Speeding-up Mathematical Morphology Computations with Special-Purpose Array Processors

Alberto Broggi*
Dipartimento di Ingegneria dell'Informazione
Università di Parma
Parma, ITALY, I-43100

Abstract

The first part of this paper will analyze the computational complexity of the implementation of Mathematical Morphology operations on three different architectures: general-purpose serial systems, pipeline systems, and cellular systems. For each considered architecture, a different computing technique is devised, exploiting the specific system characteristics, and obviously reaching different throughputs. The second part will present an efficient algorithm for the computation of morphological operations (based on generic non-convex structuring elements) on cellular architectures, taking as an example the PAPRICA system. Finally some considerations on the optimization of the instruction set will conclude the paper.

1 Introduction

Mathematical Morphology [24, 32, 16] concerns the study of shape using the tools of set theory, and it is commonly and efficiently used in image analysis tasks. Mathematical morphology has been extensively used in low-level image processing and analysis applications, since it allows to filter and/or enhance only some characteristics of objects, depending on their morphological shape. A lot of tutorials can be found in the literature, such as in [16, 32, 45, 24, 40, 38, 42].

Within the mathematical morphology framework, a binary image A is defined as a subset of the two-dimensional Euclidean space E^2 $(Z \times Z)$:

$$A = \{a = (a_i, a_j) \mid a_i, a_j \in Z\} \tag{1}$$

Three monadic transforms are defined on the generic image A: *complement* $(\cdot)^c$, *reflection* $\breve{(\cdot)}$, and *translation* $(\cdot)_t$: respectively,

$$A^c \triangleq \{x \in E^2 \mid x \neq a, \text{ for every } a \in A\} \tag{2}$$

$$\breve{A} \triangleq \{x \in E^2 \mid x = -a, \text{ for some } a \in A\} \tag{3}$$

$$(A)_t \triangleq \{x \in E^2 \mid x = a + t, \text{ for some } a \in A\} \tag{4}$$

Moreover, following the definitions given in [16], some dyadic operators between sets are: *dilation* $\oplus$, *erosion* $\ominus$, *opening* $\circ$, *closing* $\bullet$, and *matching* $\oslash$: respectively,

$$A \oplus B \triangleq \{x \in E^2 \mid x = a + b, \text{ for some } a \in A, b \in B\} \tag{5}$$

$$A \ominus B \triangleq \{x \in E^2 \mid x + b \in A, \text{ for every } b \in B\} \tag{6}$$

$$A \circ B \triangleq (A \ominus B) \oplus B \tag{7}$$

$$A \bullet B \triangleq (A \oplus B) \ominus B \tag{8}$$

$$A \oslash Q \triangleq (A \ominus Q_1) \cap (A^c \ominus Q_0) \ , \tag{9}$$

where $B \in E^2$ and Q is a couple $Q = (Q_0, Q_1)$, where both $Q_0, Q_1 \in E^2$, with the constraint that $Q_0 \cap Q_1 = \emptyset$. In the previous expressions A is the image to be processed, while B represents the *structuring element*, namely another subset of E^2 whose shape parametrizes each operation.

Moreover, it is well-known that if a structuring element B is *convex*, it can be expressed with a chain of dilations of simpler sets:

$$B = \Big(\big((B_1 \oplus B_2) \oplus B_3\big) \oplus \dots\Big) \oplus B_n \tag{10}$$

In the literature a lot of different special-purpose architectures has been developed in order to exploit the special nature of the elaboration performed by morphological operators. A considerable part of them exploits spatial parallelism following the SIMD classification, in which a single pixel is elaborated by its own processing element. The main problem of this kind of architectures lays in the interconnecting topology, which limits the size of practical structuring elements. Some other architectures that have been investigated are formed by a number of specialized processors that operate in pipeline on pixel values. Furthermore, a

*This work was partially supported by CNR Progetto Finalizzato Trasporti under contract number 91.01031.PF93. The author can be reached at the following e-mail address: broggi@CE.UniPR.IT

lot of work has been devoted to the speed-up of morphological operations on general-purpose serial architectures, both using special data representations and performing efficient structuring elements decompositions.

The following section will present and discuss the implementation of mathematical morphology operations on these three different kinds of architectures, showing the computing algorithms as well as their main performance indexes, together with some examples. Section 3 will focus on special-purpose cellular processors and on the definition of an algorithm for the decomposition of generic structuring elements on such systems, addressing the problem of optimizing their Instruction Set. Finally section 4 will draw some conclusions, showing the future possible improvements.

2 Hardware platforms

In this section the application of mathematical morphology operators on three different kinds of architectures will be investigated, starting from general-purpose serial architectures, covering pipeline systems, and finally presenting and discussing the philosophy of cellular processors. The efficiency of the algorithms used to compute morphological operations depends strongly on the particular hardware architecture; thus, for each considered system, a natural technique for computing a dilation will be presented. Moreover, the performances depend on different parameters, which will be analyzed and compared, showing some speed-up methods explained by some examples. Generally, both in the case when the use of a large structuring element is not efficient (on serial systems), or when it is impossible (due to hardware constraints in small-neighborhood based parallel systems), the solution to the problem of structuring element decomposition becomes of basic importance. The decomposition algorithms are based on different optimality criteria [30] in order to exploit the special hardware characteristics of each system. It is anyway important to remember that in the literature some other decomposition criteria, such as the decomposition of binary images for object recognition purposes [28], are discussed.

The following sections will present some mathematical morphology computing techniques, based on different hardware platforms, and will devise some speed-up methods mainly based on structuring elements decomposition.

2.1 General-purpose serial architectures

In general-purpose systems, when used to run morphological image analysis tasks, there is no upper bound to the size of structuring elements: in fact, using a bit-mapped representation of images, during the computation of the new value of an image pixel the processing element can access the values stored in every other image pixel within a constant time. On the other hand, the computational complexity of serial morphological operations depends on the number of elements which form the morphological operands. As an example, the computation of $A \oplus B$ requires one vector sum and one logical union for each couple of elements $a \in A$ and $b \in B$, and thus

$$\mathcal{C}(A \oplus B) = \#(A) \times \#(B) \ , \tag{11}$$

where $\mathcal{C}(\cdot)$ indicates the computational complexity of a given operation, and $\#(\cdot)$ represents the number of elements in a set. Some speed-up methodologies, based on different optimality criteria, have been developed for serial systems [30, 40, 8, 15, 39, 38]; they are based on the decomposition of convex structuring elements into a chain of dilations of simpler sets. The optimization criteria range from the minimization of the total number of computations (the one considered here), to the minimization of the number of decomposing sets, and to the minimization of the total number of elements in the decomposing sets. Although in [43] it has been proved that all convex structuring elements are algorithmically decomposable, unfortunately there is no answer to the problem of optimizing the automatic partitioning of non-convex sets into unions of convex ones for further decomposition.

Using the well-known visual representation of morphological sets, in dilation $R = I \oplus B$, where

$$I = \begin{array}{|c|c|c|c|c|}\hline & & \bullet & \bullet & \bullet \\ \hline & \bullet & \bullet & \bullet & \bullet \\ \hline & \bullet & \bullet & & \\ \hline \bullet & \bullet & \bullet & \bullet & \\ \hline & \bullet & \bullet & & \\ \hline \end{array} \quad \text{and} \quad B = \begin{array}{|c|c|c|c|c|c|c|c|c|}\hline & & \bullet & \bullet & \bullet & & & & \\ \hline \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ \hline \end{array} \ , \tag{12}$$

the number of vector sums and logical unions, according to equation (11), is given by $\#(I) \times \#(B) = 15 \times 12 = 180$.

Using the following decomposition, the structuring element B can be expressed as the dilation between subsets B_1 and B_2:

$$B = B_1 \oplus B_2 = \begin{array}{|c|c|c|}\hline \bullet & \bullet & \bullet \\ \hline \end{array} \oplus \begin{array}{|c|c|c|c|c|c|c|c|c|}\hline & & & \bullet & & & & & \\ \hline & \bullet & & & \bullet & & & \bullet & \\ \hline \end{array} \ ; \tag{13}$$

using the *chain rule* property [16],

$$\begin{aligned} R &= I \oplus B = I \oplus (B_1 \oplus B_2) = \\ &= (I \oplus B_1) \oplus B_2 = R' \oplus B_2 \ , \end{aligned} \tag{14}$$

where $R' \triangleq I \oplus B_1$. The number of sums required to compute

$$R' = I \oplus B_1 = \begin{array}{|c|c|c|c|c|c|c|}\hline & & \bullet & \bullet & \bullet & \bullet & \bullet \\ \hline & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ \hline & \bullet & \bullet & \bullet & \bullet & & \\ \hline \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \\ \hline & \bullet & \bullet & \bullet & \bullet & & \\ \hline \end{array} \tag{15}$$

is given by $\#(I) \times \#(B_1) = 15 \times 3 = 45$, while the number of sums required to complete the elaboration

$(R = R' \oplus B_2)$ is given by $\#(R') \times \#(B_2) = 25 \times 4 = 100$. Thus, the decomposition shown in the example, while requiring two dilations instead of one, decreases the total number of computations from 180 to 145.

The decomposition $B = B_1 \oplus B_2$ in order to offer computational savings when applied to $I \oplus B$ must obviously lead to the following relation:

$$\begin{aligned} \mathcal{C}(I \oplus B) \quad > \quad & \mathcal{C}\Big((I \oplus B_1) \oplus B_2\Big) = \\ & = \mathcal{C}(I \oplus B_1) + \mathcal{C}(R' \oplus B_2) \ . \end{aligned} \quad (16)$$

Considering the *overlapping* $\mathcal{O}(\cdot,\cdot)$ between two subsets A and B as a function $E^2 \times E^2 \mapsto Z$, defined as the number of equal elements in the set

$$C = \bigcup_{i=1}^{\#(A)} \bigcup_{j=1}^{\#(B)} \Big(a_i + b_j\Big) \ , \quad \text{with } a_i \in A, b_j \in B \ , \quad (17)$$

namely

$$\mathcal{O}(A,B) = \#(A) \times \#(B) - \#(A \oplus B) \ , \quad (18)$$

after few algebraic manipulations, the first part of relation (16) becomes:

$$\frac{\mathcal{O}(I,B_1)}{\#(I)} > \frac{\#(B_1)}{\#(B_2)} + \frac{\mathcal{O}(B_1,B_2)}{\#(B_2)} \ , \quad (19)$$

and, using expression (18),

$$\frac{\#(B) - \#(B_1)}{\#(B_2)} > \frac{\#(R')}{\#(I)} \ . \quad (20)$$

Equations (19) and (20) show that a decomposition is as much efficient (i.e. decreases the total number of computations) as the increment in the number of elements between the original image I and the set obtained by the first dilation R' is small; thus B_1 should be composed by a minimum number of elements with respect to B_2. Moreover, the overlapping between B_1 and B_2 should be kept small, and preferably should be zero.

On the other hand, the *logarithmic decomposition* [8, 44, 38] is another efficient method to speed-up morphological operations on large convex structuring elements. Such a decomposition minimizes the total number of points in the chain of decomposed sets, but it has the disadvantage to be of use only for convex structuring elements.

A generic non-convex structuring element B, at least in theory, can be decomposed as

$$B = B_1 \cup B_2 \cup ... \cup B_m \ , \quad (21)$$

where each B_i, $i = 1, ..., m$, is a convex set which can be further decomposed as shown before. Thus the computational complexity of a generic dilation $I \oplus B$ is given by:

$$\begin{aligned} \mathcal{C}(I \oplus B) \ &= \ \mathcal{C}\Big[(I \oplus B_1) \cup (I \oplus B_2) \cup ... \cup (I \oplus B_m)\Big] \\ &= \ \sum_{i=1}^{m} \mathcal{C}(I \oplus B_i) \ . \end{aligned} \quad (22)$$

In general-purpose serial architectures, the word parallelism range from 16 to 64 bits; when employed for binary morphological operations, this data parallelism is not fully exploited. In fact, using a $2^n \times 2^m$ pixels image, each vector sum involves only $n+m+2$ bits[1], while each logical union operates on a single bit. Some special data packaging [40] can be used in order to operate in parallel on multiple image pixels, processing simultaneously p adjacent binary pixel values, allocated on the same word of parallelism $p \cdot (m+n+2)$. With this solution the computational complexity still depends on the number of items in the structuring element, but now it is inversely proportional to the hardware word parallelism.

Some further application-dependent speed-up can be achieved when dealing with small decomposed structuring elements (such as 3×3 square-shaped or 8-connected neighborhood, and 3×3 diamond-shaped or 4-connected neighborhood). In fact, when a dilation is iterated (for example in *seed propagation*), only the pixels belonging to the external contour of binary objects are involved: the list of these pixels can be obtained easily from the previous iteration, increasing the overall performances by applying the dilation only to border pixels (*contour-processing* [41]).

2.2 Parallel architectures

Two different parallel systems will be considered in the following: architectures which distribute a single elaboration among different Processing Elements (PEs), implementing *operational parallelism*, and architectures which subdivide the processing of the whole data structure among different PEs running simultaneously the same code, implementing *spatial parallelism*. The next sections will deal with systems involving these two different exploitations of parallelism: *pipeline* and *cellular* architectures.

2.2.1 Systems with operational parallelism

In *pipeline* systems, the PEs are generally specialized and thus devoted to different processings. Each PE is in charge of a different elaboration stage, sending the results to the next PE for further processing.

A lot of pipeline implementations have been discussed in the literature: the TAS [20], capable of

[1] Each element has two coordinates, m and n bits long; considering also an eventual carry for each operation, $n+m+2$ bits are needed.

performing three stages of binary morphological operations using an hexagonal structuring element; the Cytocomputer [31], which provides a serial pipeline of stages, applying a single 3×3 morphological or 8-bit algebraic operation on the whole image; the MITE [19], based on multiple pipelines with reconfigurable interconnections; the MVI Genesis 4000 [34], which consists of several different specialized stages (arithmetical, geometrical, and statistical), reaching a 10 Mbytes/s throughput.

In the following, the computational complexity of a morphological dilation will be derived for pipeline systems formed by a set of PEs, each one operating on a single image pixel, and with the possibility to access a restricted neighborhood. This is a natural implementation of low-level image processing tasks [25], where image data come (generally in a raster-scan format [1]) from the acquisition system, and enter the first processing stage. This solution is particularly efficient when dealing with real-time constraints, since it allows to reach a considerably high throughput.

The method used to compute a dilation is based on the following property [16],

$$A \oplus B = \bigcup_{b \in B} (A)_b \ , \tag{23}$$

which shows a different technique, based only on operations easily implementable on pipeline systems, such as unions and translations. In fact, if the image data are loaded one at a time into a device that behaves like a shift register, a translation of the original image is just a delayed output of the shift register. Moreover, some additional hardware may be needed to prevent wrap-around of border pixels, as indicated in [1]. Unfortunately, delay elements can provide only translations in the direction opposite to the scanning direction (generally in row-major order, according to the axis orientation in figure 1.a). Thus a structuring element B, containing at least one item $b = (b_0, b_1)$ such that $\Big((b_0 \leq 0) \cap (b_1 \leq 0)\Big) \cup \Big((b_0 > 0) \cap (b_1 < 0)\Big)$, as indicated in figure 1.b, when used to parametrize a dilation, implies a non-causal delay, which cannot be performed. Recalling the translation invariance property of dilation [16]

$$(A)_x \oplus B = (A \oplus B)_x \ , \tag{24}$$

if $\tilde{b}$ represents the most astringent non-causal delay, the problem can be solved performing an initial causal delay $(-\tilde{b})$ on the input image A. Thus dilation $(A)_{-\tilde{b}} \oplus B$ contains only causal delays. The final result

$$\begin{aligned} (A)_{-\tilde{b}} \oplus B &= (A \oplus B)_{-\tilde{b}} = \\ &= \Big(\bigcup_{b \in B} (A)_b \Big)_{-\tilde{b}} = \bigcup_{b \in B} (A)_{b-\tilde{b}} \end{aligned} \tag{25}$$

can now be recovered by just a translation $(\tilde{b})$ in the output buffer.

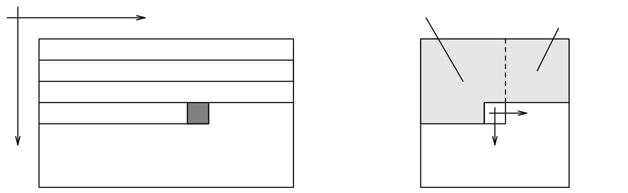
The most important constraint which limits the dimensions of practical structuring elements is the high number of delay elements required by morphological operations on large structuring elements. Thus, in this case a decomposition is required to reduce the hardware complexity of the system; as a drawback, it introduces some extra processing steps, which anyway can be assigned to different PEs.

Let's consider a pipeline in which all the elaborations are synchronized. If in a generic dilation $A \oplus B$ the structuring element B can be decomposed in $B = \Big(\Big((B_1 \oplus B_2) \oplus B_3\Big) \oplus \ldots \Big) \oplus B_n$, where each B_i has a dimension smaller than the neighborhood accessible by each PE, and if the number of PEs is larger than n, then the chain of dilations can be distributed among all the PEs. Each PE iterate its elaboration on every image pixel a_i, as soon as the data is available to the first pipeline stage. Since each dilation is performed in parallel by the chain of PEs, in stationary conditions the computational complexity is given by:

$$\mathcal{C}(A \oplus B) = \mathcal{F}_x \times \mathcal{F}_y \times \mathcal{C}(a_i \oplus B_j) \ , \tag{26}$$

where $\mathcal{F}_x$ and $\mathcal{F}_y$ represent the frame dimensions, and $\mathcal{C}(a_i \oplus B_j)$ indicates the complexity of a dilation between an image composed by a single element a_i and a generic structuring element of the set $\{B_1, B_2, ..., B_n\}$; the latter represents the complexity of the dilation associated to any single pipeline stage. Thus, for sufficiently long pipeline systems, the complexity of a generic morphological operation based on a convex structuring element is independent of the structuring element dimensions, while it depends directly on the image dimensions.

Equation (22), introduced in the case of serial architectures, indicates also the computational complexity of morphological operations based on non-convex structuring elements performed on this kind of pipeline systems. Thus, if the system is composed by multiple pipelines operating in parallel, such as the MITE [19], the total computational complexity can be reduced assigning each sub-elaboration of convex sets ($I \oplus B_i$, according to equation (22)) to a different pipeline.



a) image scanning directions *b) generic structuring element*

Figure 1: Data scanning in pipeline systems.

2.2.2 Systems with spatial parallelism

When a bit-mapped data representation is used, mathematical morphology operations involve repeated computations over large data structures, thus any form of spatial parallelism improves the overall performances. Parallel architectures with spatial parallelism are formed by a high number of PEs devoted to the simultaneous elaboration of a single image area.

A number of processor arrays with a bidimensional grid interconnection scheme and a SIMD processing paradigm have been conceived, designed and implemented, starting from the original ideas of Unger [37]. Fountain [11] has classified in chronological order the different research projects and implementations according to a three generation taxonomy. The early ILLIAC [3] and CLIP [26] machines belong to the first generation, mainly devoted to low level image processing tasks. The second generation comprises systems such as the ICL DAP [29], the Goodyear MPP [2] and the CLIP4 system [12] which have evolved to complete processing systems with dedicated operating systems and languages, and extended the application spectrum to high speed scientific computations.

The triggering factors of the third generation were the availability and widespread use of VLSI technologies of the 80's and the natural mapping of a bidimensional interconnection scheme over the planar structure of a silicon chip. This led to an explosion of different proposals and implementations [36, 27, 18, 17, 21, 14] and in some cases to the redesign of previous architectures. Most designs share original characteristics such as the bidimensional mesh interconnection scheme and bit serial computation while other ones, such as the Connection Machine [17], have increased the complexity of the interconnection network or widened the data path of the elementary PE, as in the CLIP7 [13] or MasPar [23] systems. A recent commercially available machine, the AIS [21], has a 1-D interconnection scheme emulating, via a specialized memory interface, a 2-D mesh organization.

In order to run mathematical morphology applications, in massively parallel architectures each pixel is assigned to a PE, which computes its new value, depending on the neighborhood configuration, which reflects the particular structuring element. The computational paradigm is then based on Cellular Automata [7, 9] and all the possible operations that the PEs can execute form the *instruction set*. Generally the interconnecting topology limits the dimensions of practical structuring elements, typically reducing it to 3×3 [10], as in the Cytocomputer and MPP implementations. Thus a generic structuring element needs to be decomposed into a chain of simpler and smaller ones, with the constraint of belonging to the instruction set.

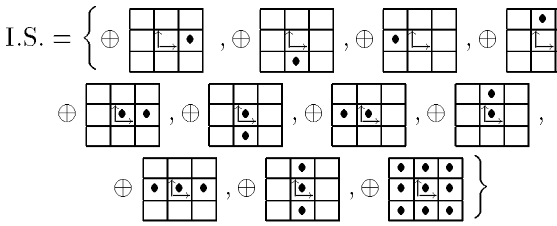
Generally in massively parallel architectures the integration of a high number of simple PEs is preferred with respect to a higher complexity of each single PE. The simple single-bit architecture of each PE allows to reach a hardware efficiency higher than the one obtained by architectures with a higher word parallelism, since the computations performed do not always involve multiple-bit operations. On the contrary, some Cellular Automata based architectures, such as the CAM-6 [35] or the CAM-8 [22], process from 4-bit data up to 16-bit data in parallel, raising the problem of grey-level structuring element decomposition [33, 44].

Thus, the efficiency of the computation depends on the instruction set, which must be accurately tuned. Next section will lead to the definition of an algorithm for the optimization of the decomposition of generic binary structuring elements on a special-purpose cellular architecture, and it will address the problem of the definition of an efficient instruction set.

3 Morphological processing on cellular architectures

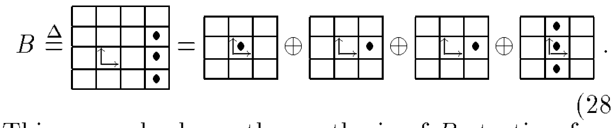
From the previous considerations, it is clear that the Array Processor is the architectural implementation that exploits in a better way the features of morphological operations.

When dealing with this kind of systems, a set is said to be *convex with respect to a specific instruction set* when it can be obtained starting from a single pixel overlapped onto the Origin, and applying only operations belonging to the instruction set of the specific system. Thus, if the structuring element is convex, then its decomposition can be easily devised by the convexity definition itself. As an example, using a cellular architecture with an instruction set composed by the following operations,



(27)

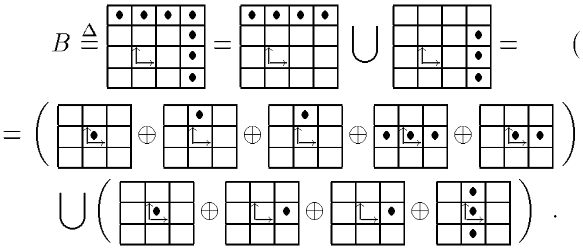
the following decomposition can be devised:



(28)

This example shows the synthesis of B starting from a single element in the Origin, and using only morphological operations belonging to the instruction set. Obviously, in order to enhance the probability that a generic structuring element is convex with respect to a specific instruction set, the instruction set itself should contain the higher number of instructions, and thus a first rough consideration is that a CISC-oriented implementation should be followed.

In the case when the generic structuring element B is non-convex, it must be subdivided according to equation (21). As already mentioned, the decomposition problem is equivalent to the synthesis of the structuring element starting from one (in the case of convex) or more than one (in the case of non-convex) elements on the Origin. An example of synthesis of non-convex set is given in the following:

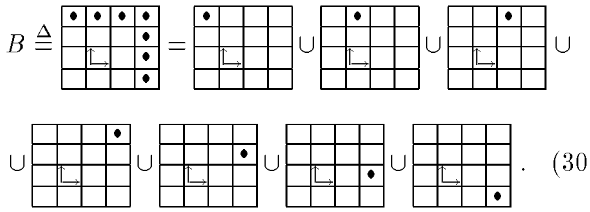


(29)

Thus, starting from a single element on the Origin, and applying only operations belonging to the instruction set, is there a rule determining the computational complexity for the synthesis of a generic $n \times m$ non-convex structuring element? In the following subsections some algorithms will be presented and their computational complexity will be computed in order to devise an efficient method for the automatic decomposition of generic structuring elements on the special-purpose cellular architecture PAPRICA.

3.1 Algorithms for set synthesis and their complexity

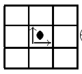
A rough indication of the computational complexity can be devised by considering a generic structuring element B as composed by the logical union of a number of convex sets equal to the number of elements in set B, each one formed by just one item b_i. For example:



(30)

In this case the synthesis of a generic single-element set, starting from the single item overlapped onto the Origin, is performed by a chain of simple translations. The complexity of this solution is then expressed by:

$$\mathcal{C}(B) = \sum_{i=1}^{\#(B)} \Big(\mathrm{d}(b_i, O)\Big) \ , \tag{31}$$

where the improper expression $\mathcal{C}(B)$, that will be used hereinafter, stands for $\mathcal{C}\Big($  $\oplus B\Big)$, and $\mathrm{d}(b_i, O)$ indicates the distance of element $b_i \in B$ from the Origin O using the metric of the specific instruction set[2].

Unfortunately this solution implies to execute the same translations more than once, decreasing the overall performances. A simple improvement to this algorithm consists in translating the initial element over the whole area defined by the structuring element, and keeping (with a union operation) only the intermediate results that correspond to any b_i, with $i = 1, 2, ..., \#(B)$. Thus, in this case:

$$\mathcal{C}(B) = m \times n \ , \tag{32}$$

where m and n are the dimensions of the minimum bounding box which contains both the structuring element and the Origin, as shown in figure 2.

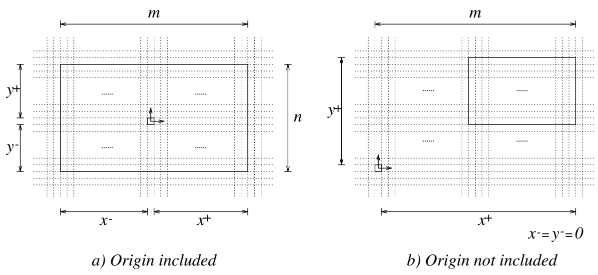


Figure 2: Decomposition parameters.

It is clear that in the previous solution some translations are useless. This consideration provides a minor improvement to the synthesis algorithm: the initial element could be translated only onto useful positions. Being $e = \#(B)$, the elements of B can be ordered in $e!$ different sequences S_j:

$$S_j = \left\{ b^{(j,1)} \to b^{(j,2)} \to ... \to b^{(j,e)} \right\} \ , \tag{33}$$

with $j = 1, 2, ..., e!$. Thus the computational complexity can be reduced to

$$\mathcal{C}(B) = \min_j \sum_{i=1}^{e} \mathrm{d}\left(b^{(j,i-1)}, b^{(j,i)} \right) \ , \tag{34}$$

where $b^{(j,0)}$ represents the Origin. Note that the determination of the minimum coincide with the solution of the traveling salesman problem. Since, if $O \notin B$, $\mathrm{d}(b^{(j,i-1)}, b^{(j,i)}) \geq 1$, a lower bound can be detected:

$$\mathcal{C}(B) \geq e = \#(B) \ , \tag{35}$$

which becomes

$$\mathcal{C}(B) \geq e - 1 = \#(B) - 1 \ , \tag{36}$$

[2]If the instruction set (as the one presented in (27)) comprehends only single-step translations in the 4 main directions, then a diagonal translation implies 2 operations, and thus a 4-neighbors metric is used. The 8-neighbors metric is used when the instruction set comprehends also diagonal translations. In the following, only the 4-neighbors metric will be used.

in the case of $O \in B$.

Being x^+, x^-, y^+, and y^- the maximum distances (considered 0 if negative) between the elements of B and the Origin as shown in figure 2, another lower bound can be found replacing them in equation (34):

$$\mathcal{C}(B) \geq x^+ + x^- + y^+ + y^- \ . \tag{37}$$

Since $x^+ + x^- = m - 1$ and $y^+ + y^- = n - 1$,

$$\mathcal{C}(B) \geq (m-1) + (n-1) \ . \tag{38}$$

Furthermore, exploiting the main feature of cellular systems, namely the capability to perform synchronously the same operation on more than one item, a better improvement can be achieved.

3.2 The RCT algorithm

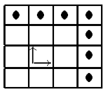
A new algorithm for parallel and cellular structuring element synthesis is presented in the following.

(a) The element overlapped onto the Origin is translated horizontally in both directions until both the ends of the bounding box of figure 2 are reached, keeping each intermediate result. This step requires a total of m translations.

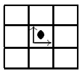
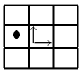
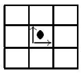
(b) Each row of the structuring element is composed with logical operations, using all the temporary sets obtained during step (a), and stored in different temporary sets (binary image planes).

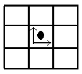
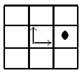
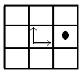
(c) Finally, the rows are translated using a mechanism that reduces the number of translations to n.

This algorithm, based on Row/Column Translations (RCT algorithm) is presented in the following example, which shows the synthesis of set B of expression (29):

$B \triangleq$  (39)

(a) Synthesis of temporary sets T_i:

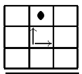
$T_1 =$  $\oplus$  ; $T_2 =$  ;

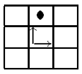
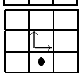
$T_3 =$  $\oplus$  ; $T_4 = T_3 \oplus$  (40)

(b) Logical composition of rows R_i:

$$\begin{array}{ll} R_1 = T_1 \cup T_2 \cup T_3 \cup T_4 \ ; & R_2 = T_4 \ ; \\ R_3 = T_4 \ ; & R_4 = T_4 \end{array} \tag{41}$$

(c) Synthesis of structuring element B:

$C_1 = R_1$; $C_2 = C_1 \oplus$  $\cup R_2$;

$C_3 = C_2 \oplus$  $\cup R_3$; $B = R_4 \oplus$  $\cup C_3$

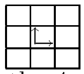
(42)

A new formulation of the computational complexity, based on the RCT algorithm, is then presented: following the previous considerations, it is

$$\mathcal{C}_{\mathrm{RCT}}(B) = (m-1) + (n-1) \ . \tag{43}$$

The previous expression shows that the performances of the algorithm, with complexity $\mathrm{O}(n+m)$, are independent of the shape of the specific structuring element: this algorithm can in fact be used also to synthesize any generic non-convex structuring element. On the other hand, the synthesis of a generic structuring element B with a serial architecture is an $\mathrm{O}(e)$ problem, where $e = \#(B)$.

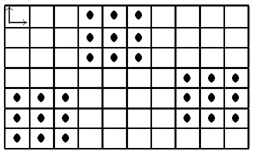
It is important to note that in the determination of the computational complexity in equation (43), only morphological operators based on a neighborhood larger than 1×1 are considered. Their execution imposes that each PE accesses the memory of other PEs, in order to fetch the processing data. On the other hand, the execution of 1×1 morphological operators, namely *logical* operations, requires only the access to the local memory of each PE, and thus they are performed more efficiently.

The RCT algorithm is based only on single-step translations in the 4 main directions, but which is the performance improvement when more general structuring elements can be used in addition to single-step translations and logical unions? The operations that a generic cellular architecture can typically implement in hardware are a subset of all the possible 3×3 structuring elements with the origin in the center (), which may contain also the translations in the 4 main directions as special cases.

The following example will show how the RCT algorithm can be improved using the specific set of 3×3 structuring elements implemented on the special-purpose massively parallel architecture, PAPRICA [5, 14, 4], but can be easily extended to any other cellular architecture holding its specific instruction set.

3.3 Set decomposition on a special-purpose architecture

The following example shows a generic non-convex set. In this case the structuring element B is composed by a union of equal non-overlapped convex sets:

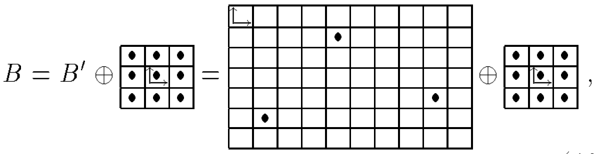
$B \triangleq$  (44)

The RCT algorithm would lead to the following computational complexity:

$$\mathcal{C}_{\mathrm{RCT}}(B) = (m-1) + (n-1) = 9 + 6 = 15 \ . \tag{45}$$

PAPRICA instruction set, already introduced in (27), is based on more general 3×3 structuring elements; this subsection will show a decomposition algorithm that exploits the special features of this special-purpose architecture, and reaches better performances with respect to the RCT algorithm. The following presentation will deliberately focus only on PAPRICA instruction set implementation, rather than the architectural details of the system.

For example, performing an initial 8-directions dilation,



(46)

the new set B' can now be decomposed following the RCT algorithm, achieving a lower computational complexity. In fact,

$$\mathcal{C}_{\mathrm{RCT}}(B') = 8 + 5 = 13 \; , \tag{47}$$

and thus, adding the complexity of the 8-directions dilation,

$$\mathcal{C}(B) = 1 + \mathcal{C}_{\mathrm{RCT}}(B') = 1 + 13 = 14 \; . \tag{48}$$

The advantage lays in the fact that with a single operation both the number of rows and columns in the resulting set have been reduced. Moreover, if the Origin was placed within the bounding box that contains the 3 elements, the complexity reduction would have been much more sensible (reaching $\mathcal{C}(B) = 1 + (7+4) = 12$).

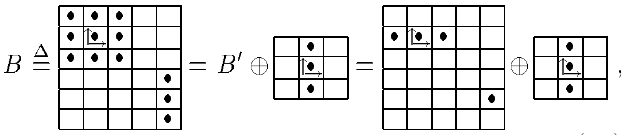
Thus, the use of more general dilations allows computational savings only when it removes more than one column and/or row external with respect to the Origin. As a consequence, the complexity based on the RCT algorithm is the lower possible unless the application of any other dilation reduces by an amount of 2 or more the dimensions of the resulting set.

In order to formalize the final formulation of the decomposition algorithm, let's distinguish between two different cases:

1. there exist at least one dilation that can be applied to the whole set, as in example (46), decreasing by an amount of 2 or more the dimensions of the resulting set;

2. there exist a dilation which can be applied only to a subset of the original structuring element, decreasing the dimensions of one of the resulting sets.

An example of the first case is given in the following:

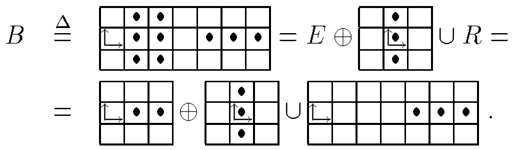


(49)

where $\mathcal{C}_{\mathrm{RCT}}(B) = 4 + 5 = 9$, while applying a vertical dilation $\mathcal{C}(B) = 1 + \mathcal{C}_{\mathrm{RCT}}(B') = 1 + (4+3) = 8$.

If a dilation reduces only by 1 the sum of the dimensions of the set, then its application does not modify the total computational complexity, since the other modifications that it could eventually produce are internal with respect to the bounding box of the resulting set, which normally fixes the complexity of the synthesis of that set, as shown in [6].

On the other hand, an example of the second case is given in the following: it shows the partial decomposition of a set B into two subsets: the eroded one (E) and the remainder one (R). In this case a logical union is required:



(50)

In this case the computational complexity, beside the dilation operation, must take into account the decomposition of both the eroded and remainder sets; thus $\mathcal{C}(B) = 1 + \mathcal{C}(E) + \mathcal{C}(R) = 1 + 2 + 6 = 9$, while $\mathcal{C}_{\mathrm{RCT}}(B) = 6 + 2 = 8$. The increment in the computational complexity is due to the fact that, on single-bit PEs, each operation (in this case a translation) is performed on a single binary set; thus, if the synthesis of the two sets E and R requires the same operations, they will be executed twice. Such a decomposition may be useful when the first part of the synthesis of E and R involve the same elaborations, that can thus be executed only once, as shown in figure 3

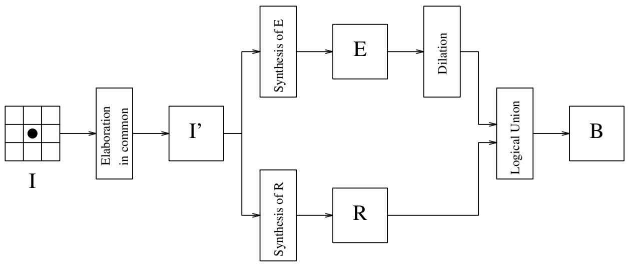
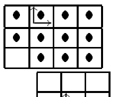
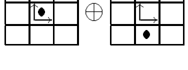
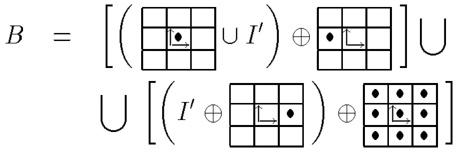


Figure 3: Processing in the case of decomposition with remainder.

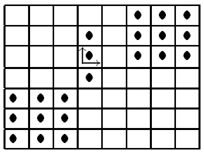
and in the following example. The synthesis of set $B \triangleq$  can be obtained using one temporary set $I' =$  :



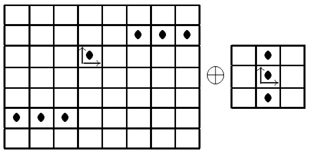
(51)

Thus, following this last formulation of the algorithm, it is $\mathcal{C}(B) = 4$, while $\mathcal{C}_{\mathrm{RCT}}(B) = 5$.

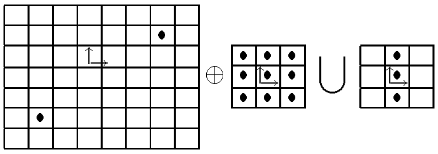
The only rule missing in the presented algorithm refers to the problem of choosing, in case of possibility, which one of the two previous cases leads to the best decomposition. The following example shows a set B that admits two different decompositions following the two previous cases (note that $\mathcal{C}_{\mathrm{RCT}}(B) = 13$):

$B \triangleq$  . (52)

Decomposition following case 1: ($\mathcal{C}(B) = 12$)

$B =$  . (53)

Decomposition following case 2: ($\mathcal{C}(B) = 11$)

$B =$  . (54)

Note that in the specific case of PAPRICA architecture, expression (53) could be further improved, thanks to the presence of a horizontal dilation.

The final formulation of the algorithm for set decomposition on architectures with the possibility to perform a fixed set of 3×3 dilations and only logical unions is thus summarized in the following.

1. If the application of a dilation that can lead to an eroded set whose dimensions sum is smaller by an amount of 2 or more with respect to the original one B does not exist in the instruction set, then proceed with the RCT algorithm;

2. otherwise, one or both of the two following cases are possible:

   (a) $B = E_1 \oplus D_1$, and/or

   (b) $B = E_2 \oplus D_2 \cup R$,

   where D_1 and D_2 represent the dilations that achieve the best dimensions reduction for each case. Note that there is no penalty in this choice: in fact, due to the chain rule property [16], if a dilation would produce better results, it could be swapped and become the first dilation to be applied.

   - If only one case is possible, then proceed with that decomposition;
   - otherwise, if $\mathcal{C}(E_1) \leq \mathcal{C}(E_2)$, then proceed with decomposition (a);
   - otherwise, decomposition (b) should be chosen only if there exist an intermediate set I' such that (following notations of figure 3)

   $$\mathcal{C}(E_1) + 1 > \mathcal{C}(I') + \mathcal{C}(I' \to E_2) + \\ + \mathcal{C}(I' \to R) + 1 \ , \quad (55)$$

   where $\mathcal{C}(A \to B)$ indicates the computational complexity of the synthesis of set B starting from set A; note that, in this specific case, $\mathcal{C}(A \to B) = \mathcal{C}(B) - \mathcal{C}(A)$. Thus, from expression (55), if

   $$\mathcal{C}(E_1) > \mathcal{C}(E_2) + \mathcal{C}(R) - \mathcal{C}(I') \ , \quad (56)$$

   then proceed following case (b).

It must be noted that in order to choose between case (a) and (b), both decompositions must be investigated; the decomposition was thus implemented through a recursive algorithm.

4 Conclusions

In this paper the implementation of mathematical morphology operations on different computer architectures has been considered. First its computational complexity has been computed for general-purpose systems, then some cellular algorithms have been proposed. Generally, a massively parallel cellular architecture has a high number of simple PEs; they implement simple operations based on 3×3 structuring elements and logical operations. One of the main design problems of such systems is the definition of the best instruction set capable of running morphological processings with as high efficiency as possible. A decomposition algorithm has been devised for special-purpose architectures implementing, beside translations, a few other morphological operations. The application of this algorithm to a number of real-world cases, shows that single-step translations are the most used operations, while more complex dilations are seldom used. This implies that the instruction set should be RISC-oriented, preferring the integration of a higher number of PEs with respect to a more complete instruction set.

These considerations have been applied to PAPRICA architecture, contributing to the evolution of the project toward PAPRICA II and PAPRICA III. Moreover some further considerations, now still under evaluation and development, on the possible use of different logical operations in addition to the logical union have partially determined the choice of PAPRICA III instruction set. A different paper will cover this problem as well as the use of larger structuring elements, trying to reach the optimal lower bound for cellular architectures, namely, following the symbols of figure 2, $\mathcal{C}(B) \geq \max(x^+, y^+, x^-, y^-)$.

References

[1] L. Abbot, R. M. Haralick, and X. Zhuang. Pipeline Architecture for Morphological Image Analysis. *Intl. Journal of Machine Vision and Applications*, (1):23–40, 1988.

[2] K. Batcher. Design of a massively parallel processor. *IEEE Transactions on Computers*, C-29:836–840, 1980.

[3] W. L. Bouknight et al. The ILLIAC IV system. *IEEE Proceedings*, 60:369–388, 1972.

[4] A. Broggi, G. Conte, F. Gregoretti, L. Reyneri, L. Rigazio, C. Sansoè, and C. Zamiri. PAPRICA. In *CAD and Architectures: Reports on Architectures and Algorithms for VLSI Design*, pages 21–141. CNR - Progetto Finalizzato MADESS, Roma, 1990.

[5] A. Broggi, V. D'Andrea, and F. Gregoretti. A low-cost parallel VLSI architecture for low-level vision. In M. Takagi, editor, *MVA'92 - IAPR Workshop on Machine Vision and Applications*, pages 11–15, Tokyo, Japan, 1992. International Association for Pattern Recognition, IAPR.

[6] A. Broggi and G. Gennari. Scomposizione di Matching Elaments per PAPRICA. Tech. Rep. CS93-1, Dip. Ingegneria dell'Informazione, Università di Parma, 1993.

[7] E. F. Codd. *Cellular Automata*. Academic Press, New York, 1968.

[8] S. J. Crabtree, L.-P. Yuan, and R. Ehrlich. A Fast and Accurate Erosion-Dilation Method Suitable for Microcomputers. *Computer Vision, Graphics, and Image Processing: Graphical Models and Image Processing*, 53(3):283–290, May 1991.

[9] D. Farmer, T. Toffoli, and S. Wolfram, editors. *Cellular Automata*, Amsterdam, March 1983. North-Holland.

[10] F.Gerritsen and P.Verbeck. Implementation of cellular logic operators using 3×3 convolution and table lookup hardware. *Computer Vision, Graphics and Image Processing*, 27, 1984.

[11] T. Fountain. *Processor Arrays: Architectures and applications*. Academic-Press, London, 1987.

[12] T. Fountain and V. Goetcherian. CLIP4 Parallel Processing System. *IEE Proceedings*, 127E:219–224, 1980.

[13] T. Fountain and K. Matthews. The CLIP 7A Image Processor. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 10(3):310–319, May 1988.

[14] F. Gregoretti, L. M. Reyneri, C. Sansoè, A. Broggi, and G. Conte. PAPRICA - A Dedicated Processor for Morphological Image Analysis. In *Proceedings 7th International Conference on Image Analysis and Processing*, Bari, Italy, September 20-22 1993.

[15] F. C. A. Groen and N. J. Foster. A fast algorithm for cellular logic operations on sequential machines. *Pattern Recognition Lett.*, 2:333–338, 1984.

[16] R. M. Haralick, S. R. Sternberg, and X. Zhuang. Image Analysis Using Mathematical Morphology. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 9(4):532–550, 1987.

[17] W. D. Hillis. *The Connection Machine*. MIT Press, Cambridge, Ma., 1985.

[18] I.N.Robinson and W.R.Moore. A parallel processor array architecture and its implementation in silicon. In *Proceedings of IEEE Custom Integrated Circuits Conference*, pages 41–45, New York, 1982. Rochester.

[19] M. Kimmel, R. Jaffe, J. Manderville, and M. Lavin. MITE: Morphic Image Transform Engine, an architecture for reconfigurable pipelines of neighborhood processes. In *IEEE Computer Society Workshop for Pattern Analysis and Image Database Management*, pages 493–500, Miami Beach, Florida, November 1985.

[20] J. Klein and J. Serra. The texture analyser. *Journal of Microscopy*, 95(2):349–356, April 1973.

[21] L.A.Schmitt and S.S.Wilson. The AIS-5000 Parallel Processor. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 10(3):320–330, May 1988.

[22] N. Margolus and T. Toffoli. Cellular Automata Machines. In G. D. Doolen et al., editors, *Lattice Gas Methods for Partial Differential Equations*, pages 219–249, Redwood City, California, 1990. Addison Wesley.

[23] MasPar Computer Corporation, Sunnyvale, California. *MP-1 Family Data-Parallel Computers*, 1990.

[24] G. Matheron. *Random Sets and Integral Geometry*. John Wiley, New York, 1975.

[25] D. McCubbery and R.Lougheed. Morfological Image Analysis Using a Raster Pipeline Processor. In *Workshop Computer Architecture for Pattern Analysis and Image Database Management*. IEEE Computer Society, November 1985.

[26] M.J.Duff, D.M.Watson, T. Fountain, and G. Shaw. A cellular logic array for image processing. *Pattern Recognition*, 15:229–247, 1973.

[27] NCR Corporation, Dayton, Ohio. *Geometric Arithmetic Parallel Processor*, 1984.

[28] I. Pitas and A. N. Venetsanopoulos. Morphological Shape Decomposition. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 14(1):38–45, January 1990.

[29] S. Reddaway. DAP - A distributed array processor. In *1st Annual Symposium on Computer Architectures*, pages 61–65, Florida, 1973.

[30] C. H. Richardson and W. Schafer. A Lower Bound for Structuring Element Decomposition. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 13(4):365–369, April 1991.

[31] R.M.Lougheed and D.L.McCubbrey. The Cytocomputer: A Practical Pipelined Image Processor. In *7th Annual Symposium on Computer Architectures*, pp. 271–277, 1980.

[32] J. Serra. *Image Analysis and Mathematical Morphology*. Academic Press, London, 1982.

[33] F. Y.-C. Shih and O. R. Mitchell. Decomposition of gray-scale morphological structuring elements. *Pattern Recognition*, 24(3):195–203, 1991.

[34] S. R. Sternberg. An overview of image algebra and related architectures. In *Integrated technology for parallel image processing*, pages 79–100. Academic Press, London, 1985.

[35] T. Toffoli and N. Margolous. *Cellular Automata Machines*. MIT Press, Cambridge, MA, U.S.A., 1987.

[36] T.Sudo, T.Nakashima, M.Aoki, and T.Kondo. An LSI adaptive array processor. In *Proc. IEEE International Solid-State Circuits Conference*, pages 122,123,307, 1982.

[37] S. Unger. A computer oriented toward spatial problems. *Proceedings IRE*, 46:1744–1750, 1958.

[38] R. van den Boomgaard. *Mathematical Morphology: Extensions Towards Computer Vision*. PhD thesis, Universiteit Van Amsterdam, 1992.

[39] R. van den Boomgaard and R. van Balen. Fast Algorithms for Morphological Image Operations using Bitmapped Binary Images. In B. Zavidovique and P.-L. Wendel, editors, *Computer Architectures for Machine Perception*, pages 453–460, Paris, December 1991.

[40] R. van den Boomgaard and R. van Balen. Methods for Fast Morphological Image Transforms Using Bitmapped Binary Images. *Computer Vision, Graphics, and Image Processing: Graphical Models and Image Processing*, 54(3):252–258, May 1992.

[41] B. J. H. Verwer and L. J. van Vliet. A contour processing method for fast binary neighborhood operations. *Pattern Recognition Lett.*, 7:27–36, 1988.

[42] S. S. Wilson. Theory of Matrix Morphology. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 14(6):636–652, June 1992.

[43] J. Xu. Decomposition of Convex Polygonal Morphological Structuring Elements into Neighborhood Subsets. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 13(2):153–162, February 1991.

[44] X. Zhuang. Decomposition of Morphological Structuring Elements. Tech. Rep., Dept. of Electrical and Computer Engineering, University of Missouri-Columbia, 1992.

[45] X. Zhuang and R. M. Haralick. Morphological structuring element decomposition. *Computer Vision, Graphics, and Image Processing*, 35:370–382, September 1986.