An Efficient Hardware Architecture without Line Memories for Morphological Image Processing

Christophe Clienti, Michel Bilodeau, Serge Beucher

An efficient hardware architecture without line memories for morphological image processing

Christophe Clienti, Michel Bilodeau and Serge Beucher

MINES Paristech
CMM - Centre de morphologie mathmatique, Mathmatiques et Systèmes
35 rue Saint Honoré - 77305 Fontainebleau cedex, France.

Abstract. In this paper, we present a novel hardware architecture to achieve erosion and dilation with a large structuring element. We are proposing a modification of HGW algorithm with a block mirroring scheme to ease the propagation and memory access and to minimize memory consumption. It allows to suppress the needs for backward scanning and gives the possibility for hardware architecture to process very large lines with a low latency. It compares well with the Lemonnier's architecture in terms of ASIC gates area and shows the interest of our solution by dividing the circuit area by an average of 10.

1 Introduction

For more than 40 years, the use of Mathematical Morphology for image processing has been constantly growing in many domains such as medical imaging, computer vision multimedia application or security [1].

Real-time image processing with large images and big kernels or structuring elements could be very time-consuming. A possibility to reduce the computational power required by architectures is to use the well-known kernel decomposition. For example, if we consider an erosion with a $N \times N$ structuring element, the complexity in terms of min operations to find the minimum in a kernel will be $\mathcal{O}(N^2)$. Now, if we apply $N/2$ times a 3×3 structuring element, the complexity will be $\mathcal{O}(N)$ to get the same result.

When using simple structuring elements such as rectangles, rhombuses or octagons, the decomposition will be done by using a segment structuring element successively applied on images with a neighborhood operation (an erosion for instance). Figure 1 shows an example of a decomposition using three segments to produce an hexagon, a pixels access for 2D or 3D images must be done using the Bresenham coordinate generation [2]. This type of decomposition used with recursive implementation of erosions or dilations allows to approximate many types of 2D structuring elements [3]. Applications are numerous and we can cite an algorithm using segment structuring elements to estimate the skew angle of scanned documents [4].

The Interest of segment structuring elements no longer needs to be proved, so it could be useful to design fast morphological operators to take advantage of

Fig. 1. Structuring element composition example

decompositions. The first part of the paper is dedicated to existing algorithms designed to work with segment structuring elements, the second part shows our modifications to reduce memory requirements and latency, the third part presents our architecture and a comparison in terms of memory consumption is made between our dedicated architecture and the existing ones.

2 Existing algorithms description

Two main recursive algorithms exist to compute erosion or dilation along the discrete lines of an image with centered segment structuring elements. Both use pixels min/max propagations in a forward and backward way. These algorithms are often described in the literature and only a brief description will be made here.

In both approaches we consider an image line f of size M, with an orientation θ and a centered segment structuring element of size k.

2.1 Lemonnier's Algorithm

This algorithm described in [5] works for dilation only, but erosion could be achieved using the duality property between erosion and dilation.

Two scans are needed to process one line. Firstly, a recursive forward scan on f is performed considering maxima propagation every k pixels to produce the line h. Secondly, a recursive backward scan on h is made to spread maxima in the other direction to get a centered structuring element.

The Lemonnier's algorithm is interesting because only two max operators are used for processing each pixel, but the high number of *if statements* must be taken into account and an implementation on SIMD processors would not be well-suited. Lemonnier's algorithm could be mapped to data flow dedicated architectures as described in figure 2.

Both passes of the algorithm are synthesized in two pipeline stages with two line memories in between. The latter are needed to store first stage pixels of line h_n in a forward way in the first memory, and at the same time to read pixels of line h_{n-1} of the second line memory in a backward way by the second stage. When the processes terminate on each stage, line memories are swapped to store the result of line h_{n+1} of the first stage and to read line h_n by the second stage. However, the double buffering principle introduces one line latency which could be problematic for hard real-time systems. Moreover output lines are reverted due to a backward scan of the second stage which raises a synchronization constraint with other systems.

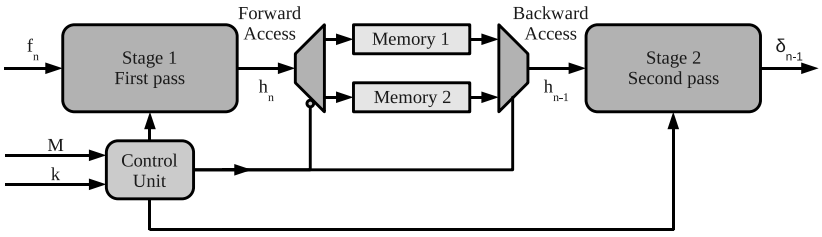

Fig. 2. Functional view of Lemonnier's architecture

2.2 Van Herk - Gil - Werman's Algorithm

This algorithm, described by M. van Herk [6] and by Gil - Werman [7] in different papers, operates in three stages. Firstly a propagation of f in a forward way is done using equation 1. We notice that the number of maximum operations does not depend on the value of k (odd). We will focus on dilation but the erosion case can be processed simply by replacing maxima with minima.

$$g(x) = \begin{cases} f(x) & \text{if } x \bmod k = 0 \\ max(g(x-1), f(x)) & \text{otherwise} \end{cases}, x = 0, 1, \cdots, M-1 \qquad (1)$$

Secondly a propagation of f in a backward way is done using equation 2.

$$h(x) = \begin{cases} f(x) & \text{if } x \bmod (k-1) = 0 \\ max(h(x+1), f(x)) & \text{otherwise} \end{cases}, x = M-1, \cdots, 1, 0 \quad (2)$$

Finally, the dilation is performed by merging g and h using equation 3. This algorithm uses only three maximum operations per pixel to dilate a line with any size of segment structuring elements.

$$\delta_f(x)) = max\left(g\left(x + \frac{k}{2}\right), h\left(x - \frac{k}{2}\right)\right), x = 0, 1, \cdots, M-1 \qquad (3)$$

These equations do not take into account two major padding problems. The first one is the access from outside of line f in equation 3. The second one, and maybe the most important, is to compute equations 1 and 2 when M is not a multiple of k. One possibility is to add padding before and after f to satisfy access outside of f and to get a line size multiple of k. Such a solution would not be feasible because it adds stall cycles.

Figure 3 presents an example of padding for a line of size 19 and a centered structuring element of size 7. The A padding is added to obtain a line size multiple of k, and the B padding is added to prevent outside access. The left B padding size must be equal to $\lfloor k/2 \rfloor$ and the right B padding added to A

padding must also be equal to $\lfloor k/2 \rfloor$. The A padding size (PSA) is defined by the following equation:

$$\text{PSA} = (k - (M-1) \bmod k) - 1$$

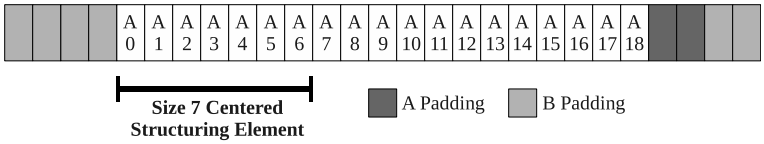

Fig. 3. Padding example needed to merge first and second pass of HGW algorithm

Obviously equations 2 and 3 must be rewritten in order to emulate the different types of padding.

$$h(x) = \begin{cases} f(x) & \text{if } x \bmod (k-1) = 0 \\ f(x) & \text{if } x = M-1 \\ max(h(x+1), f(x)) & \text{otherwise} \end{cases} \tag{4}$$

$$\delta_f(x) = \begin{cases} g(x + \frac{k}{2}) & \text{if } x - \frac{k}{2} < 0 \\ max(g(M-1), h(x - \frac{k}{2})) & \text{if } M \leq x + \frac{k}{2} < M + \text{PSA} \\ h(x - \frac{k}{2}) & \text{if } x + \frac{k}{2} \geq M + \text{PSA} \\ max\left(g\left(x + \frac{k}{2}\right), h\left(x - \frac{k}{2}\right)\right) & \text{otherwise} \end{cases} \tag{5}$$

The complete HGW algorithm including padding emulation management is described in algorithm 1.

Even if the number of cases in equations 4 and 5 grows up due to padding emulation, this is not an obstacle for a SIMD implementation. Contrary to Lemonnier's algorithm, cases do not depend on line values, but only on array indexes. In this way, these equations could be implemented using multiple loop instead of using *if statement* inside a larger loop.

Designing a dedicated architecture using this algorithm without modifications is not efficient because of multiples line memories to use (twice more than Lemonnier's algorithm design). Figure 4 shows a functional diagram of such an architecture. We notice that we need to revert f to compute h, but we have to reverse also g because of synchronization constraints in the merge unit. The latter also embed a memory of size k to merge properly g and h. So a naive design for this algorithm uses four memories of size M and one memory of size k which is not optimal comparing to the Lemonnier's algorithm design. Moreover, this implementation suffers also from mirrored output lines, and from a one line latency.

Algorithm 1 Dilation using HGW algorithm

Require: f, k, M
Ensure: δ_f

```
PSA:=(k-(M-1) mod k)-1

for x from 0 to M-1 do
  if x mod k=0 then
    g[x]:=f[x]
  else
    g[x]:=max(g[x-1],f[x])
  end if
end for

for x from M-1 downto 0 do
  if x=M-1 then
    h[x]:=f[x]
  else if (x+1) mod k =0 then
    h[x]:=f[x]
  else
    h[x]:=max(h[x+1],f[x])
  end if
end for

for x from 0 to M-1 do
  if x-k/2<0 then
    δf[x]:=g[x+k/2]
  else if x+k/2 ≥ M then
    if x+k/2 < M+PSA then
      δf[x]:=max(g[M-1],h[x-k/2])
    else
      δf[x]:=h[x-k/2]
    end if
  else
    δf[x]:=max(g[x+k/2],h[x-k/2])
  end if
end for
```

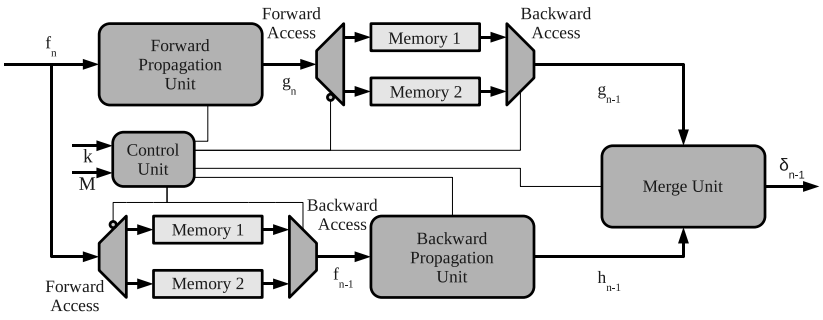

Fig. 4. Functional view of HGW architecture

3 Algorithm proposition

3.1 HGW modification

We notice that propagations to produce g and h in HGW algorithm are almost the same even if the scan way is opposite.

We define a block as a group of pixels of size equal to k. If we mirror each block of f to produce f' we can construct h' using g propagation equation (1) with the same scan way. To reconstruct original h, we just need to mirror each h' block. Figure 5 shows how to mirror f to produce f' regarding a specific structuring element of size 7.

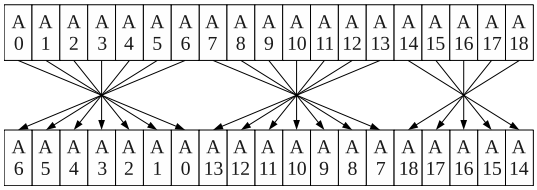

Fig. 5. Block mirroring to produce f' using size 7 structuring element

The benefits of this modification mainly affect dedicated architecture because we use only memories of size k instead of memory of size M. An implementation using a general purpose processor does not really make sense here because image lines could be completely stored in cache memory, so block mirroring is not useful.

Figure 6 proposes a functional view of a data flow dedicated architecture using the principle of block mirroring to compute HGW algorithm.

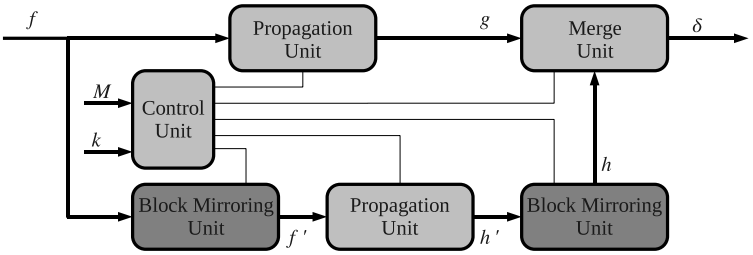

Fig. 6. Block mirroring to produce f' using size 8 structuring element

The proposed modification gives the opportunity to remove the need of two propagation policies to compute g and h. We use only equation 1 in both cases.

3.2 Block Mirroring Unit

If we consider the line size as a multiple of the structuring element size, the data flow block mirroring could be realized using two memories of size k, one to write a block b in a forward way and one to read the block $b-1$ in a backward way. Memories are swapped every time a pixel arrives from a new block.

But considering only blocks of size k is not satisfactory because it is necessary to introduce padding when the last block has not got the same size as others. We need a specific management of the last block mirroring not to introduce stall cycles in the data flow architecture.

The principle is to use read-first memories and reverted address counters to store and load pixels of the last blocks while freezing the memory swap. Figure 7 shows for an image composed of two lines, how pixel memories are managed to implement block mirroring considering the last block size inferior to the structuring element size. The two first states have already been described and the figure presents only the key part of the specific last block management. It is worth mentioning that the memory state 3 is guaranteed thanks to the use of read first memories.

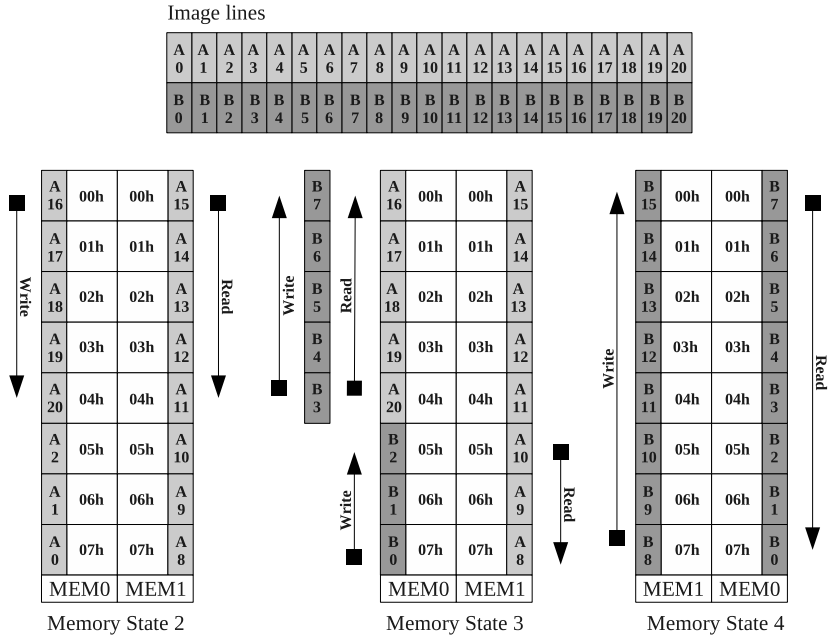

Fig. 7. Block mirroring memory management

3.3 Merge Unit

The merge unit is the same as the one used in the original HGW algorithm hardware implementation. This unit is in charge of delaying g pixel flow and compute the maximum with h. The delay line used is needed to perform access to pixels $g(x+k/2)$ and $h(x+k/2)$ before computing the maximum. The functional view of the architecture shows us that pixels from h arrive two blocks after pixels from g, so we only need to delay g with one delay line of size k. Moreover the unit also takes care of the double padding policy described in HGW algorithm equation 5. Figure 8 presents the data flow used and produced by the merge unit considering a line of size 19 and a structuring element of size 5.

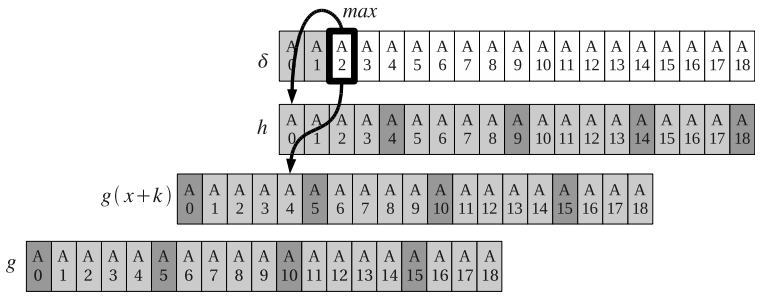

Fig. 8. Delay unit data flow

3.4 Performances

The system was synthesized on a FPGA Virtex 4LX60. The design was optimized to reach a 200 Mhz frequency and the system uses 3 BlockRam of 18Kbits and 700 Slices. The maximum structuring element size could be 1023 with a line length of 65535 pixels. The processing time with 512×512 images is 1.3 ms for a horizontal segment centered structuring element of size k. The latency is proportional regarding k and is defined as follows:

$$Latency = 3 \cdot \frac{k}{2} \cdot T_{clk} \text{ where } T_{clk} \text{ is the system period}$$

The latency reduction permits to get results faster, which is important if many operators are pipelined or if many passes are necessary to process an application.

Memories represent a large part of the design, if knowledge on the structuring element maximum size is known, memories size could be reduced because the design is image size independent in terms of memory consumption. This is a key point of this architecture allowing dramatic reduction of the ASIC surface. For example the size of a circuit could be divided by an order of magnitude

compared to Lemonnier's algorithm implementation or the standard HGW algorithm implementation. Figure 9 shows for different sizes of maximum allowable structuring elements the circuit area in terms of gates. The circuit area is obtain using post place and route synthesis results.

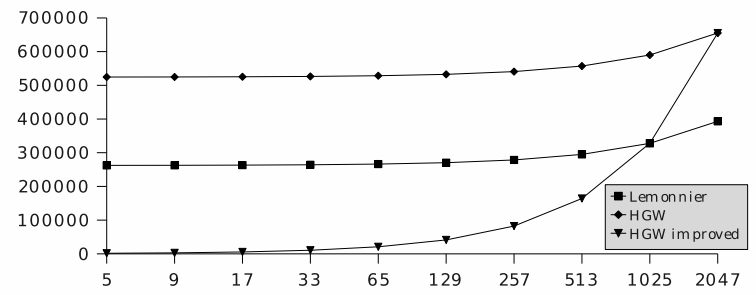

Fig. 9. Number of gates versus the maximum allowable structuring element

A benchmark is proposed between our solution and two different SSE2 software implementations running on an AMD Opteron 280. The first software implementation use the classical approach to compute HGW algorithm because our block mirroring scheme is only related to hardware implementation. The SSE2 version is optimized to process vertical structuring elements, so horizontal structuring elements must be computed by doing 90 degrees rotations of images using SSE2 instructions. The second software implementation uses a logarithmic decomposition[8].

The figure 10 shows timing results for different sizes of segment structuring elements. Performances of our hardware solution is similar to a SSE2 software implementation of the HGW algorithm on a generic purpose processor running at a clock frequency twelve times faster than our prototype clock frequency. If needed by applications, multiple instances of our architecture could be generated to process multiple lines in parallel, thus improving drastically the processing time. The Logarithmic decomposition is interesting but could not be easily ported to hardware because of numerous line scans depending of the decomposition step. It is necessary to expect a high number of stages using block mirroring scheme and embedding memory which size corresponds to maximum structuring element size. Moreover some stages could not be used because the decomposition could be smaller than the number of stages, on the contrary the decomposition could not be fitted entirely into the number of stages, imposing to do multiple passes in the architecture.

4 Conclusion and future work

Our architecture is really efficient for small embedded SoC systems which need to compute image processing algorithms using mathematical morphology with

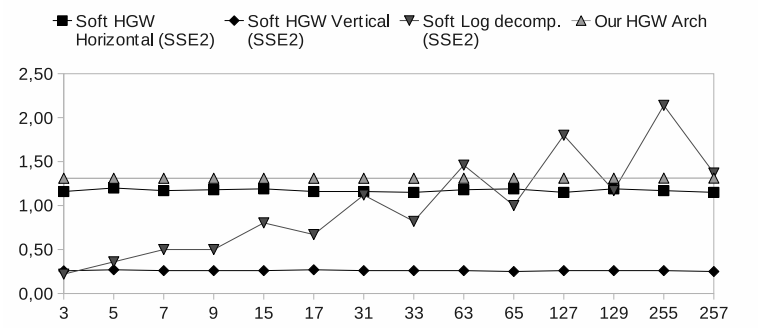

Fig. 10. Processing time versus structuring element width

cost and power consumption constraints. In the past, architectures based on Lemonnier's algorithm were the most efficient to compute erosion and dilation with line structuring elements in a recursive way. We propose a new approach based on a modification of the HGW algorithm which produces an architecture without line memories and have a low latency. Line size has almost no impact on the size of the circuit and the architecture could change the size of the structuring element between two lines without introducing stall cycles which gives the opportunity to take into account perspective problems. Our present work aims to build a criterion to automatically change the size of the structuring element between lines.

References

1. Soille, P.: Morphological Image Analysis: Principles and Applications. Springer-Verlag New York, Inc. 2003.
2. Bresenham, J.: Algorithm for computer control of digital plotter. IBM System Journal, 4:25–30, 1965.
3. Soille, P., Breen, Edmond J. and Jones, R.: Recursive Implementation of Erosions and Dilations Along Discrete Lines at Arbitrary Angles. IEEE Trans. Pattern Anal. Mach. Intell. 1996, Vol 18, Number 5, 562–567.
4. Najman L.: Using mathematical morphology for document skew estimation. SPIE, 5296,182–191, 2003.
5. Lemonnier, Fabrice and Klein, Jean-Claude: Fast dilation by large 1D structuring elements, IEEE Workshop on Nonlinear Signal and Image Processing, Neos Marmaras, Halkidiki, Greece, June 20–22, 1995, 479–482, 5229.
6. Van Herk, M.: A fast algorithm for local minimum and maximum filters on rectangular and octagonal kernels. Pattern Recognition Letters, 1992, Vol 13, Number 7, 517–521.
7. J. Gil and M. Werman: Computing 2-D Min, Median, and Max Filters, IEEE Trans. Pattern Anal. Mach. Intell.,1993, Vol 15, Number 5, 504–507.
8. R. van den Boomgaard and D.A. Wester: Logarithmic shape decomposition, in Aspects of Visual Form Processing, C. Arcelli, L.P. Cordella, and G. Sanniti di Baja (Eds.), World Scientific Publishing Co.: Singapore, Capri, Italy, 1994, pp. 552-561.