Effect of Sorting Algorithms on High-level Synthesized Image Processing Hardware

Kohei Shinyamada^a, Akira Yamawaki^{a,*}

^aKyushu Institute of Technology, 1-1 Sensui-cho, Tobata-ku, Kitakyushu-shi, Fukuoka, 804-8550, Japan

*Corresponding Author: yama@ecs.kyutech.ac.jp

Abstract

Image processing methods can be broadly classified into hardware and software processing. Hardware is suitable for embedded systems because of its high performance and low power consumption. In hardware development, high-level synthesis is often used because of its ease of development. However, in order to generate high-performance hardware, it is necessary to write at the software level, considering the configuration of the hardware. Since sorting algorithms are often used inside image processing, it is necessary to generate high-performance sorting algorithm hardware. In previous research, methods for generating high-performance sorting hardware using high-level synthesis and performance comparisons have been conducted, but no comparison has been made for image processing as a whole. In this study, we will examine the dynamic background subtraction method, which is an image processing method that uses sorting algorithms. As a result, it was found that simple algorithms such as bubble sort and odd-even sort can realize pipeline processing, which is a feature of hardware, and produce high-performance image processing hardware.

Keywords: FPGA, Image Processing, HLS.

1. Introduction

Image processing methods can be divided into hardware processing and software processing. In general, hardware processing is more suitable for embedded systems because it has higher performance and lower power consumption ⁽¹⁻²⁾. The market scale of embedded image processing systems is expanding year by year ⁽³⁾. In order to increase competitiveness and gain market share in such a market, it is essential to develop and launch products with

image processing hardware as early as possible. To achieve this, we use high-level synthesis tools that automatically convert high-level languages such as C into hardware description languages such as VHDL. However, high-level synthesis also has its weaknesses, and if the description does not consider the characteristics of the hardware, efficient hardware will not be generated.

If we look at the internals of image processing, there are processes where sorting algorithms are used. For example, in dynamic background subtraction using the median method, a sorting algorithm is used ⁽⁴⁾. The median filter, which is used to remove sesame salt noise from images, also calculates the median internally. Thus, the sorting algorithm is used in various image processing and image preprocessing steps, so improving the sorting algorithm can improve the overall performance of the image processing.

Prior research has been conducted on methods for generating high-performance sorting hardware using high-level synthesis and on comparing the performance of sorting algorithms ⁽⁵⁻⁷⁾. However, these studies evaluated sorting algorithms alone, and it is not clear to what extent differences in sorting algorithms affect the performance of high-level synthesis hardware in overall image processing. In this study, we examine the dynamic background subtraction method as an image processing method that uses a sorting algorithm, and compare high-level synthesis hardware with different sorting algorithms in terms of processing speed, hardware scale, and power efficiency.

2. High-level synthesizable sorting algorithms

The high-level synthesis tool cannot convert programs that contain recursive functions into a hardware description language. Therefore, the target of the evaluation is a sorting

algorithm that does not use recursive functions.

2.1 Bubble Sort

Bubble sort is one of the simplest methods of sorting. The elements are checked in order from the end of the column, and if there is a reversal of the size relationship, the elements are swapped back and forth. When this process is done from the end of the column to the top, the smallest element is placed at the top. The next step is to do the same process starting from the tail and working up to the second from the top, and the next smallest element will be the second. By repeating this process, the whole thing can be sorted. The program written in C language is shown in Figure 1 (a).

2.2 Odd-even Sort

Odd-even sort is an improved version of bubble sort. While bubble sort scans in one direction, odd-even sort scans in pairs, so it can be processed in parallel. The program written in C language is shown in Figure 1 (b).

2.3 Selection Sort

Selection sort searches for the minimum or maximum value from unaligned elements and replaces it with the last element of the array. The program written in C language is shown in Figure 1 (c).

3. Dynamic Background Subtraction Method as Image Processing for Verification

The purpose of this study is to evaluate the effect of sorting algorithms on image processing. In this study, the median-based dynamic background subtraction method is used as the image processing for verification.

First, the background subtraction method will be explained. The background subtraction method is a process of taking the difference between the input image and a background image prepared in advance. Since taking the difference shows the difference from the foreground, it can be applied to surveillance cameras.

There are disadvantages to the simple background subtraction method. In the simple background subtraction method, the foreground image is static. Therefore, in an environment where the background changes from moment to moment, it is not possible to obtain the correct difference image. In order to solve this problem, the dynamic background subtraction method exists.

```
uint8_t bubble_sort( uint8_t * array ){
    int i, j;
    uint8_t tmp;

    for( i = 0; i < N; i++ ){
        for( j = N - 1; j > i; j-- ){
            if( array[j] < array[j-1] ){
                tmp = array[j];
                array[j] = array[j-1];
                array[j-1] = tmp;
            }
        }
    }

    return array[ N / 2 - 1 ];
}
```

(a) Bubble Sort

```
uint8_t odd_even_sort(uint8_t *array){
    int i, j;
    uint8_t tmp;

    for(i=0; i<N/2; i++){
        for(j=0; j < N-1; j+=2){
            if(array[j] > array[j+1]){
                tmp = array[j];
                array[j] = array[j+1];
                array[j+1] = tmp;
            }
        }
        for(j=1; j < N-1; j+=2){
            if(array[j] > array[j+1]){
                tmp = array[j];
                array[j] = array[j+1];
                array[j+1] = tmp;
            }
        }
    }

    return array[ N/2 -1];
}
```

(b) Odd-even Sort

```
uint8_t selection_sort(uint8_t *array){
    int i, j, mini;
    uint8_t tmp;

    for(i=0; i<N; i++){
        mini = i;

        for(j=i+1; j<N; j++){
            if(array[j] < array[mini]){
                mini = j;
            }
        }

        if(mini != i){
            tmp = array[i];
            array[i] = array[mini];
            array[mini] = tmp;
        }
    }

    return array[ N/2 -1];
}
```

(c) Selection Sort

Fig. 1. The program written in C

The dynamic background subtraction method is to dynamically update the foreground image and take the difference between that image and the input image. By using this method, the effect of background changes on the difference image is reduced. There are several methods for generating dynamic background images, including the sequential update method and the statistical background subtraction method, but here the median method using a sorting algorithm will be used for verification.

This section describes the median-based dynamic background subtraction method. First, for the generation of the foreground image, any N frames are stored in memory. Then, the data of identical pixels are sorted and the median values are obtained. The median-based dynamic background subtraction method dynamically constructs a foreground image using these median values and calculates the difference between that image and the input image. The program written in C language is shown in Figure 2.

4. Experiments and Discussions

4.1 Experimental environment

The FPGA boards used in the evaluation are as follows. The configuration of the experimental environment is shown in Figure 3.

Maker: Digilent
Product number: Z7-10
FPGA: Zynq-7010
- Note: SoC integrated ARM Cortex-A09 and FPGA
Memory Capacity: 1GB

Fig. 3. Configuration of the experimental environment

```
void bgs_med( uint8_t *prev_img, uint8_t *curr_img, uint8_t *diff_img )
{
    #pragma HLS INTERFACE s_axilite port=return
    #pragma HLS INTERFACE m_axi port=prev_img offset=slave bundle=d0
    #pragma HLS INTERFACE m_axi port=curr_img offset=slave bundle=d1
    #pragma HLS INTERFACE m_axi port=diff_img offset=slave bundle=d0

    uint8_t buff[N][IMG_W];
    int i, j, k;

    for( i = 0 ; i < IMG_H; i++){ // Write to Line Buf
        for( k = 0; k < N; k++){
            for( j = 0; j < IMG_W; j++){
                #pragma HLS PIPELINE II=1
                buff[k][j] = prev_img[ (k * IMG_W * IMG_H) + (i * IMG_W) + j ];
            }
        }
    }

    uint8_t x[N]; // Get median and Calc diff
    for ( j = 0; j < IMG_W; j++){
        #pragma HLS PIPELINE II=1
        for( k = 0 ; k < N; k++){ x[k] = buff[k][j];
            diff_img[ i * IMG_W + j ] = abs(curr_img[ i * IMG_W + j ] - sort(x));
        }
    }
}
```

Fig. 2. Description of the dynamic background subtraction method in C language

4.2 Evaluation method

Evaluate in terms of processing speed, hardware scale, and power efficiency.

Processing speed is calculated based on the number of clocks it takes to complete a series of processes.

$$Exec\ Time\ [s] = \frac{Total\ number\ of\ Clocks\ [clk]}{Clock\ Frequency\ [Hz]}$$

Hardware scale is the amount of SLICE, LUT, FF, etc. used.

Power efficiency is calculated by taking the hardware scale into account. The base for SLICE is the value of the simplest sorting algorithm, bubble sort HW. The higher the operating power improvement ratio; OPIR, the higher the performance of the hardware.

$$OPIR = \frac{CPU\ Exec\ Time[s] \times CPU\ Freq[Hz]}{HW\ Exec\ Time[s] \times HW\ Freq[Hz] \times \frac{SLICE(*)}{SLICE(0)}}$$

- SLICE(*): SLICE of Hardware for comparison
- SLICE(0): SLICE of Reference Hardware

4.3 Experimental results

In this section, the evaluation is conducted according to the evaluation methodology defined above and the results are summarized. The frequency of the FPGA is 100MHz, and the frequency of the embedded processor is 650MHz. The image size was 256 x 256 pixels, and was randomly and automatically generated internally. The median image was generated from 16 images. A graphical representation of the execution time is shown in Figure 4.

(a) Bubble Sort

Summarized in Table 1.

FPGA Exec Time:	17.1[ms]
Embedded CPU Exec Time:	227.1[ms]
OPIR:	86.5

(b) Odd-even Sort

Summarized in Table 1.

FPGA Exec Time:	16.9[ms]
Embedded CPU Exec Time:	204.4[ms]
OPIR:	79.3

(c) Selection Sort

Summarized in Table 1.

FPGA Exec Time:	134.9[ms]
Embedded CPU Exec Time:	251.0[ms]
OPIR:	14.1

Table 1. Experimental Results

Execution time [ms]	Bubble	Odd-Even	Selection
FPGA	17.1	16.9	134.9
Embedded processor	227.1	204.4	251.0

Amount of Hardware	Bubble	Odd-Even	Selection
SLICE	1009	1000	863
LUT	2706	2750	2316
FF	2826	2689	2752
DSP	0	0	0
BRAM	5	5	6
SRL	111	87	87

Fig. 4. Execution Time

Fig. 5. Operating Power Improvement Ratio

4.4 Considerations

When image processing was performed in hardware, bubble sort and odd-even transpose sort were very fast compared to selection sort. The computational complexity of all three algorithms compared in this study is $O(n^2)$. However, the reason why bubble sort and odd-even transpose sort were faster is thought to be because these

algorithms are very simple and suitable for pipeline processing. The total number of clocks for the entire image processing is also close to the number of pixels in the image, so it is thought that the performance of the hardware can be maximized.

Since the time spent on sorting accounts for the majority of the processing time in the median-based dynamic background subtraction method, it was found that high performance image processing hardware can be generated by speeding up the sorting algorithm.

In addition to that, the operating power improvement ratio shows that the hardware is more power efficient than the embedded processor; similar results can be obtained when compared to a PC CPU.

5. Experiments and Discussions

In this study, we evaluated the impact of sorting algorithms on high-level synthesized image processing hardware. Sorting algorithms are used in various types of image processing, so it is important to improve their performance. However, it is unclear to what extent differences in sorting algorithms affect the performance of actual high-level synthesized image processing hardware. Therefore, we used the median-based dynamic background subtraction method as a validation image processing to compare the performance.

In conclusion, we found that simple algorithms such as bubble sort and odd-even sort were fast because they could achieve pipeline processing, which is a feature of hardware. In contrast, selection sort could not achieve ideal pipeline processing and its performance was not as good as the former.

In the dynamic background subtraction method, it was found that the sorting of pixel values accounted for most of the processing time. Therefore, by implementing an appropriate sorting algorithm, we can build a fast system that maximizes the performance of the hardware.

In the future, we would like to verify whether similar results can be obtained with other image processing that uses the sorting algorithm.

References

- (1) "Stacking Up Software To Drive FPGAs Into The Datacenter," Next Platform, 20 Nov. 2016. [Online]. Available: <https://www.nextplatform.com/2016/11/20/stacking-software-drive-fpgas-datacenter/>. [Accessed 1 Oct. 2019].
- (2) "GPU vs FPGA Performance Comparison," Bertin, 19 May. 2016. [Online]. Available: <https://www.bertendsp.com/gpu-vs-fpga-performance-comparison/>. [Accessed 1 Oct. 2019].
- (3) "Applying Deep Learning Vision Technology to low-cost/power Embedded Systems," Synopsys, 18 Jan. 2017. [Online]. Available: <https://www.slideshare.net/JennyMidwinter/applying-deep-learning-vision-technology-to-lowcostpower-embedded-systems>. [Accessed 1 Oct. 2019].
- (4) B. LAUGRAUD, S. PIÉRARD, M. BRAHAM, and M. VAN DROOGENBROECK : "Simple Median-Based Method for Stationary Background Generation Using Background Subtraction Algorithms", In International Conference on Image Analysis and Processing (ICIAP), Workshop on Scene Background Modeling and Initialization (SBMI), Lecture Notes in Computer Science, Vol. 9281, pp. 477-484, Springer, 2015
- (5) Jmaa, Yomna Ben, R. B. Atitallah, D. Duvivier and M. B. Jemaa : "A Comparative Study of Sorting Algorithms with FPGA Acceleration by High Level Synthesis," *Computación y Sistemas*, Vol.23, pp. 213-230, 2019
- (6) A. Antonov, D. Besedin and A. Filippov : "Research of the Efficiency of High-level Synthesis Tool for FPGA Based Hardware Implementation of Some Basic Algorithms for the Big Data Analysis and Management Tasks," 2020 26th Conference of Open Innovations Association (FRUCT), pp. 1-7, 2020
- (7) Janarbek Matai, Dustin Richmond, Dajung Lee, Zac Blair, Qiongzhi Wu, Amin Abazari, and Ryan Kastner : "Resolve: Generation of High-Performance Sorting Architectures from High-Level Synthesis," In Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '16), pp.195–204, 2016