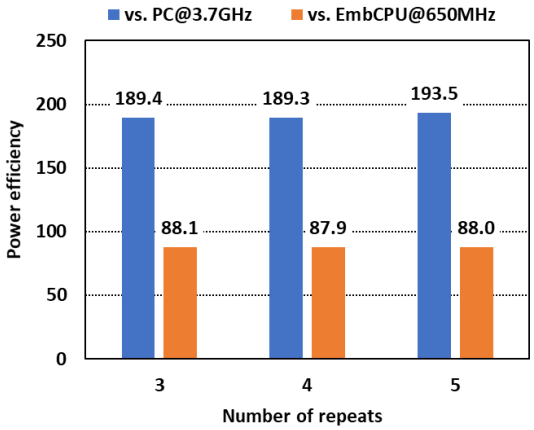

Fig. 7 Power Efficiency

4.4. Power Efficiency Estimation

To investigate a power efficiency about hardware module compared to the software execution, we estimate a operating power efficiency by Eq. (3).

Fig.7 shows the power efficiency estimated by the measured values shown in Fig. 6 and Eq. (3). This result shows that the hardware module generated by HLS from out restricted software can achieve a significant power efficiency compared to the software execution on PC and

$$\text{Power Efficiency} = \frac{\text{Software operating power}}{\text{Hardware operating power}} = \frac{\text{SW exec. time} \times F_{CPU}}{\text{HW exec. time} \times F_{FPGA}} \quad (3)$$

F_{CPU} = 3.7GHz, PC = 650MHz, embedded CPU

F_{FPGA} = 100MHz

embedded processor.

5. Conclusion

In this paper, we focused on speeding up the EPB bubble image conversion system and developed a software description method to generate efficient hardware by high-level synthesis. In this study, we developed a software description method to generate efficient hardware by high-level synthesis. When the hardware generated by high-level synthesis was compared with the software, it was confirmed that the power efficiency was also greatly improved. Furthermore, when compared to the execution on an embedded processor, the performance was also improved. Considering these points, we can conclude that the proposed method is an effective method for embedded image processing systems that require high-speed processing with low power consumption.

References

1. Shinya TAKAMAEDA, "Hardware Design with High Level Synthesis : Custom FPGA Computer Development Is Available to Everyone," Journal of the Institute of Electronics, Information and Communication Engineers, Vol.100, No.2, pp. 103-108 (2017) in Japanese.
2. Nane R, Sima VM, Olivier B, Meeuws R, Yankova Y, and Bertels K, "DWARV 2.0: a CoSy-based C-to-VHDL hardware compiler," In Proceedings of 22nd international conference on field programmable logic and applications, Oslo, Norway, 29–31 August 2012, pp 619‑622.
3. Pilato C and Ferrandi F, "Bambu: a modular framework for the high-level synthesis of memory-intensive applications," In: Proceedings of 23rd international conference on field programmable logic and applications, Porto, Portugal, 2–4 September 2013, pp. 1–4.
4. Canis A, Choi J, Aldham M, Zhang V, Kammoona A, Anderson JH, Brown S, and Czajkowski T, "LegUp: high-level synthesis for FPGA-based processor/accelerator systems," In: Proceedings of the 19th ACM/SIGDA international symposium on field programmable gate Arrays, Monterey, CA, USA, 27 February–1 March 2011, pp 33–36.
5. Toru HIRAOKA, "A High-Speed Method for Generating Edge-Preserving Bubble Images" IEICE TRANS. INF & SYST. VOL.E103-D, NO.3 MARCH 2020.

Authors Introduction

Ms. Jiang Qin

She entered the department of Electrical and Electronic Engineering, Faculty of Engineering, Kyushu Institute of Technology in 2018. She is now pursuing her B.E. study at the same university. Her current research interests are in Hardware development of image conversion.

Dr. Akira Yamawaki

He is an Associate Professor of Faculty of Engineering at Kyushyu Institute of Technology in Japan. He received his PhD. in Electrical Engineering from the Graduate School of Engineering, Kyushu Institute of Technology, in 2006. His reserch interest in Digital Circuit Systems.