DYNAMIC SCALING SCHEME FOR HARDWARE-ACCELERATED EDGE DETECTION

Quang Huy Nguyen[1], Myo Tun Aung[2], Timo Bretschneider[2], and Ian McLoughlin[1]

[1]School of Computer Engineering
Nanyang Technological University
N4-02a-32, Nanyang Avenue, Singapore 639798
{nguyenquanghuy, mcloughlin}@ntu.edu.sg

[2]EADS Innovation Works Singapore
41 Science Park Road #01-30
Science Park II, Singapore 117610
{myo_tun.aung, timo.bretschneider}@eads.net

ABSTRACT

Many real-time signal and image processing algorithms require extensive usage of floating-point arithmetic, which is very complex in implementation and resource intensive in hardware. However, faster and less complex fixed-point solutions do not provide the needed accuracy for selected applications Therefore, in this paper, an optimised dynamic scaling fixed-point process is suggested and implemented. The solution is tested for edge detection in polarimetric synthetic aperture radar (POLSAR) data, i.e. data with a very high dynamic range, and compare with an implementation using floating-point arithmetic. In summary, the dynamic scaling scheme provides a fast solution while preserving high accuracy. Using ModelSim to examine the performance, the proposed architecture operates four times faster than the original implementation- using a third of the original hardware resources. The accuracy was maintained at a high level, with the difference in output values of less than one for most of the cases.

Index Terms— Dynamic scaling, fixed-point, floating point, real-time embedded system, field programmable gate arrays.

1. INTRODUCTION

Real-time processing of POLSAR data has always been motivational yet challenging because of the computational complexity and multi-dimensional data structure. In recent years, a number of new algorithms for edge detection in POLSAR data have been proposed, e.g. the likelihood ratio edge detector [2] and the Roy's largest eigenvalue-based edge detector [3]. However, most of these edge detectors are computational expensive. Thus, Nguyen et al. [1] described a hardware-accelerated solution for the latter edge detector by making use of pipelining and parallelisation capabilities in FPGAs. The resulting implementation achieved a comparable accuracy with respect to the original C-language based implementation at a 25 times faster speed. However, floating-point modules were used extensively, and thus the maximal frequency speed was limited to 100 MHz. At the same time, the hardware resource usage was relatively high, with a slice count of up to 75,000 slices. The focus of this paper is on how to construct a fast and compact architecture that performs at a higher frequency, much lower hardware resource usage and comparable accuracy to the original FPGA implementation. This was achieved by designing a dynamic scaling scheme converting all expensive floating-point modules to appropriate fixed-point modules of the same data length.

The remainder of this paper is organised as follows: Section 2 discusses the advantages and disadvantages of floating-point and fixed-point representation for POLSAR processing. Section 3 presents the working principle of the implemented dynamic scaling scheme, while Section 4 discusses the required changes to the original FPGA implementation. Furthermore, a detailed performance comparison in terms of speed and accuracy is provided. Finally, Section 5 concludes the paper.

2. FIXED-POINT VERSUS FLOATING-POINT

The precision of the fixed-point number system is determined directly by the total number of representing bits, while for floating-point numbers, the precision is fixed inside the mantissa part, which can vary according to the actual application. In the used multi-look NASA/-JPL POLSAR data set [4], a 32-bit floating-point

representation is used, the mantissa is set to 24-bits. Thus, the precision of the 32-bit fixed-point number is higher than that of the 32-bit floating-point number. However, since the exponent part of a floating-point number can scale within a very large range, the precision of floating-point numbers can be maintained over the full range, even for very small numbers.

Careful examination of the NASA/-JPL data set revealed that the dynamic of the input data varies significantly, which suggest the use of floating-point arithmetic. However, the spatially compact support of the edge detector's templates (7×7 pixels) templates showed that most of the time, the contained neighbouring input values vary only within a limited range. Hence, the use of a floating-point numbers is not necessarily required. The diagram in Fig. 1 shows the differences in the input data range by subtracting exponent value of neighbouring pixels in a 7×7 window. Most of the time, the range differences are relatively small, except for specific regions, where the input data exhibits strong changes over a large range.

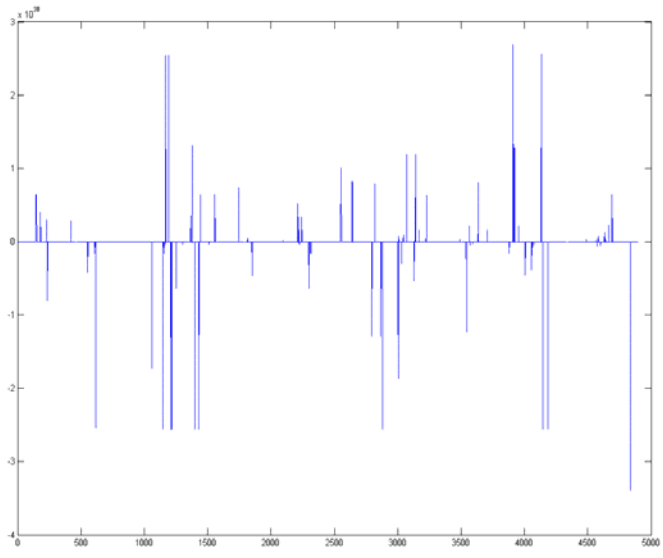

Fig. 1. Spatially variant range differences for a template

This observation motivated the use of a scaled fixed-point approach in order to adapt to the data's characteristics. However, the absolute differences among templates are large and unpredictable. Thus, for a rigid scaling fixed-point implementation, the computed result might suffer precision loss. To overcome this risk, a dynamic scaling scheme is implemented.

3. DYNAMIC SCALING SCHEME

Firstly, two input floating-point numbers are compared to determine the larger one, where it is sufficient to compare their exponent parts. In case of equality, no scaling or conversion is required and the mantissas are directly used in fixed-point representation during the later calculation. However, in case of inequality, three possibilities for choosing the amount for scaling exist: (1) scaling to preserve the most significant bit (MSB), (2) scaling to preserve the least significant bit (LSB), or (3) compromising at some middle range bit. Scaling to preserve the MSB means that the mantissa of the larger number remains unchanged, while the smaller mantissa is padded with leading zeros. The number of zeros is determined by the difference between the two original exponents. For this scaling policy, the smaller number suffers the risk of losing some of its LSBs, i.e. losing its original precision. In some extreme cases, where the difference between two exponents is too large, the small number is rendered to zero. Contrary, scaling can be carried to preserve the LSB, where the mantissa of the smaller number is preserved, and the mantissa of the larger number is zero padded behind the LSB. In this case, if the difference between the two exponents is too large, the larger number can saturate or overflow. The third scaling policy proves to be the best compromise out of the three policies.

For each template of 7×7 pixels, the third scaling policy pads zeros in front of the MSB and behind the LSB trying to minimise precision lost and preventing overflow at the same time. In order to determine the most suitable scaling amount, a MATLAB model for edge detection in the NASA/-JPL image data was constructed by comparing the achieved accuracy for a certain amount of scaling against the original FPGA implementation. Afterwards the optimal scaling was determined. Experimental results for above data set have shown that padding two zero bits in front of the MSB and six zero bits behind the LSB results in the highest obtainable precision. This empirical approach has to be repeated for every scene, but can be simplified by using previously obtained settings for the same area – an approach that is also used for the automatic gain setting in the corresponding radar sensor.

4. ARCHITECTURE & PERFORMANCE

In the original FPGA design of the Roy's largest eigenvalue edge detector, processing was spread over

four main blocks, using predominately floating-point modules. By replacing these floating-point modules with fixed-point modules, a number of changes had to be made to the original design. These are discussed in the following. For details on the functionality of each main block, the interested reader may refer to [1].

The first block in the original FPGA implementation, named Block A, involves the summation of covariance matrices for the test regions $r1$ and $r2$ – shown in Figure 2.

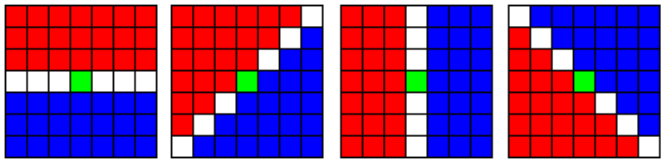

Fig. 2. Edge templates of 7×7 pixels. The green pixel in the centre indicates the pixel under test. The two test regions $r1$ and $r2$ are shown in red and blue colours, respectively. Each pixel contains a 3×3 Hermitian covariance matrix.

In the original architecture, Block A required six floating-point adders per template orientation. By replacing these with six fixed-point adders of the same data width, the performance improved significantly, i.e. the latency was reduced from 21 clock cycles to 6 clock cycles, maximal achievable operating frequency was increased from 156 MHz to 510 MHz, and hardware resource usage was reduced from 2100 slices to 400 slices.
As sliding the edge template horizontally from a pixel i to a pixel i+1, fixed-point scaling for input region $r1$ & $r2$ can be simplified by only comparing the original summation exponent with exponents of the three new input pixels. Thus, the number of required comparisons for scaling is reduced, from 21 to four.

The second and third block in the original FPGA implementation, namely Blocks B and C, involved the calculation of the inverse matrix $\mathbf{Z}_{r2}^{-1}$. In these two calculation blocks, the architectures are very similar and consist of three floating point adders and four floating-point multipliers, with only one exception of an extra floating-point divider in Block C. Again by replacing the floating-point modules with fixed point modules, the latency can be reduced to 22 clock cycles as compared to the original 66 clock cycles. The maximal achievable frequency was 275 MHz as opposed to 146 MHz and 106 MHz for Blocks B and C, respectively. Hardware resources also dropped about ten times from 6,000 slices to 550 slices.

The last block, namely Block D, is the most crucial block of the whole design because of its computational complexity. In the original design, the block implemented a custom-tailored fixed- and floating-point module, where all the arithmetic computations are performed in the floating point domain. Afterwards, the outcome of these arithmetic operations is converted to the fixed-point domain, and used as input to numerous trigonometric look-up tables (LUT) in the context of sine/ cosine and arctangent CORDIC. For the new fixed-point implementation, this final step is subdivided into two blocks, namely Block D1 and Block D2. The first block comprises all fixed-point arithmetic modules, while the latter block includes all the fixed-point trigonometric LUT. By dividing the computation into two blocks, the resultant latency dropped from 83 clock cycles to 25 clock cycles and 35 clock cycles, respectively. This is an improvement to the final pipelined architectures, since the output result is efficiently generated at the latency of the slowest unit. i.e. block D2. Another improvement is on the hardware resource usage. Because the original design involved the conversion between floating-point and fixed-point, two additional conversion modules were required. Moreover, precision lost as a result of conversion, questioning the use of all previous floating-point modules seems to be wasted since the accuracy is not preserved after that conversion. The graphical view of the new dynamic scaling fixed-point architecture is shown in Fig. 3, and the accuracy performance assessment is shown in Table I, respectively. With the output values range from zero to infinity, the accuracy was found to maintain at high level, as the difference in output values was limited to less than one for most of the cases, i.e. 42.5% of the total pixels.

The post-synthesis timing report of the new dynamic scaling fixed-point implementation suggested a maximum operating frequency of 200 MHz. Using this figure and simulating the proposed design for a sample set of multi-look NASA/-JPL POLSAR data under ModelSim, new output data were computed every 105ns.
A short summary of the performance analysis for is shown in Table II, while the side-by-side performance between the original floating-point implementation and the proposed dynamic scaling fixed-point implementation is shown in Table III.

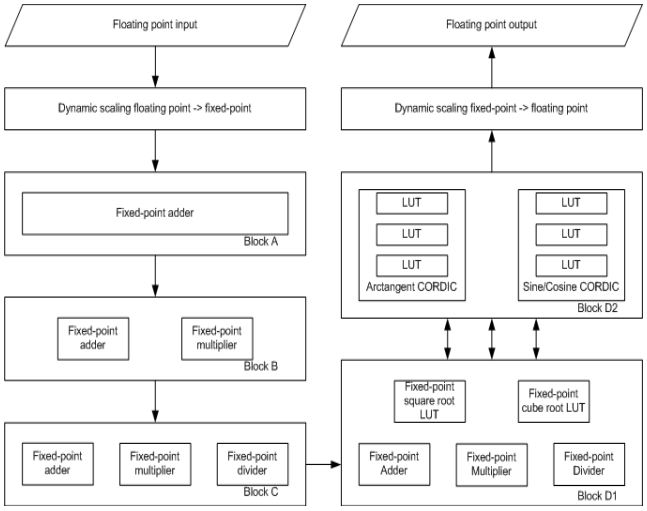

Fig. 3. Pipeline architecture of new dynamic scaling fixed-point implementation.

TABLE I
ACCURACY ASSESSMENT OF THE DYNAMIC SCALING IMPLEMENTATION

Accuracy	Percentage (256×256 pixels)
<0.01	1.4
±0.10	15.3
±1	42.5
±10	28.4
±100	12.4

TABLE II
PERFORMANCE OF THE DYNAMIC SCALING IMPLEMENTATION

	Slice count	Maximum frequency [MHz]	Latency [clock cycles]
Block A	350	510	6
Block B	250	275	22
Block C	300	275	15
Block D1	300	275	25
Block D2	4200	200	35
Core level	5400	200	35

TABLE III
PERFORMANCE COMPARISON

	Dynamic Scaling	Original FPGA
Slice count	22000	75,000
Maximum frequency (MHz)	275	106
Cycles per pixel	35	83
Pipeline structure (Stages)	4	4
Parallelised structure (Cores)	4	4
Time per frame (256×256 pixels; ms)	15	60

5. CONCLUSIONS

This paper described an optimised architecture for original FPGA implementation of edge detection in POLSAR data using a dynamic scaling scheme. The performance evaluation was carried out using a nine-look NASA/JPL C-band data and was benchmarked against the original FPGA implementation. In terms of accuracy, the dynamic scaling fixed-point implementation performed comparable to the original FPGA implementation, while it improved significantly with respect to the processing time and hardware resource usage. The dynamic scaling fixed-point implementation replaced all the respective floating point modules in original implementation with new fixed-point modules, which include scaling, arithmetic operations and an arctangent CORDIC function. This new architectures further enhance the computational speed, thus making real-time processing of POLSAR on-board a satellite more feasible.

REFERENCES

[1] Q. H. Nguyen, K.-Y. Lee, M. T. Aung, T. Bretschneider, I. Mcloughlin,"Hardware-accelerated edge detection for polarimetric aperture radar data", *IEEE Int. Geosci. Remote Sens.*, July 2009.

[2] J. Schou, H. Skriver, A. A. Nielsen and K. Conradsen, "CFAR edge detector for polarimetric SAR images", *IEEE Trans. Geosci. Remote Sens.*, vol. 41, no. 1, pp. 20-32, Jan. 2003.

[3] K.-Y. Lee and T. Bretschneider, "On detecting edges in polarimetric synthetic aperture radar

imagery", *Proc. of the 27th Asian Conf. on Remote Sens.*, paper no. J-1_J4.pdf, 2006.

[4] Mudamerbok 354-1, NASA/JPL POLSAR data http://airsar.asf.alaska.edu/data/cm/cm6419/