A Floating Point Convolution Systolic Cell

F. Larochelle, J.F. Côté, A.S. Malowany

Computer Vision and Robotics Laboratory
Department of Electrical Engineering
McGill University
Montréal, Québec, Canada

Abstract

This paper describes the design of a pipeline architecture double precision floating point systolic cell for convolution. Both the pixel intensities and the partial sum values are loaded four bits at the time which implies that a set of operands is loaded every 16 clock cycles. The arithmetic operations are distributed into three pipeline stages which enables the cell to process each set of operands at the same rate as they are loaded. While offering the same precision obtained on standard computers, our systolic cell reduces the convolution time expenditure by as much as three orders of magnitude.

KEYWORDS: Convolution, Floating Point Arithmetic, Pipeline Architecture, Systolic Cell.

I. Introduction

Computer vision and image processing enables robots to guide and validate their activities by supplying them with information about their environment [1]. Unfortunately, algorithms used to extract information out of images generally involve a great amount of computation. Two dimensional convolution is a low-level image processing operation that is necessary in several algorithms [2]. It is easily implemented in software but such implementation normally requires too much execution time for real time robotic applications.

In an effort to reduce the convolution processing time, several VLSI implementations of convolution processor have been proposed [3,4,5]. In our laboratory, a CMOS systolic chip [6] as well as a special architecture board [7] that uses this chip were developed and can be used to perform high speed convolution of images. However, like every other versions proposed in the literature, it uses only integer arithmetic with eight bit precision for the input pixels and sixteen for the resulting sum . We are presently investigating the design of a fast and reliable method of performing curvature estimates of range data [8]. Such a process requires performing six convolutions on a single image in order to compute the first and second derivatives at each pixel. Noise or step edges can produce very large variations making the use of integer arithmetic convolution processors referred above very difficult without generating overflows. It is easily realized that floating point arithmetic convolution is required in such applications.

To enable the construction of a floating point convolution processor, we designed a VLSI double precision floating point systolic cell. Simulation at the architectural level was performed and the results predict that its implementation could reduce the convolution time expenditure by as much as three orders of magnitude while preserving the double precision floating point accuracy that is possible when the convolution is performed in software.

II. Architecture Overview

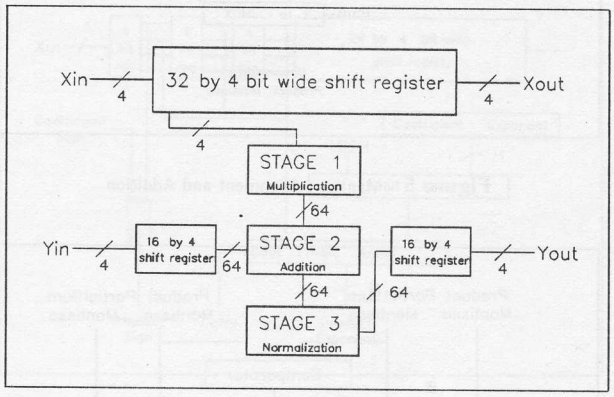

Figure 1 Architecture Block Diagram

The systolic cell is designed to multiply a pixel intensity by a given coefficient and add this product to a partial sum. The coefficient is constant for an entire image and is initially loaded into the cell. During convolution, both the pixel intensities and the partial sum values are serially loaded four bits at the time. A double precision floating point number is represented using eight bytes. It therefore requires sixteen clock cycles to load a set of operands. The cell is organized into three pipeline stages and enables operands to be processed in sixteen clock cycles as well. Figure 1 outlines the basic organization of the pipeline structure and specifies the respective task of each stage. In order to allow these cells to be chained into a systolic array, one extra 16 by 4 shift register has been incorporated inside the chip to delay the flow of intensities and ensure that each meets with the proper partial sum in the next cell. Overflows and underflows are detected and transmitted from one cell to the other. Under the IEEE

first bits of the partial sum value have already been stored into 15 4-bit registers while the last four are present on the Y input pins. At that moment, the exponent values are compared. The value with the biggest exponent is loaded into the fixed registers while the other is loaded in the variable ones. Both the fixed and variable mantissa registers are at least 54 bits wide since they must be able to hold the unnormalized product mantissa value computed in stage 1 which contains an extra bit. The least significant bit of the register containing the partial sum mantissa is set to 0 since this mantissa value contains only 53 bits.

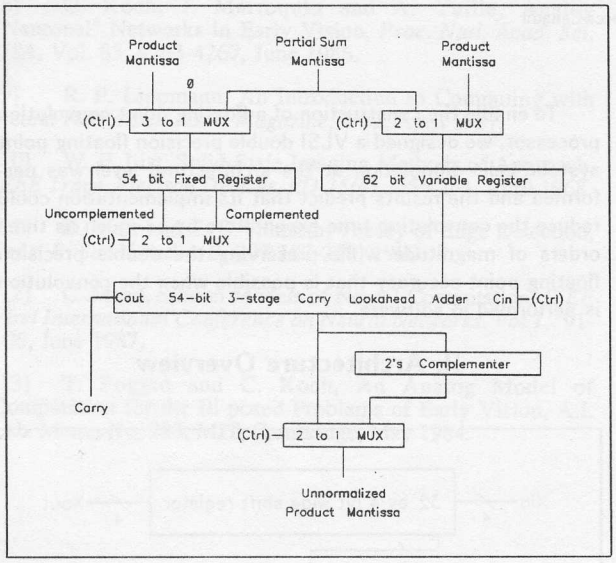

Figure 5 Mantissa Alignment and Addition

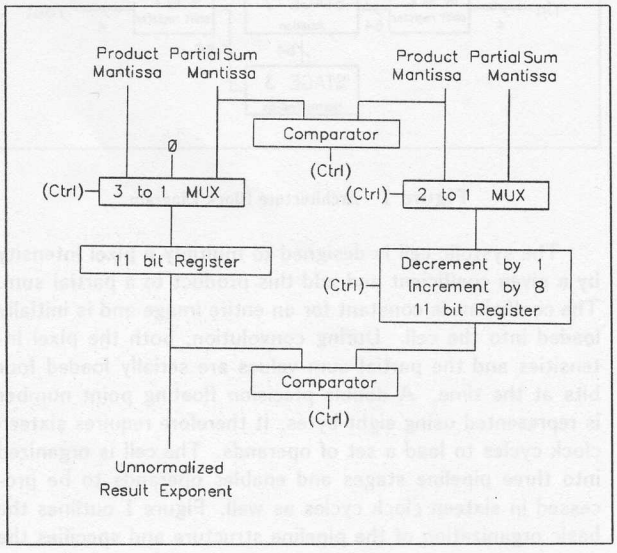

Figure 6 Exponent Incrementation and Decrementation

Before any two floating point numbers can be added together, their mantissa values must be aligned to obtain equal exponents. In our circuit, the smaller exponent is incremented by 8 and the corresponding mantissa is shifted right by 8 as long as the variable exponent value is smaller than exceed the fixed one. When this condition is reached, the variable exponent is decreased in steps of 1 until it exactly equals the fixed exponent value. Correspondingly, the mantissa is shifted left 1 bit each time the exponent is decreased. The variable register contains 8 extra bits to ensure that no bits are lost when the variable exponent value exceeds the fixed one during the alignment process.

Shifting the mantissa by steps of 8 and 1 provides a way to align the mantissas in a maximum of 14 cycles. The variable exponent value is automatically shifted left by 8 when loaded on clock 0. This makes sure that the exponents are aligned at clock 14 in the worst case. Note that if the exponents are unequal at clock 14, only 0's will be left in the variable register.

The aligned mantissas are added during the last 2 clock cycles. The 54 most significant bits of the variable register are added to the fixed register value. When both numbers have the same signs, the resulting sign is unchanged. However, when the signs are different, the content of the variable register is added to the complemented value of the fixed register with a forced carry-in to perform a two's complement addition. If the results is negative, the number is two's complemented and its sign is the same as the fixed number register value while the sign is set to the opposite value in the other case.

Overflows and underflows are treated the following way in stage 2. When either the generated product or the incoming partial sum is detected as underflow, its value is replaced by a forced 0 to leave the other number unaffected during addition. If either one of the two inputs is detected as overflow, the addition is performed normally except that the partial sum transmission unit will be signalled to output this number as an overflow (Described in section 7). Overflows and underflows generated during addition are detected in stage 3 while normalizing.

VI. Pipeline Stage 3

A floating point number is normalized when the left most 1 in the mantissa is exactly to the left of the binary point. Figure 7 shows the circuit used in stage 3 to normalize the result obtained in stage 2. Following the algorithm used in stage 2, the mantissa is shifted left by 8 and the exponent decreased by 8 until a 1 is present to the left of the binary point. When this condition is reached, the mantissa is shifted right by 1 and the exponent increased by 1 until the left most 1 in the mantissa is just left of the binary point. In the worst case, 14 clock cycles are needed to normalize any number. If normalization is not reached by clock 15, the number is reported to the partial sum transmission unit as being an underflow. Furthermore, an underflow is also signalled if after decreasing the exponent it is exactly zero or was negative and became positive. Similarly, an overflow is produced if after increasing the exponent it is all 1's or was positive and became negative.

VII. Partial Sum Transmission Unit

This unit is responsible for the transmission of the partial sum as an overflow, an underflow or as the result obtained

floating point standard, a number with the maximum exponent value represents an overflow. Similarly, one with the minimum exponent value represents a zero or an underflow.

III. The Coefficient Register

The coefficient value must be loaded in each cell before any processing is possible. To provide an efficient means of loading the coefficients into the cells, a 64-bit shift register which has both its serial input and output connected to external pins is used (See figure 2). This allows the coefficient register of each cell in a systolic array to be loaded through a common channel by chaining every coefficient register into a long shift register.

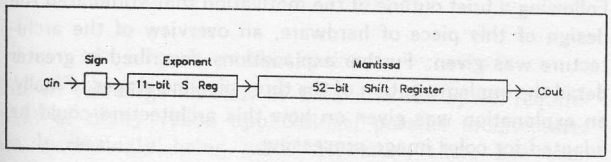

Figure 2 Coefficient Register

IV. Pipeline Stage 1

The standard algorithm for multiplying two binary numbers involves adding the mutiplicand shifted left by i to the partial product if bit i of the multiplier is 1. In our case, the multiplicand is the coefficient. Fortunately, the coefficient is constant during the entire convolution process. It was thus possible to compute in advance sixteen numbers corresponding to the coefficient value multiplied by 0 through 15. Having those previously computed, we are able to multiply four bits at the time. The modified algorithm consists of adding the coefficient value multiplied by x and shifted by 4i where x is equal to the value of the ith group of 4 bits of the pixel intensity. In this manner, we are able to multiply at the same rate as the intensity bits are loaded. Figure 3 shows the block diagram of the mantissa multiplication circuit. The outputs of the first 4-bit register of the X input buffer are used to select the product of C that will be added to the partial product register. At the rising edge of clock pulse 0, the partial sum register, which contains the result of the previous multiplication, is being loaded with the first partial product value of the multiplication just being started. At this cycle, the selected ceofficient product is added with a forced zero.

The last three groups of 4 bits in the intensity value contain the exponent and the sign bit. During clock 15, the exponents are added and the resulting sign is generated. Figure 4 shows the circuit used to perform this task. Notice that the coefficient exponent is always added with 402h. This serves two purposes. First, the exponents in double precision representation are stored in excess-1023 notation. When two such numbers are added, 1023 must be subtracted from the result to restore the excess notation representation. It is equivalent to an addition of 401h in two's complement. Secondly, when generating the product, it is possible to get a number with a 1 in the second position above the binary point. To avoid having to normalize the result at that moment, the number is interpreted as if the binary point had been shifted left by one

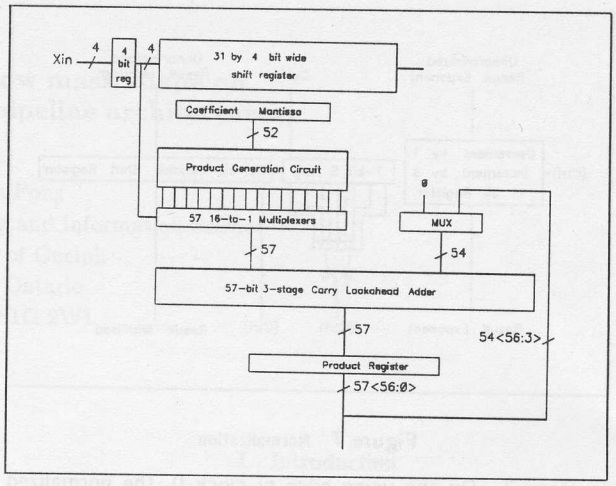

Figure 3 Mantissa Multiplication

position. To leave the value of the number unaffected, the exponent is incremented by one corresponding to the difference between 402h and 401h. To conserve a 53 bit mantissa precision, the least significant bit of the mantissa is masked when the number still has a 1 above the new binary point.

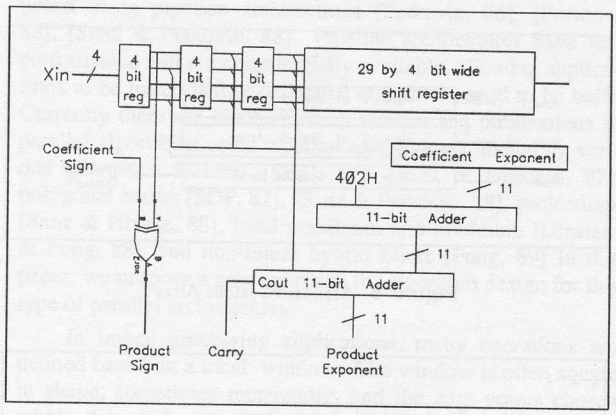

Figure 4 Exponent Addition and Sign Generation

Overflow is produced when either the coefficient or the intensity values is detected as being overflow or if the sum of two positive exponents reaches 1024. Underflow or a zero result is produced when either the coefficient or the intensity value is underflow or zero valued. An underflow is also generated when the magnitude of the sum of two negative exponents exceeds or equal 1023. Normal multiplication is performed even if overflows or underflows are detected. Underflows are signalled to stage 2 while overflows are reported to the partial sum transmission unit. This is further explained in the following sections.

V. Pipeline Stage 2

Stage 2 in the pipeline structure is responsible for the addition between the product generated in stage 1 and the incoming partial sum. Figure 5 and 6 shows a block diagram of the implementation. On the rising edge of clock 0, the 60

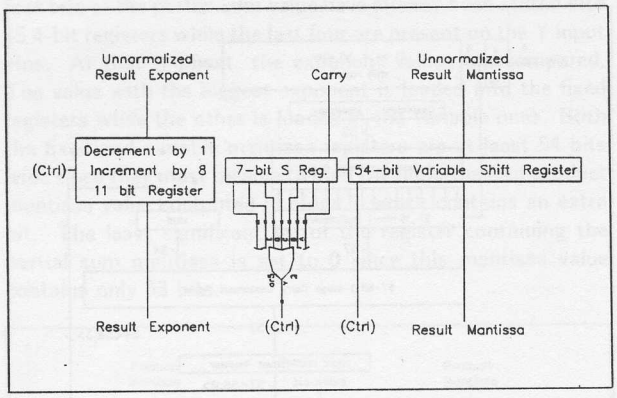

Figure 7 Normalization

in stage 3. On the rising edge of clock 0, the normalized number from stage 3 is loaded into 16 4-bit registers. For the following 16 clock cycles, the number is shifted out 4 bits at the time. If the number was signalled as an overflow or an underflow, the transmitted bits are forced to be all 1's or 0's respectively.

VIII. Special Application Requirements

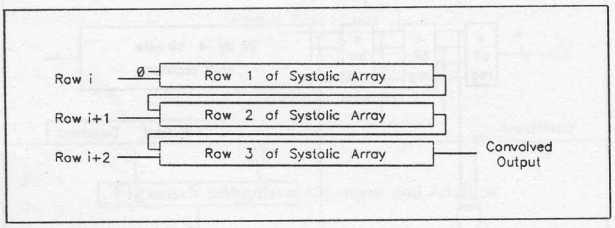

Figure 8 Standard Systolic Array

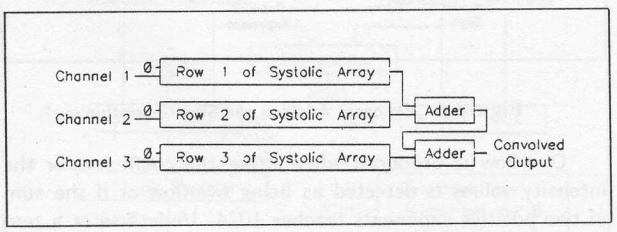

Figure 9 Modified Systolic Array

Figure 8 shows a typical organization of systolic cells to generate a 3x9 systolic array. All partial sum registers are chained together and each row is fed through three different channels. For standard convolution, row 1, 2, and 3 of the systolic array are fed with row i, i+1, and i+2 of the image. However, each row could easily be supplied with pixels that are not from the same image. The systolic array can be used to perform the convolution on a color image where the rows of the systolic array are fed with pixels from the red, green, and blue frames. In order to prevent the cancellation of opposite gradients in different color frame when doing edge detection on color images, it is necessary to take the absolute value of the result from each row and add them together afterward. Figure 9 shows the modified systolic array. The systolic cell can be used to add the results from each row since it behaves as a simple serial floating point adder when the coefficient register is loaded with the value one. To allow the addition of the absolute value of the results, the systolic cells are made to be able to mask the sign bit of its operands. An input to the systolic cell is reserved to determine if the cell is used in absolute or normal mode.

IX. Conclusion

This paper presented the design of a pipeline architecture for a double precision floating point convolution systolic cell. Following a brief outline of the motivation that stimulated the design of this piece of hardware, an overview of the architecture was given. Further explanations described in greater detail the implementation of the three pipeline stages. Finally, an explanation was given on how this architecture could be adapted for color image processing.

X. Acknowledgment

The authors wish to acknowledge the financial support of CMC, FCAR, and NSERC.

References

1. J. Carayannis, P. Freedman, and A. Malowany, "An Integrated Programming Environment For A Generic Robotic Workcell", Proceedings of the Symposium of Manufacturing Application Languages, June, 1988, Winnipeg, Man., pp. 71-85.

2. M.D. Levine, *Vision in Man and Machine*, New York: McGraw -Hill Book Company, 1985.

3. McWhiter, J.G., Wood, d., Evans, R.A., McCanny, J.V. and McCabe, A.P.H., "Multibit Convolution Using A Bit Level Systolic Array", *IEEE Transaction on Circuits and Systems*, Vol. CAS-32, no. 1, Jan. 1985.

4. Kung H.T.and Song, S.W., "A Systolic 2-D Convolution Chip", Proceedings 1981 IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management, Nov. 1981, pp. 159-160.

5. Mimaroglu, T., "A High-Speed Two-Dimensional Hardware Convolver for Image Processing", Proceedings of the Pattern Recognition and Image Processing Conference, 1982, pp. 386-389.

6. Y. Boudreault and A. Malowany, "A VLSI convolver for a robot vision system", Proceedings of the Canadian Conference on Very Large Scale Integration, pp. 265-270, (1986).

7. J.F. Côté, C. Collet, D.D. Haule, and A.S. Malowany, "A High Performance Convolution Processor", Proceedings of the SPIE conference on Vision Communications and Image Processing, Cambridge, Mass., Nov. 1988.

8. M.E. Malowany and A.S. Malowany, "Making Curvature Estimates of Range Data Amenable to a VLSI Implementation", Proceedings of the SPIE conference on Vision Communications and Image Processing, Cambridge, Mass., Nov. 1988.