International Journal of MC Square Scientific Research Vol.9, No.1,2009

High-Performance Designs for Linear Algebra Operations on Image Processing

R. Madhusudhanan,
Research scholar,Sathyabama university, Chennai.
J. Augustin
Lecturer Department of ECE Kamaraj College of Engineering and Technology
Virudhunagar, India.

Abstract

Numerical linear algebra operations are key primitives in scientific computing. Performance optimizations of such operations have been extensively investigated. With the rapid advances in technology, hardware acceleration of linear algebra applications using field-programmable gate arrays (FPGAs) has become feasible. In this paper, we propose FPGA-based designs for several basic linear algebra operations, including dot product, matrix-vector multiplication, matrix multiplication, and matrix Factorization. By identifying the parameters for each operation, we analyze the trade-offs and propose a high-performance design. In the implementations of the design a convolution filter using dot product, this is especially useful in noise removal. The proposed designs are implemented on Xilinx FPGAs. Experimental results show that our designs scale with the available hardware resources. Also, the performance of our designs compares favorably with that of general-purpose processor-based designs.

Keywords: Salt and Pepper, Adaptive, Pipeline, FIFO, Rank order, Non linear

1. Introduction

Convolution filter is a powerful instrument used in image processing. The traditional median filtering algorithm, without any modifications gives good results. There are many variations to the classical algorithm, aimed at reducing computational cost or to achieve additional properties. Convolution filter are used mainly to remove salt-and pepper noise. Doing this, they preserve edges in the image (preserve their location and do not affect their steepness, unlike Gaussian filters), but unfortunately Convolution filtering may destroy small features in the image. A way to avoid it is to apply center weighted median filtering instead of a plain median; the drawback of this solution is the detoriation of the filter's ability to suppress impulse noise.

Common drawback of various kinds of the Convolution filtering is their computational cost. Computing a two-dimensional Convolution for an NxN window, requires sorting of NxN elements for every image pixel and choosing the Convolution value for the output. After the s orting, each queue element is assigned a value called a rank, specifying its position in the queues a result of sorting.

Therefore, using Convolution filtering in any real-time vision system requires a significant computational power. One way to speed up the computations is to implement the algorithm in hardware, e.g. with the help of FPGA circuits. The rationale behind this is to use the FPGA's inherent ability to execute operations in parallel. Moreover, programmable logic creates the possibility to tailor the implementation to the user's needs. All this results in a significant speedup over the software implementations by using sequential processors. One drawback of hardware-based algorithm development is the complexity of the design process as implementing algorithmically complex operations is very difficult. Convolution filtering, like many other low-level image processing algorithms is fairly simple - the main problem in this case is the amount of data to handle.

2. Convolution filter

Convolution filter is a non-linear, low-pass filtering method, which you use to remove "speckle" noise from an image. A convolution filter can outperform linear, low-pass filters on this type of noisy image because it can potentially remove all the noise without affecting the "clean" pixels. Convolution filters remove isolated pixels, whether they are bright or dark.

Prior to any hardware design, the software versions of the algorithms are created in MATLAB. Using MATLAB procedural routines to operate on images represented as matrix data, these software algorithms were designed to resemble the hardware algorithms as closely as possible. While a hardware system and a matrix manipulating software program are fundamentally different, they can produce identical results, provided that care is taken in development. This approach was taken because it speeds understanding of the algorithm design. In addition, this approach facilitates comparison of the software and synthesized hardware algorithm outputs. This project is focused on developing hardware implementations of image processing algorithm for use in an FPGA based image processing system. The rank order filter is a particularly common algorithm in image processing systems.. For every pixel in an image, the window of neighboring pixels is found. Then the pixel values are sorted in ascending, or rank, order. Next, the pixel in the output image corresponding to the origin pixel in the input image is replaced with the value specified by the filter order. The VHDL code can be simulated to verify its functionality.

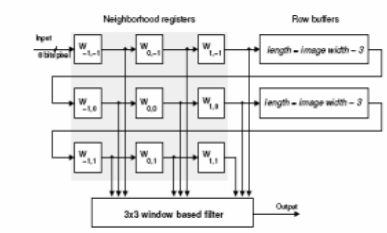


Fig. 1 Implementation of a 3 × 3 filter window filter.

3. Moving Window Architecture

In order to implement a moving window system in VHDL, a design was devised that took advantage of certain features of FPGAs. FPGAs generally handle flip -flops quite easily, but instantiation of memory on chip is more difficult. Still, compared with the other option, off-chip memory, the choice using on-chip memory was clear. It was determined that the output of the architecture should be vectors for pixels in the window, along with a data valid signal, which is used to inform an algorithm using the window generation unit as to when the data is ready for processing. Since it was deemed necessary to achieve maximum performance in a relatively small space, FIFO Units specific to the target FPGA were used. Importantly though, to the algorithms using the window generation architecture, the output of the window generation units is exactly the same. This useful feature allows algorithm interchangeability between the two architectures, which helped significantly, cut down algorithm development time. A window size was chosen because it was small enough to be easily fit onto the target FPGAs, and is considered large enough to be effective for most commonly used image sizes. With larger window sizes, more FIFOs and flip -flops must be used, which increases the FPGA resources used significantly. Figure 1, 2 shows a graphic representation of the FIFO and flip flop architecture used for this design for a given output pixel window.

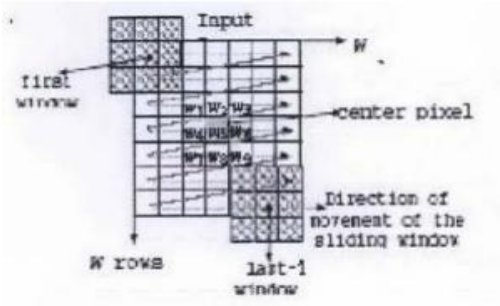


Fig.1 Moving Window Architecture

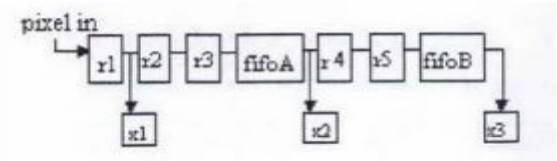


Fig. 2 Reading Pixels from Window.

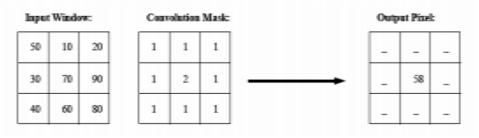
4. Linear algebra operations: Dot Product

For dot product, the minimum number of external I/O operations is 2n+1. There are n fixed point multiplications and n fixed point additions. We can easily identify k and l as the design parameters because they affect Tcomp. Since there is no data reuse in this operation, no on-chip memory or onboard memory is needed by the design. To reduce the latency, we overlap Tcomp and TI=O. Furthermore, we overlap the floating-point multiplications and additions. To achieve the lower bound, k ¼ . Thus, we propose an architecture which consists of an identical number of fixed point adders and multipliers. The fixed point adders and multipliers are pipelined so that additions, multiplications, and external I/O operations are overlapped. Moreover, pipelining for the adder and multiplier results in high clock speed.

5. Implementation and Testing:

Convolution is another commonly used algorithm in DSP systems. It is from a class of algorithms called spatial filters. Spatial filters use a wide variety of masks, also known as kernels, to calculate different result s, depending on the function desired. For example, certain masks yield smoothing, while others yield low pass filtering or edge detection.

$$y(n_1,n_2) = \sum_{k1=-\infty}^{\infty} \sum_{k2=-\infty}^{\infty} A(k_1,k_2)k(n_1-k_1,n_2-k_2),$$



Convolution Output = (50*1 + 10*1 + 20*1 + 30*1 + 70*2 + 90*1 + 40*1 + 60*1 + 80*1)/9 = 57.7778 => 58

The algorithm has three main purposes:

(a) To remove 'Salt and Pepper' noise.

(b) To smoothen any non impulsive noise.

(c) To reduce excessive distortions such as too much thinning or thickening of object boundaries.

7. Result

The Convolutionnal filter is designed to remove impulsive noise from images. Therefore, our algorithm's performance was first tested with basic salt and pepper noise with a noise density of 0.25. The next test involves processing images that contain impulsive and/or nonimpulsive noise. It is well known that the median filter does not provide sufficient smoothening of non-impulsive noise. Therefore, Gaussian and 'salt and pepper' noise were added to the image which was then processed by the algorithm. The Fig a, b show the performance of the adaptive median filter.

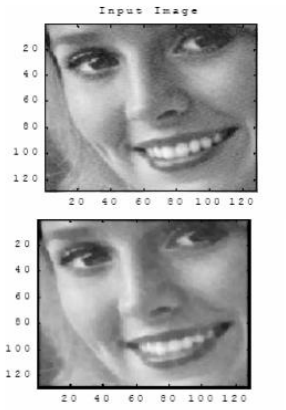


Fig 4.Results of filtering with a 3X3 Convolutional

filter.

6. Conclusion

The architecture is pipelined which processes one pixel per clock cycle, thus to process an image of size 256 x 256 it requires 0.65 ms when a clock of 100 MHz is used and hence is suitable for real time applications The Convolutional filter successfully removes impulsive noise from images. It does a reasonably good job of smoothening images that contain non-impulsive noise. Overall, the performance is as expected and the successful implementation of the Convolutional filter is presented.

References:

1. Zdenek Vasicek, Lukas Sekanina, Novel Hardware Implementation of Adaptive Median Filters 978-1-4244-2277-7/08/ ©2008 IEEE

2. Olli Vainio, Yrjö Neuvo, Steven E. Butner, A Signal Processor for Median-Based Algorithms, IEEE Transactions on Acoustics, Speech, Processing VOL 37. NO. 9, September 1989.

3. V.V. Bapeswara Rao and K. Sankara Rao, A New Algorithm for Real-Time Median Filtering, IEEE Transactions on Acoustics, Speech, Processing VOL ASSP-34. NO. 6, December 1986.

4. M. O. Ahmad and D. Sundararajan, Parallel Implementation of a Median Filtering Algorithm, Int. Symp. on Signals and Systems, 1988.

5. Dobrowiecki Tadeusz, Medián Szűrők, Mérés és Automatika, 37. Évf., 1989. 3.szám
[6] Xilinx Foundation Series Quick Start Guide, 1991-1997. Xilinx. Inc.