Efficient Implementation of Greyscale Morphological Filters

Donald G. Bailey

School of Engineering and Advanced Technology, Massey University
Palmerston North, New Zealand
D.G.Bailey@massey.ac.nz

***Abstract*—Morphological filters are often implemented using a series decomposition. This paper presents a parallel decomposition that is able to exploit separability. By maximising the reuse of hardware between the parallel filters, a novel computationally efficient filter structure may be derived. Results show that such filters may be implemented on a Virtex-5 FPGA with pixel clock rates approaching 1 GHz.**

I. Introduction

Morphological filtering is a commonly used image processing operation, where the contents of an image are filtered on the basis of shape and size. Mathematical morphology is based on set theory, with each operation transforming the set of object pixels by a structuring element (SE) [1]. The two basic morphological operations are erosion and dilation. On a greyscale image, I, erosion selects the minimum pixel value within the structuring element, S:

$$I \ominus S = \min_{i,j \in S} \{I[x+i, y+j]\} \tag{1}$$

Conversely, dilation selects the maximum value within the flipped structuring element:

$$I \oplus S = \max_{i,j \in S} \{I[x-i, y-j]\} \tag{2}$$

More complex morphological operations can be composed from combinations of erosions and dilations.

A. Decomposition

Large SEs may be decomposed into a series of smaller SEs and implementing the filters in series.

$$\begin{aligned} I \oplus (S_1 \oplus S_2) &= (I \oplus S_1) \oplus S_2 \\ I \ominus (S_1 \oplus S_2) &= (I \ominus S_1) \ominus S_2 \end{aligned} \tag{3}$$

An alternative decomposition is based on implementing filters in parallel and combining the results:

$$\begin{aligned} I \oplus (S_1 \cup S_2) &= (I \oplus S_1) \cup (I \oplus S_2) = \max(I \oplus S_1, I \oplus S_2) \\ I \ominus (S_1 \cup S_2) &= (I \ominus S_1) \cap (I \ominus S_2) = \min(I \ominus S_1, I \ominus S_2) \end{aligned} \tag{4}$$

These two types of decomposition are compared in Fig. 1.

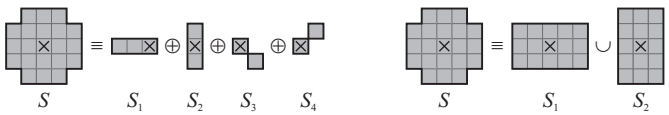

Fig. 1. Structuring element decompositions; left: series, right: parallel

Rectangular SEs are separable; a series decomposition gives a 1D column filter followed by a 1D row filter, significantly reducing the computation.

978-1-4244-8982-4/10/$26.00 ©2010 IEEE

B. Prior Work

Binary morphologic filters, because of their usefulness and relative simplicity, were some of the first image processing operations to be implemented on FPGAs [2, 3]. Their regular structure makes a streamed pipeline implementation attractive, and most FPGA based filter implementations use this structure.

Where decompositions have been used, they have generally been restricted to the series decomposition (see for example [4]). However, the size and shape of the SEs that can be created by series decomposition is limited.

Waltz developed a state machine based approach for morphological filtering that extended the concept of separability to enable efficient binary filtering with any arbitrary SE [5, 6]. The basic concept is to store all of the structural information required from the column processing as state information that could then be processed by the row processing to perform binary erosion and dilation. This work was extended to greyscale filtering [7], but the problem was too much state information is required to make it practical. Later work [8] overcame this problem using a parallel decomposition by decomposing the SE into separate rows.

C. Contributions of This Paper

This paper extends the work of Waltz, specifically in the context of FPGA implementation. The number of operations is further reduced through a systematic design procedure that exploits parallel decomposition using rectangular SEs.

The remainder of the paper is structured as follows. Section II presents series decompositions for 1D filters, minimising the logic required. The extension to 2D is described in section III. Particular focus is given on exploiting reuse between the composite parallel filters. Building on these principles, section IV presents the filter design procedure and analyses the complexity of the resultant filters. Section V compares the theoretical analysis with practical implementation on Spartan 3E and Virtex 5 FPGAs. Finally the results and limitations of this approach are discussed in section VI.

II. One Dimensional Filtering

Consider erosion with an 8×1 pixel structuring element. When processing a streamed input image, such a filter requires seven delays to give access to the previous seven pixels. Using a set of two-input minimum operations, the samples may be combined in a tree structure to reduce the propagation delay, as illustrated in Fig. 2.

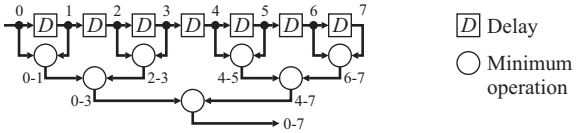

Fig. 2. An 8×1 erosion filter; a minimum tree reduces the propagation delay

A. Reduction of Computation

Several of the minimum operations are redundant. Those labelled "2-3", "4-5", and "6-7" can reuse the minimum from "0-1" with appropriate delays. Similarly the minimum labelled "4-7" can be produced by delaying "0-3" by four delays. The reduced form for the 8×1 structuring element is shown in Fig. 3. It corresponds to a serial decomposition of the SE with a series of two-pixel SEs, with the pixel in each successive SE spaced by the next increasing power of 2. When the SE is not a power of 2 wide, the number of delays within the last stage may simply be reduced to give a smaller window.

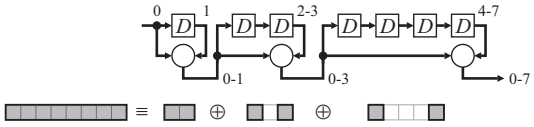

Fig. 3. Reduced form, from a serial decomposition of the structuring element

B. Transpose Form

The filter structure may be transposed by reversing the flow direction is reversed, and pick-off points are replaced by minimum operations and vice versa. This is demonstrated in Fig. 4 for the filters in Fig. 2 and Fig. 3.

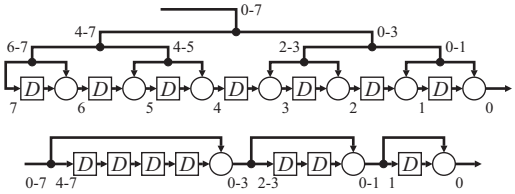

Fig. 4. Transpose forms of the original, the tree, and the reduced filters

In the transpose form, the numeric labels represent the distance (in delays) to the output rather than to the input. The computational requirements for the transpose form are the same as for the original because there is always the same number of minimum operations as pick-off points.

III. Two Dimensional Filtering

A two dimensional rectangular filter may be created by cascading a column filter and a row filter in series. The column filter may be formed by replacing the pixel delays with row delays (using circular memory or FIFO based row buffers).

Non-rectangular windows may be created by using a parallel decomposition. Consider the simple 5×5 "circular" SE from Fig. 1. The parallel decomposition consists of a 3×5 and a 5×3 filter. Note that the component 3×1 and 1×3 filters must be delayed by one column or row respectively to align the two rectangular SEs correctly. The direct parallel implementation is shown in Fig. 5, where *R* represents a row delay.

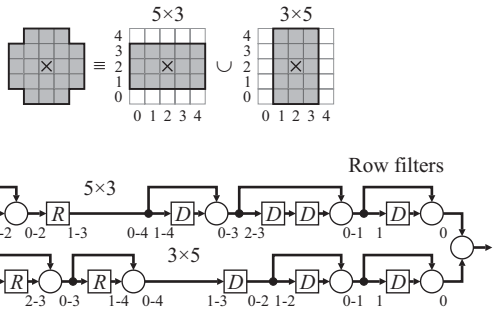

Fig. 5. Direct parallel implementation of a 5×5 circular erosion

A. Exploiting Reuse

Obviously this filter is not optimal. Seven row buffers are required to implement this filter, rather than the 4 normally required for a window that spans 5 rows. Any blocks in common to the parallel filters may be shared, requiring only a single instance. For example, in Fig. 5 the column filter block producing 0-1 is common to both windows, as is the last block of the row filters. The block producing the 0-2 signal in the 3-element column filter is the same as that producing the 0-4 signal in the 5-element filter. Swapping the 0-3 and 0-4 blocks of the 5-element column filter allow this block to be shared. Fig. 6 shows the results of merging the parallel filters and reusing components. This has reduced the filter to the minimum number of row buffers, delays, and minimum operations for this decomposition.

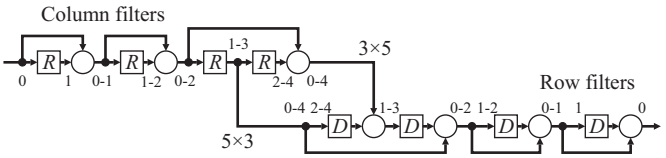

Fig. 6. The result of merging the parallel filters of Fig. 5

With larger filters, there are more parallel filters making it difficult to determine which combinations of blocks are best to reuse. Using the reduced structure for the column filters will tend to place the smaller blocks at the front of the filter, and using the transpose structure for the row filters will place the smaller blocks at the end of the filter. Such blocks are more likely to be in common with many of the parallel filters.

IV. Filter Design Procedure

This section describes a design procedure that exploits reuse. To illustrate, a 21×21 circular SE will be used.

1) Decompose the structuring element: Determine the set of rectangular SEs that combine to make the SE. For each rectangular filter the row and column spans are determined. As shown in Fig. 7, the 21×21 circular filter decomposes into seven parallel filters. Each of the component 1D filters is sorted in order from shortest to longest.

2) Design the shortest column filter: The design procedure from section II is used to produce the reduced form filter. The output is then delayed by an appropriate number of rows to align it correctly. For the 21×21 filter, the smallest column

filter is 7 elements, spanning rows 7-13. The corresponding filter (Fig. 8) consists of a 7 element filter consisting of three blocks, followed by 7 delays for alignment.

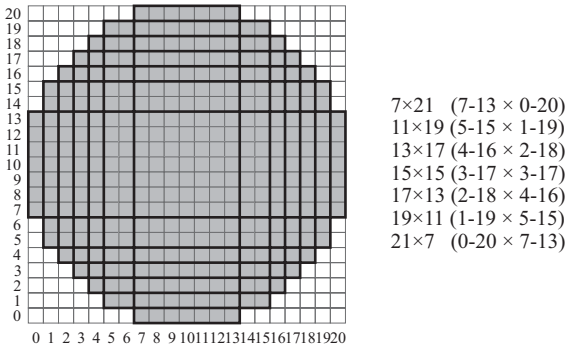

Fig. 7. Decomposition of a 21×21 circular structuring element

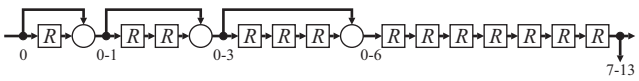

Fig. 8. Step 3 of filter design procedure builds the smallest filter

3) Add successive filters to the chain: In order of filter length, each column filter is added by picking off the point in the delay chain for the start of the span, and inserting a minimum operation at the point corresponding to the end of the span. In this example, the next filter has 11 elements, spanning rows 5-15. A pick-off after the 5th delay in the chain corresponds to the span 5-11. Adding two delays to the chain will give the span from 9-15. The minimum of these signals will give the span from 5-15. If the span difference in between successive filters is greater than that of the shortest filter, then a further minimum block may be required. For the 21×21 filter, the set of 7 parallel column filters is shown in Fig. 9.

4) Design the row filters: Steps 2 and 3 are repeated for designing the row filters. Note that it is not necessary that the row filters have the same sizes or spans as the column filters.

5) Convert row filters to transpose structure: This produces the set of row filters with injection points corresponding to each filter, as shown in the bottom row of Fig. 9.

6) Link corresponding column and row filters: This step combines the two components of each rectangular filter in series. In the case of the 21×21 circular filter, the 7-13 column filter is combined with the 0-20 row filter, and so on. The final filter structure can be seen in Fig. 9.

A. Complexity Analysis

In analysing the filter complexity, it will be assumed that each parallel filter added in step 4 can be made with only one additional minimum operation. Then, the number of row buffers required is one less than the vertical size of the structuring element. Similarly, the number of pixel delays is also one less than the width of the SE.

The number of minimum operations may be determined by considering the two parts of the row and column filter sets. The first part is the reduced filter required to span the narrowest filter. The number of minimum operations is determined by the logarithm of the minimum span.

The second part is the construction of the parallel filters. Here, each parallel filter pair (apart from the first one) requires three minimum operations. One is required in each of the column and row filters to create the component filters. Where the filters are connected, the column filter has a pick-off point, and the row filter, using the transpose structure, has an injection point with the third minimum operation.

The number of parallel filters depends on the size and shape of the SE; circular SEs will be considered here. The number of parallel filters is proportional to the SE radius. Starting from the top, each successive parallel filter spans two rows less than the previous filter until the 45° point is reached and the component filter becomes square. By symmetry, this represents half of the parallel filters – the total is therefore

$$n \approx 2r\left(1-\tfrac{\sqrt{2}}{2}\right) \tag{5}$$

where r is the radius of the circular SE. Since $r \approx N/2$ and each filter requires 3 minimum operations, the total number of minimum operations is therefore

$$N_{\min} \approx 3N\left(1-\tfrac{\sqrt{2}}{2}\right)+\log_2 w \approx 0.9N+\mathrm{O}(\log_2 N) \tag{6}$$

For small N, the resource requirements are approximately proportional to the diameter of the SE.

V. Implementation Results

Morphologic filters with circular SEs with sizes from 5×5 to 25×25 were implemented using Handel-C. Correct functionality was verified through simulation within the DK environment. The output was targeted to a Spartan 3E1200-4 and Virtex 5VLX110-3 FPGA, with the place and route performed by Xilinx ISE 9.1 (with effort set to medium).

The row buffers were implemented using a circular memory buffer in BlockRAM, controlled by a single counter. A register was used on both the input and output so that the

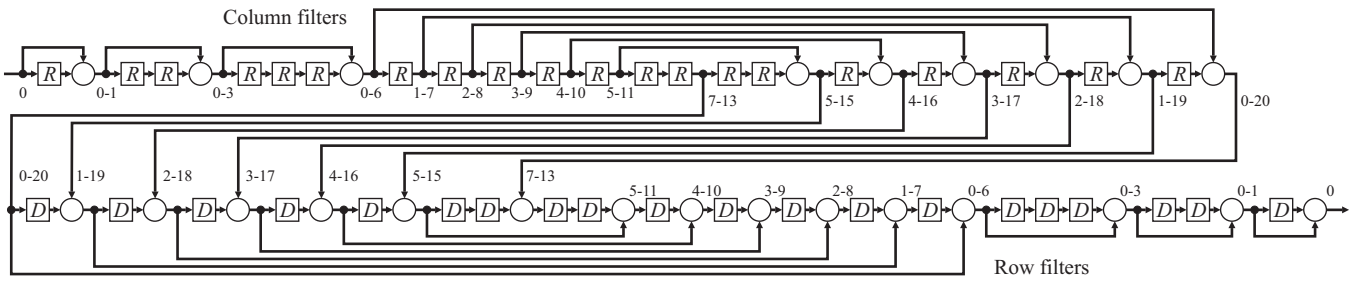

Fig. 9. The 21×21 circular structuring element, consisting of 7 parallel rectangular filters. Top row: column filters; bottom row: row filters

BlockRAMs would be pipelined, and not limit the timing.

Table I lists the results for the Virtex-5 and Table II for the Spartan 3. The first point of note is that the maximum operating speed is approximately constant, independent of the size of the SE. It appears that the propagation delay through the series of minimum operations is not the limiting factor.

TABLE I. IMPLEMENTATION RESULTS FOR VIRTEX-5

N	5	7	9	11	13	15	17	19	21	23	25
Flip flops	106	138	170	202	234	266	298	322	354	386	418
LUT-logic	136	184	216	264	264	312	312	360	408	408	456
LUT-shift register	1	1	1	1	1	1	1	9	9	9	9
LUT-route thru	30	52	37	89	53	126	24	164	68	77	102
Total LUTs	167	237	254	354	318	439	337	513	485	494	567
BlockRAMs	4	6	8	10	12	14	16	18	20	22	24
f_{max} (MHz)	967	923	1212	931	952	931	1202	952	924	924	965

TABLE II. IMPLEMENTATION RESULTS FOR SPARTAN 3E

N	5	7	9	11	13	15	17	19	21	23	25
Flip flops	90	122	154	186	218	250	282	306	338	370	402
LUT-logic	136	184	216	264	264	312	312	360	408	408	456
LUT-shift registers	1	1	1	1	1	1	1	9	9	9	9
LUT-route thru	5	7	9	11	11	13	13	15	17	17	19
Total LUTs	142	192	226	276	276	326	326	384	434	434	484
BlockRAMs	4	6	8	10	12	14	16	18	20	22	24
f_{max} (MHz)	579	588	572	565	572	565	567	587	575	572	587

The Virtex-5 implementation uses 16 flip-flops more than the Spartan 3E. It is unsure what these are required for, but it is suspected that they may be associated with the streamed I/O. Eight flip-flops are required for each delay, and for each pipeline register associated with the row buffers. The remaining flip-flops are used by the counter for the row buffer and for the control logic built implicitly by Handel-C. The reduction in the number of flip-flips for $N \geq 19$ is when the minimum window changes from 5 pixels to 7 pixels wide. The string of delays the row filter is replaced by a shift register.

Each minimum operation uses 16 LUTs: eight for the comparison, and eight to select the minimum value. The remaining 24 LUTs implement the counter for the row buffer and control logic. The number of block-RAMs required is as expected: one smaller than the diameter of the window.

To summarise the results, the resources required by the implementation match what was expected for each filter size.

VI. DISCUSSION AND CONCLUSIONS

The parallel decomposition has a number of advantages over the more commonly used series decomposition. It is able to handle a wider range of SE shapes and sizes. Series decomposition requires the SE be convex, and have 180° rotational symmetry. Neither of these are constraints on the parallel decomposition.

All of the row buffers are of identical length for the parallel decomposition. For a series decomposition, diagonal SEs require either different length row buffers, or extra delays inserted to enable a single length row buffer to be used. Consequently, the control logic overhead of the parallel decomposition presented here is reduced.

A limitation of the parallel decomposition is that the number of operations grows more quickly with filter size than that required by a series decomposition (if one exists). For small filters the lower overhead means that the resource requirements of the parallel decomposition are lower than that for the series decomposition. However, this advantage is lost for larger diameter filters. In spite of this, the resource requirements grow proportionally with the diameter of the SE.

As identified earlier, if the smallest component filter span is one pixel then additional minimum operations are required to produce the component filters over that analysed earlier. The number of additional minimum operations may be reduced by not completely merging the parallel filters. For column filters, this will require additional row buffers, so is probably not worthwhile. However for row filters, this would only require a few extra delays, with minimal additional resource costs.

The analysis presented here was for an erosion filter. All of the comments also apply for dilation, with the exception that a maximum operation is used rather than a minimum operation, and that the SE is flipped both horizontally and vertically (this is equivalent to rotating by 180°). Many SEs are rotationally symmetric anyway. If so, the same circuitry may be used for both erosion and dilation simply by inverting the data on both the input and the output. The modest resource requirements enable more complex filters (openings, closings, and other composite filters) to be readily implemented on an FPGA.

To summarise, this paper has discussed the parallel decomposition of SEs for morphological filtering. It is shown how this can lead to a separable implementation. A new filter design procedure has been described that both simply and efficiently merges the parallel filters into a single composite structure. The resources required by the composite filter are approximately proportional to the diameter of the SE. Finally, it was demonstrated that the resulting filter can be implemented efficiently on an FPGA. The clock rate of the resulting implementation is independent of the filter size for the filters tested (up to 25 pixels diameter).

REFERENCES

[1] J. Serra, "Introduction to mathematical morphology," *Computer Vision, Graphics and Image Processing,* vol. 35, pp. 283-305, 1986.

[2] S. Hemmert, B. Hutchings, and A. Malvi, "An application specific compiler for high-speed binary image morphology," in *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'01)*, Rohnert Park, California, 2001, pp. 199-208.

[3] B.A. Draper, J.R. Beveridge, A.P.W. Bohm, C. Ross, and M. Chawathe, "Implementing image applications on FPGAs," in *16th International Conference on Pattern Recognition*, Quebec, 2002, pp. 265-268.

[4] C. Clienti, M. Bilodeau, and S. Beucher, "An efficient hardware architecture without line memories for morphological image processing," in *10th International Conference Advanced Concepts for Intelligent Vision Systems*, Juan-les-Pins, France, 2008, pp. 147-156.

[5] F.M. Waltz, "Separated-kernel image processing using finite-state machines (SKIPSM)," in *Machine Vision Applications, Architectures, and Systems Integration III*, Boston, Massachusetts, 1994, pp. 386-395.

[6] F.M. Waltz and H.H. Garnaoui, "Application of SKIPSM to binary morphology," in *Machine Vision Applications, Architectures, and Systems Integration III*, Boston, Massachusetts, 1994, pp. 396-407.

[7] F.M. Waltz, "Application of SKIPSM to grey-level morphology," in *Machine Vision Applications, Architectures, and Systems Integration III*, Boston, Massachusetts, 1994, pp. 428-435.

[8] F.M. Waltz and J.W.V. Miller, "Gray-scale image processing algorithms using finite-state machine concepts," *Journal of Electronic Imaging,* vol. 10, pp. 297-307, 2001.