A Reconfigurable ASIC-like Image Polyphase Interpolation Implementation Method

Lei Yang
Institution of Automation, Chinese Academy of Sciences, University of Chinese Academy of Sciences
95 Zhongguancun East Road, 100190, Beijing, China
yanglei2013@ia.ac.cn

Ruoshan Guo, Shaolin Xie and Donglin Wang
Institution of Automation, Chinese Academy of Sciences,
95 Zhongguancun East Road, 100190, Beijing, China
{ruoshan.guo & shaolin.xie & donglin.wang}@ia.ac.cn

Abstract—**This paper presents a reconfigurable application specific integrated circuit (ASIC) like image interpolation implementation method. Parallel data flow and pipeline structure are used and optimized for efficient hardware acceleration. The coefficients could be easily reconfigured to support multiple interpolation filter taps and image resolution. We implement this method on a reconfigurable mathematical processing unit (MaPU). Experiment shows good computing performance and flexibility. With parallel acceleration and flexible configuration support, the implement method would be quite helpful for effective ASIC-like image processing hardware design.**

Keywords-image interpolation; acceleration; reconfigurable implementation

I. INTRODUCTION

Image interpolation is a procedure used in expanding and contracting digital images [1]. It relates to methods of placing new pixel values into a regularly sampled grid given a discrete subset of points taken from a smaller grid. There are many kinds of interpolation methods, such as bilinear, bicubic, spline interpolation and polyphase interpolation [2]. These methods have different features and limitations. For example, linear interpolation method relies on continuity, bicubic method assumes the signal and its first directive's continuity, and spines have continuous high-order directives [3]. Among these methods, polyphase interpolating has variable filters for different requirement so as to offering better image resizing quality, and it is commonly used in industry [4].

A polyphase interpolator adapts interpolation filter with variable taps. Custom interpolation design has different circuit structure for different filter taps. Usually the interpolation in vertical direction is slower than in horizontal direction because of the memory features that indexing or buffering columns are much slower than rows, as described in [5] and [6]. In industry, vertical interpolation usually has more complex circuit and uses more hardware resources, and it usually uses shorter-tap filter than in horizontal direction. As a result, traditional interpolation circuit has different circuit for different-tap filters.

As for the interpolation quality, filters that have more taps are better for image details reserving, but the overshooting and distortion is more obvious sometimes [7]. Generally, filters with more taps usually have better performance for natural images, while fewer taps are more suitable for computer generated images or vector graphs, as natural images usually have more nonstationary pixel edge and the other kind of images are usually more regular, as analysed in [8, 9]. The selection of filters is a trade-off based on application's requirement and hardware complexity. Meanwhile, many applications use auto-adapted filtering taps to meet different requirements, as the works described in [10, 11]. So the interpolation applications have the problem, that either they have to use fixed filtering circuits and live with the quality limitation, or they have to include different filtering circuits with a lot more complex circuit structure.

This paper introduces a reconfigurable ASIC-like implementation. The algorithm is specially optimized for parallel acceleration, and a reconfigurable control unit based on finite state machine (FSM) is used to control bus interface and computing unit. We implement a interpolation platform on a mathematical processing unit (MaPU) [12], a reconfigurable processing accelerator with SIMD structure for parallel acceleration. This paper brings up 2 new points: an ASIC-like interpolation algorithm optimization for parallel acceleration, and a reconfigure interpolation circuit structure to support multiple taps and variable image resolution.

II. ALGORITHM OPTIMIZATION

A. Custom Algorithm Flow

The polyphase image filter has 2 parts: the vertical filter and horizontal filter. The filter adapts to Lanczos-n function [13]:

$$f(x) = \sin c(x) \cdot \sin c(x/n) \tag{1}$$

We use a vertical filter *Vfilter* and a horizontal filter *Hfilter* as the discrete representation of the function, where the size of *VFilter* is *Vtap*Phase*, and the size of *HFilter* is *Htap*Phase*. *VTap* and *HTap* is the tap number in vertical direction and horizontal direction respectively.

For the pixel *X(i,j)* in the source image, we mark the vertical interpolating pixel *Y(i,j)*, and the final target pixel *Z(i,j)*. In vertical direction, the corresponding position of target pixel *Y(x,j)* in the original image is *(i+u, j)*, which is determined by the vertical zoom ration of target image *Y* and

Corresponding author; yanglei2013@ia.ac.cn

978-1-5090-3025-5/17/$31.00 ©2017 IEEE

original image *X*. The phase *u* is determined by the vertical filter *Vfilter(uPhase, Vtap)*, where

$uPhase = \lfloor u * Phase \rfloor$ ($\lfloor x \rfloor$ is the integral part of x).

Then the vertical interpolating target image *Y(x,j)* is calculated as:

$$Y(x,j) = f(X, Vfilter(uPhase, Vtap)) = \sum_{index=0}^{Vtap-1} Vfilter(uPhase, index) * X(i - Vtap/2 + index + 1, j) \quad (2)$$

After getting vertical target image *Y*, the final calculation is horizontal interpolating target image *Z(x,y)*. *Z(x,y)*'s corresponding position in image *Y* is *(x, j+v)*, and the phase *v* determines the horizontal filter *Hfilter(vPhase, Htap)*, where $vPhase = \lfloor v * Phase \rfloor$. Then the final target image *Z(x,y)* is calculated as:

$$Z(x,y) = g(Y, Hfilter(vPhase, Htap)) = \sum_{index=0}^{Htap-1} Hfilter(vPhase, index) * Y(x, j - Htap/2 + index + 1) \quad (3)$$

B. Custom Algorithm Flow

We consider the traditional interpolation mentioned before. The interpolation in vertical direction includes the following procedures:

```
1. Set i = 0; j = 0; pos = 0;
2. Calculate interpolating coordinate:
   pos = pos + pos_acc;
3. Calculate integral part of the coordinate:
   pos_int = (int)pos
4. Calculate phase from coordinate:
   pos_phase = (int)((pos − pos_int)∗NUM_PHASE)
5. Fetch source pixel according to pos_int:
   for(f = 0; f < F; f + +)
      {d[f] = src_image[pos_int − 3 + f][j]}
6. Get filter's coefficient according to pos phase:
   for(f = 0; f < F; f + +)
      {c[f] = c_table[pos_phase][f]; }
7. The F tap filtering calculation:
   out image[i][j] = 0;
   for(f = 0; f < F; f + +)
      {out_image[i][j] = +d[f] ∗ c[f]; }
8. Loop between step 2 to step 7 for each OW and OH.
```

(i, j) is the pixel coordinate of output image, and *OW*, *OH* is the output image's width and height. *NUM_PHASE* is the phase number of the filter and *F* is the tap number. For the input image *src_image*, the size of output image '*out_image*' is *OW*OH*. *C_table* represents the table of interpolation coefficients, and the item number of the table is *NUM_PHASE*. Each item in this table is an *F* tap interpolation coefficient. '*pos*' stands for the outputting pixel *(i,j)*'s position in the original image, and *pos_acc* is the interpolation step length. The core loop includes calculating the phase and integral part of each pixel, and get the corresponding source pixels and filter coefficients.

In the algorithm each target pixel requires the computing of its coordinate and phase for further obtaining source pixel's location and filter coefficient. In each output pixel there exists complex coordinate calculation and memory indexing operation.

C. Optimization Design

We optimize the tradition interpolation algorithm, to fit the efficient ASIC-like implementation. First, the complex process of computing coordinates and coefficients is removed from the interpolation core, as this part of computing is complex while having fixed pattern. All the coefficients are pre-determined or calculated by a specially designed unit (the scaler processing unit in MaPU [12]) and stored in the configurable FSM as control logic, and all the pixels access the memory in fixed order. For example, the interpolation in vertical direction of 1080 pixels have 1080 sets of coordinates and coefficients. As for a full image of 720*480 interpolating to 720*1080, traditional algorithm takes 720*1080 times of coordinates' calculation. As for the optimized algorithm, the process only takes 1080 times of calculation.

Further optimization is achieved by using the smallest zoom ratio to calculate coordinates and coefficients. For example, for the interpolation from 720*480 to 720*1080, in vertical direction the zooming ratio is 480:1080, which equals to the smallest ratio of 4:9. Then only 9 times of coordinates' calculation are needed, saving a lot of computing requirement. We prepare some frequently-used sets of coordinates in the configurable FSM for the acceleration of many common image resolution scaling. The optimized interpolation flowchart contains the following steps:

```
1. Generate update table and c table new
2. Set i = 0; j = 0; l = 0; loop = 0; pos int = 0;
3. Calculate integral part of coordinate:
   pos_int = (update_table)?(pos int + 1) : pos_int;
4. Index source pixel according to pos_int:
   for(f = 0; f < F; f + +)
      {d[f] = src_image[pos_int − 3 + f][j]; }
5. Index filter coefficient according to l:
   for(f = 0; f < F; f + +)
      {c[f] = c_table new[l[f]; }
6. The F tap filtering calculation: out image[i][j] = 0;
   for(f = 0; f < F; f + +)
      {out_image[i][j] = +d[f]∗c[f];}
7. Loop for each l < L, loop < LOOP, j < OW.
```

The main differences between traditional algorithm and optimized algorithm exist in the calculation of pixel's coordinate and phase. An important optimization is the mechanism of update table (*update_table*), as shown in figure 1. We can see the update table contains *L* table entries, and it only takes *L* loops to finish this part of computing. Considering the interpolation filer's phase number *NUM_PHASE* is 64, and the filter has 6 taps, then the size of

Identify applicable sponsor/s here. If no sponsors, delete this text box. (sponsors)

traditional *c_table* is *NUM_PHASE*F=64*4*. It contains 64 table entries, each of which is 6 tap filtering coefficients. While the size of the new interpolation coefficients table *c_table_new* is *L*F*. The computation amount and complexity of the optimized flow are shown in table 1 in comparison with traditional ones (interpolation in vertical direction, the image's size is *Lh*Lv*).

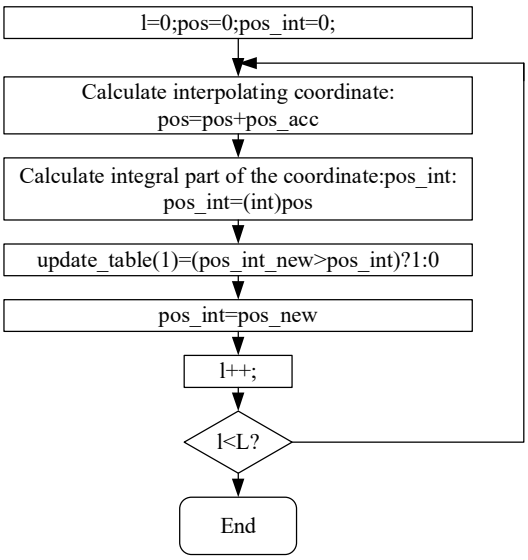

Figure 1. The calculation of update table

TABLE I. ALGORITHM COMPLEXITY COMPARISON BETWEEN TRADITIONAL METHOD AND OPTIMIZED METHOD

	Nh*Nv to Lh*Lv	**720*480 to 720*1080**
Traditional method	O(Lh*Lv)	O(720*1080)
Optimized 1	O(Lv)	O(1080)
Optimized 2	O(L)	O(9)

III. RECONFIGURABLE AISC-LIKE IMPLEMENTATION

A. *Overall structure*

The structure of implementation circuit is shown in figure 2. The reconfigurable parallel interpolation unit contains a local memory, a programmable FSM controlling unit, 2 bus interface units, a data buffer, a coefficients buffer, and some multiplier–accumulators (MAC). The FSM controlling unit is reconfigurable and controls the behaviour of all the other units. Local memory stores input/output image data and filtering coefficients. Bus unit 0 loads the source image data into data buffer, and bus unit 1 loads the coefficients stored in local memory into coefficients buffer. The reconfigurable parameter *F* is the filter tap number in FSM. For each row of interpolating, the data buffer keeps *F* rows of source image pixels, and in each clock cycle *P* pixels are sent to *P* MACs. The coefficients buffer stores *F* sets of coefficients for a row's interpolation. In each clock cycle a set of filtering coefficients is broadcasted to all the MACs, and the MAC units finish all the multiplications and accumulations in *F* clock cycles and output *P* interpolating pixels. The *P* output pixels are written back to local memory through the bus interface unit 2.

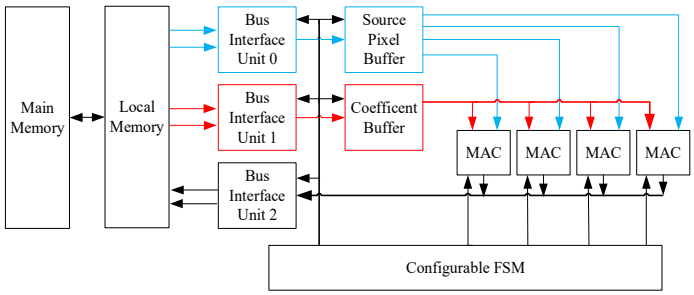

Figure 2. Overall of interpolation system structure

An important advantage of the implementation is that we could use the same circuit to implement the algorithm both in horizontal and in vertical direction with simple configuration. Here we take the advantage of multi-granularity parallel memory (MGP) system, as descripted in [12]. MGP provides a way to fetch matrix data in a row or a column in equal time. So the time of inversing matrix data is saved. In our implementation, we fetch *P* pixels in a matrix row for vertical interpolation in each cycle, or we fetch *P* pixels in a matrix column for horizontal interpolation in a clock cycle.

B. *Coefficients Design*

The coefficient path refers to the bus interface unit 1 and the path that coefficient buffer handling data to the MAC units, as the red lines and rectangles shown in figure 2.

We store some frequently-used interpolating coefficients in a table and store it into a certain memory block.

The data flow has 3 main steps. First, for each row of interpolation pixel, the first set of coefficients is updated. Then through BIU coefficients are load to the buffer, and the procedure loops after every L pixel rows. Finally, all the coefficients are broadcasted to each of the MAC unit.

Then we need a coefficients writing buffer and coefficients reading buffer to buffer the coefficients into the core interpolation computing. The algorithm reads a set of filter coefficients for each pixel row's processing. The FSM identifies reading buffer id and then broadcasts the coefficients into each MAC unit.

C. *Data Buffering and Pipeline Design*

The coefficient path refers to the bus interface unit 0 and the coefficient buffer handling data to the MAC units, as the blue lines and rectangles in figure 2.

The data has fixed pattern to input. We use a data supply window to control the data update process. The window size and position are determined by the image scaling ratio and are controlled by the configurable FSM unit.

When the input pixels don't update, the data buffer control pointer moves *N* rows, and each time of updating there are *N* pixel rows loaded. Meanwhile, when the input pixels need to update, the writing pointer's starting address doesn't change,

and it updates *N+3* pixel rows each time. The data writing buffer update scheme is demonstrated in figure 3.

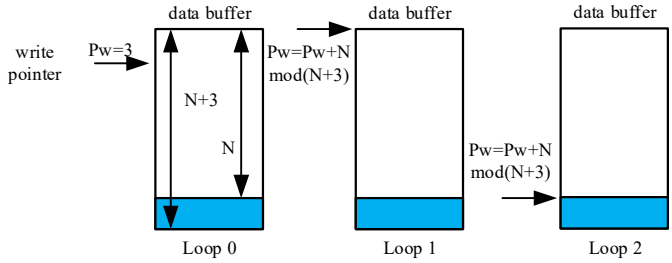

Figure 3. The calculation of update table

IV. EXPERIMENT AND PERFORMANCE

In this section, we run the interpolation platform to process some image zooming experiments. The source image size has several groups: 64*64 is the basic process unit of our platform, 240*135 is a commonly used interpolation unit (1/64 in area of a 1080p image), and full 1080p source image.

To get quite precise statistics result and to examine the overhead of invoking the kernel on the accelerator and the sustained performance, all the data is measured by running the interpolation for 10 times and get the average result. The results of these experiments are shown in table 2. The precise time is obtained by MaPU's cycle accurate simulator [14]. The first 3 groups of experiment are the basic interpolation unit used in our platform. As shown in the table, the configuration time for a certain interpolation circuit is around 20-30us, and approximately equals to the interpolation time of a one piece of image unit. As for full size image like 720p and 1080p, the configurable time takes about 0.09 of full interpolating time. And the interpolation rate is growing with larger image. Usually, 8 tap and 16tap interpolation is widely used in TV image zooming, and the interpolation speed is quite high enough for TV application requirement.

TABLE II. THE INTERPOLATION STATISTICS

Group	Configuration Time (ns)	Kernel Time (ns)	Configuration Time Percentage
8Taps,64*64	22162	8370	0.726
8Taps,240*135	22853	59352	0.278
16Taps,240*135	29020	91755	0.240
PI16taps 720P X2	291673	3106181	0.086
PI 16taps 1080p X2	591580	6649375	0.082

This remarkable performance does not only rely on the advantages of instruction level parallel (ILP) and SIMD, but also benefits from the elaborately implemented algorithm pipeline structure. Besides the reduction of the kernel interpolation complexity, the memory accessing designs are also important, to reduce memory access operations and increase the functional units' usage ratio. Except for matrix transposition, which relies on the memory features, we intend to keep data moving and processing within computational function units, and certainly prohibit the excrescent transfers involving local memory. Simulator statistics prove that the tested algebraic instructions comprise more than 50% operations for pure computation and only 18% operations for loads and stores on average. And this is a quite important rule for any ASIC-like data-dense algorithm implementation.

V. SUMMARY

This paper introduces a reconfigurable ASIC-like image implementation method. The algorithm is specially optimized for parallel acceleration, and a reconfigurable control unit based on finite state machine (FSM) controls bus interface and computing unit. Some basic implementation thoughts and algorithm optimization compared with traditional algorithm are introduced, such as the separation of phase and coefficients calculation, and parallel data flow for parallel implementation. We implement the method on a mathematical processing unit (MaPU), a reconfigurable processing accelerator with SIMD structure for parallel acceleration. Experiments show the good computing performance of our platform. The implementation method is typical for data sense algorithm's implementation for ASIC-like circuit or any other kind of effective hardware.

REFERENCES

[1] Prashanth, H., H. Shashidhara, and B.M. KN. Image scaling comparison using universal image quality index. in Advances in Computing, Control, & Telecommunication Technologies, 2009. ACT'09. International Conference on. 2009. IEEE.

[2] Carlson, R. and F. Fritsch, Monotone piecewise bicubic interpolation. SIAM journal on numerical analysis, 1985. 22(2): p. 386-400.

[3] Carey, W.K., D.B. Chuang, and S.S. Hemami, Regularity-preserving image interpolation. IEEE transactions on image processing, 1999. 8(9): p. 1293-1297.

[4] Franzen, O., C. Tuschen, and H. Schröder. Intermediate image interpolation using polyphase weighted median filters. in Photonics West 2001-Electronic Imaging. 2001. International Society for Optics and Photonics.

[5] Chrysafis, C. and A. Ortega, Line-based, reduced memory, wavelet image compression. IEEE Transactions on Image processing, 2000. 9(3): p. 378-389.

[6] Rixner, S., et al. Memory access scheduling. in ACM SIGARCH Computer Architecture News. 2000. ACM.

[7] Kim, D., et al., Lens distortion correction and enhancement based on local self-similarity for high-quality consumer imaging systems. IEEE Transactions on Consumer Electronics, 2014. 60(1): p. 18-22.

[8] Chen, S.E. and L. Williams. View interpolation for image synthesis. in Proceedings of the 20th annual conference on Computer graphics and interactive techniques. 1993. ACM.

[9] Burge, J. and W.S. Geisler, Optimal defocus estimation in individual natural images. Proceedings of the National Academy of Sciences, 2011. 108(40): p. 16849-16854.

[10] Wedi, T., Adaptive interpolation filters and high-resolution displacements for video coding. IEEE Transactions on Circuits and Systems for Video Technology, 2006. 16(4): p. 484-491.

[11] Vatis, Y. and J. Ostermann, Adaptive interpolation filter for H. 264/AVC. IEEE Transactions on Circuits and Systems for Video Technology, 2009. 19(2): p. 179-192.

[12] Wang, D., et al. MaPU: A novel mathematical computing architecture. in High Performance Computer Architecture (HPCA), 2016 IEEE International Symposium on. 2016. IEEE.

[13] TORAICHI, K., IMAGE INTERPOLATION TECHNIQUE USING THE SPACE-VARIANT COMPACTLY SUPPORTED FLUENCY DA FUNCTION OF DEGREE 2 Daiki FUKUI, Kazuki KATAGISHI, Yasuhiro OHMIYA Graduate school of Systems and Information Engineering University of Tsukuba.

[14] Yang, L., et al., An approach to build cycle accurate full system VLIW simulation platform. Simulation Modelling Practice and Theory, 2016. 67: p. 14-28.