To be presented at ISMM2002
International Symposium for Mathematical Morphology VI
Sydney, Australia, April 3-5, 2002

Implementation Efficiency of Binary Morphology

Dan S. Bloomberg[*]
Leptonica[†]

Abstract

The efficiency of implementations of binary morphology is investigated, using both full image rasterops and word accumulation methods. All processing speeds are expressed in a way that is relatively independent of CPU speed and the sizes of both image and structuring element; namely, elementary pixel operations per CPU cycle (*EPO/cyc*). Options for handling boundary pixels are discussed. It is shown that use of successive full image rasterops is much slower than methods where the full structuring element is applied repeatedly to small parts of the image. Processing speeds of the former range from about 1 to 3 EPO/cyc, whereas the latter are typically between 4 and 7 times faster and range from 3 to 12 EPO/cyc. For small images using rasterops, vertical operations are about twice as fast as horizontal (3.2 vs 1.6 EPO/cyc); using word accumulation, vertical operations are only slightly faster than horizontal (12 vs 10 EPO/cyc). Performance on large images is reduced by a factor of between 2 and 4, due to slow reads and writes to main memory.
Keywords: binary morphology, image morphology, rasterop, destination word accumulation, structuring element, unrolled loops

1 Introduction

Binary morphology is in widespread use in a variety of applications that require identification of shapes and textures, because its nonlinear operations are useful for making fast, localized decisions. Starting in the mid-1980s, several companies built "machine vision" systems for high speed operation, typically using specialized hardware with wide word parallel operations, and requiring the use of assembly language programming for low-level functions not supplied by the manufacturer. Binary morphology was a principal component of such systems. Today, many of these tasks can be programmed in C and run at higher speeds on inexpensive GHz microprocessors. The intent of this paper is to show the performance that is available with C compilers from the current generation of 32-bit processors. Compilers are sufficiently good, and processors sufficiently complex, that there

[*]Contact: bloomberg@ieee.org
[†]Leptonica: www.leptonica.com

is little incentive for hand-tuning assembly code. Our approach is to consider algorithms with a general understanding of the computational complexity and bandwidth limitations in the processor and memory components, but not to be concerned in detail because (1) there are too many variables in existing hardware and (2) we wish to give an estimation of performance, not just algorithmic complexity, that is relatively independent of compilers and hardware, present and future.

To express results that are relatively independent of the type and speed of the CPU, as well as the sizes of both image and structuring element, we use the *elementary pixel operation per CPU cycle*, or *EPO/cyc*. This gives the number of elementary pixel operations performed in each CPU cycle. An elementary pixel operation is one where a single pixel from a *source* image is combined by boolean logic with another pixel in a *destination* image. The *rasterop* is the low-level image processing operation that performs elementary pixel operations over all pixels in a designated rectangle in the destination image.

1.1 Some previous approaches

Many available image processing programs treat binary images as a special case within a general framework, by unpacking them into 8 or 32 bit/pixel (*bpp*) images. These are inefficient in space and computation. Historically, these general image processing platforms handled relatively small square images of size 256 to 1024 pixels, but the large binary images acquired by scanning printed pages at resolutions of 300 to 600 pixels/inch (*ppi*) require packed representations and specific algorithms applicable to packed binary pixel data.

Waltz et al [2, 6] have developed pipelined algorithms that are applied to a single pixel at a time (equivalent to *unpacked* data). They use a pair of (row, column) finite state machines, tailored to each *structuring element* (SE), for dilation or erosion. Implementations on 32-bit processors require about 100 CPU cycles/pixel on 512x512 pixel images, independent of the SE size for non-sparse SEs. For example, an erosion using a non-sparse SE with 100 hits will run at about 1 *EPO/cyc*.

Several methods using packed binary pixel data have been reported. Lien[4] applies an exhaustive neighborhood lookup table, applicable for small structuring elements, to erode image components. The result is content-dependent and, except when used iteratively on boundary pixels for thinning connected components, relatively slow. Boomgaard and Balen [1] implemented a destination word accumulation method similar to that used here, except that their algorithm, like full image rasterops (see below), is not customized for specific SEs. Using internal branches and function calls in the destination word inner loop, its efficiency is similar to that of full image rasterops. More recently, York et al [7] have implemented a similar destination word accumulation method on very long instruction word processors, using 64-bit registers, developed for multimedia applications. (This is particularly fast for grayscale morphology, because the max and min operations can be applied independently and concurrently to each of the eight bytes in each register.) On 512x512 pixel binary images, with DMA to ferry data between main memory and the mediaprocessor chip, they achieve about 4 *EPO/cyc* on each 64-bit register.

1.2 Iteration and large structuring elements

We consider operations with simple SEs, typically one-dimensional, although the methods are applicable to all SEs. Also, operations with some larger SEs can be implemented efficiently by concatenating operations using small SEs. For example,

- *2-d separable*. Implement separable 2-d SEs by sequential operations with 1-d horizontal and vertical SEs.
- *Linear decomposition*. For a SE S, erode with $2S = S \oplus S$ by concatenation: $(I \ominus S) \ominus S$.
- *Logarithmic decomposition*. Dilate or erode using a sequence of SEs with a small number of hits (e.g., 2 or 4), each SE being approximately twice as big as the previous one. By avoiding overlap, the number of hits in the composite SE doubles with each additional SE.
- *Decomposable ball*. As an approximation to a ball, an operation with a nonseparable octagonal SE can be composed by a sequence of operations using alternating 4-connected and 8-connected 3x3 neighborhood SEs. For example, an octagon of diameter 9 can be decomposed as $S_9^{oct} = S_4 \oplus S_8 \oplus S_4 \oplus S_8$.

1.3 Full image rasterops

The fundamental binary morphological operations of *dilation* and *erosion* can be defined in several equivalent ways[3, 5]. We first give a standard definition, and then modify it for other implementations. Let I represent a binary image and S represent a (typically small) binary SE. Both I and S have an origin; by convention, the origin of S is called its *center*. Then the *dilation* $\oplus$ and *erosion* $\ominus$ of I by S are defined, respectively, as

$$I \oplus S \equiv \bigcup_{b \in S} I_b = \bigcup_{x \in I} S_x \tag{1}$$

$$I \ominus S \equiv \bigcap_{b \in \overline{S}} I_b = \bigcap_{x \in I} \overline{S}_x \tag{2}$$

where I_b is the *translation* of I along the pixel vector b, S_x is the *translation* of S along the pixel vector x, and the set union and intersection operations represent bitwise OR and AND, respectively. The elements x of I are the (row, column) doubly-indexed locations $x = \{i, j\}$ of foreground (ON) pixels. S can be considered a set of pixels, in a two-dimensional binary image, consisting of "hits" at locations $b = \{i, j\}$ (relative to the *center*) for which $S(i, j) = 1$. The set $\overline{S}$ is the inversion of S about its center; namely, $\overline{S} = \{-b \mid b \in S\}$. We adopt the usual convention where ON pixels are binary 1.

The implementation of binary morphology by successive full image rasterops, given by the first definitions in (1) and (2), will be discussed in Section 2. An implementation that strictly followed the second definitions in (1) and (2) would be implemented by a sequence of SE-sized rasterops, where the full SE is ORed or ANDed at each ON pixel in the image I. This is impractial, particularly for small SEs, unless the image I has little foreground.

1.4 Source word accumulation

Other partitionings are possible, and word-sized partitionings can be implemented very efficiently. Define $I^N_\beta(\alpha)$ to take N pixels starting at location α and translate them by β, where α and β are vectors on the two-dimensional image lattice. The N pixels must be in a geometrical shape that tiles the image. In the following we take N to be 32 horizontal pixels. Then dilation and erosion can be expressed, respectively, by a "source word accumulation" (*swa*) method, where each source word is used sequentially, with the SE, to write into the destination image:

$$I \oplus S = \bigcup_{w \in I} \bigcup_{b \in S} I^{32}_b(w) \tag{3}$$

$$I \ominus S = \bigcap_{w \in I} \bigcap_{b \in \overline{S}} I^{32}_b(w) \tag{4}$$

The w in the outer sum are over all indices (i, j) where $j \bmod 32 = 0$. These give the location of the left-most pixels in each 32-bit word in the image. An implementation of (4) requires both initializing the destination image to all ON and masking the (generally unaligned) 32-bit segments that are ANDed.

Equations (3) and (4), with the sum over 32-bit words w replaced by a sum over N-sized tilings, are generalizations of (1) and (2), in that for N comprising the entire image we get the first set of definitions and for $N = 1$ we get the second set. This should be clear for erosion. In making the connection for dilation for $N = 1$, note that a conventional interpretation of the second form in (1) is a union over ON pixels in I of the ON pixels in the translated SE, but this is equivalent to a union over *all* pixels in I, where *each pixel is replicated in a translated pattern given by the SE*. Replication of OFF pixels has no effect on the union. Similarly, the union in (3) is over all units w, independent of the pixel content of the unit.

1.5 Destination word accumulation

We can also express dilation and erosion by a "destination word accumulation" (*dwa*) method in the form

$$I \oplus S = \bigcup_{w \in I} \bigcup_{b \in S} I^{32}_b(w - b) \tag{5}$$

$$I \ominus S = \bigcup_{w \in I} \bigcap_{b \in \overline{S}} I^{32}_b(w - b) \tag{6}$$

which successively computes 32-bit words in the destination. Note that $I_b^{32}(w-b)$ takes 32 bits starting at $w-b$ and shifts them by b, thus writing them into the 32-bit aligned word $I(w)$. Because words are computed independently, a union(intersection) over words can be used formally in (6) if the destination is initialized to OFF(ON) pixels. But in practice, the distinction is unimportant and initialization is unnecessary because each destination word is written once. Both *swa* and *dwa* methods are amenable to efficient implementation, but *dwa* is preferred. We discuss their use in Section 3, with emphasis on *dwa* given in (5) and (6).

In general, boundary pixels require special techniques, and options for handling boundary conditions are discussed for both full image rasterops and word accumulation. Finally, the performance of the methods is compared in Section 4. All source code is available at *http://www.leptonica.com*.

2 Implemenation by successive rasterops

According to (1), dilation is implemented by a sequence of full image rasterops, using the OR operation, on a destination image initialized to all 0s. The translations associated with each rasterop are specified by the SE. An efficient implementation has the following properties:

1. **Packed data**. The binary data is packed with 32 pixels/word. Each image raster line begins on a word boundary and 32-bit operations are used throughout. In order for 32-bit shift operations to move naturally across 32-bit boundaries, the data in each 4 byte word must have the most significant byte at the left. On little-endian machines, the byte order from left to right in a word is then 3-2-1-0. The same operations can of course be used for rasterops with bpp > 1 by scaling the width by bpp.

2. **High-level clipping**. Clipping is required to prevent data access to or from regions outside array boundaries. This must be done by adjusting the boundaries of the rasterop before any data is moved; otherwise, it is necessary to use special cases in the low-level code.

3. **Special case for aligned rasterops**. When the left sides of source and destination rectangles are 32-bit aligned (i.e., $x_{0s} - x_{0d} = 0 \bmod 32$) the low-level implementation is simpler. A test should be made before dispatch to the low-level code.

4. **Use a sequence of 1-d operations for separable 2-d SEs**. The time for operations with separable SEs grows linearly with the dimension of the SE, rather than as the square. This is particularly important when the SE is larger than 3x3.

2.1 Destination initialization

Dilation by full image rasterops is straightforward: initialize to OFF pixels and apply the OR operation for each rasterop, as described by (1). Initialization of a segment of memory using the library function `memset` is fast.

Erosion is more tricky. The destination can be initialized to OFF pixels, as with dilation, and the first full image rasterop is a COPY, whereas all succeeding full image rasterops are ANDed. Alternatively, the destination can be initialized to ON pixels, and all full image rasterops are ANDed, as described by (2). The results from either of these operations are identical, but we are not finished, because for each full image rasterop in the erosion, we should bring in OFF pixels in the shifted image from beyond the source image boundary. However, the rasterop implementation clips to the source image boundary (as well as the dest image boundary), so these OFF pixels are not used, and the corresponding pixels in the dest image, rather than being set OFF, are unaffected. It is thus required to clear these pixels *after* the full image rasterops are completed. Consequently, if the first full image rasterop in the erosion is a COPY, it is not necessary to perform any initialization of the dest.

2.2 Boundary conditions

We require that dilation and erosion be implemented as if the source image were surrounded by a sufficient number of OFF pixels so that with all shifts, these outer source pixels cover the entire dest image. For full image rasterops we never see such pixels, because the rasterop clips to the actual source image. Nevertheless, the operation must be consistent with a rasterop over the entire dest image using this extended source.

For dilation, whether or not we OR a set of OFF pixels makes no difference, but for erosion, the AND of these OFF pixels would clear the pixels near the boundary in the dest. To get the correct result using rasterop, it is necessary to clear these pixels after the set of full image rasterops. An example will make this clear. An erosion with a SE consisting of a hit in the center and another 2 pixels to the left is implemented by an unshifted copy of the source followed by an AND of the source shifted 2 pixels to the right. It is thus necessary to clear the leftmost 2 pixel columns in the dest.

There are further complications with opening and closing, which are anti-extensive and extensive, respectively. We assume that the correct operation occurs if the dest image is sufficiently extended with OFF pixels such that after the operation all ON pixels are contained within the extended dest. In some situations it is not necessary to embed the dest in a larger image with OFF exterior (border) pixels to get correct results. Define a SE as being *contained* if the center lies on a hit, and *uncontained* otherwise. Then a binary erosion with a contained SE is anti-extensive, so no border extension is required, and likewise for an opening with a contained SE. But if the SE is uncontained, the erosion is not anti-extensive, and a border extension is necessary in general for a correct opening.

It can be seen that closing in general requires a border extension. A dilation, whether or not the SE is contained, will in general result in ON pixels within the border extension. If those external ON pixels are ignored, the closing will not in general be extensive. To see this, suppose the source image pixels are all ON. A dilation extends the ON pixels into the external border, and a subsequent erosion removes them, leaving the dest image with all ON pixels. But if the external ON pixels are ignored in the erosion, some pixels in the dest near the boundary but inside the dest proper will be removed, and the operation will not be extensive.

To make the operations correct and avoid edge artifacts, a border of OFF pixels should be added to the the dest image by embedding the dest in a sufficiently large image, and the border should be removed afterwards. For efficienct implanting and extraction, the left and right borders should be an integral number of words (i.e., a multiple of 32 pixels). However, for many applications, loss of a few ON pixels near the image boundary during closing and opening (the latter with uncontained SEs) is not of concern.

3 Implemenation by destination word accumulation

In the word accumulation methods, the input image is divided into words. In *swa*, the effect of each source word on multiple destination words is found, whereas in *dwa*, the effect of various source words on each destination word is accumulated and saved. We choose the latter because it requires no masking operations and incurs less latency from writes from the cache to main memory. It can also more easily be adapted by image subdivision to parallel operations with shared memory multiprocessors. As with full image rasterops, an efficient implementation requires packed data with byte order accommodating 32-bit shift operations. The most useful SEs are linear horizontal and vertical, and these are easily implemented. Additional properties of word accumulation morphology are:

1. **Unroll loops**. Unlike full image rasterops, which can use any SE, word accumulation is most easily and efficiently implemented for specific SEs. For each SE, the iteration over hits is hard-coded into the inner loop. No branch tests are required in the computation of each destination word.

2. **No destination initialization required.** All destination words are computed and written once.

3. **No masking required.** Unlike rasterops, which require masking in the composition of words from of partial words, no masking is necessary for *dwa*. Contributions from various words to the destination word require only a dereference, a shift, and a logical (OR or AND) operation with the destination. See Section 3.2 for a caution with erosion.

4. **Avoid special cases near boundary**. There are several options, but it is necessary not to read or write outside the arrays, and preferable not to treat pixels near the boundary as special cases.

3.1 Boundary conditions

Reads and writes outside the image arrays must be avoided, but, unlike rasterops where the rectangular region of operation is arbitrary, the word accumulation methods cannot be clipped to arbitrary boundaries from outside the low-level code. It is also difficult to write special case code for pixels near the boundary. This leaves two obvious choices: (a) carry out the morphological operation on an interior subset of source image pixels and accept boundary artifacts with pixels not fully processed; or (b) add a border of OFF pixels outside the source image that is a multiple of 32 to each side, and which is sufficiently large to prevent reads from outside the (extended) array.

Because we work in 32-bit word chunks, operating on a subset of source pixels results in boundary artifacts of at least 32 pixels on the left and right sides. To get correct results for *dwa*, add a border of at least 32 pixels to the source image. Morphological operations can read from source border words but need only write to words in the dest that are within the boundaries of the original source image. It is convenient to add a border to the dest image as well, because it may be a source image for further morphological operations. The border must typically be removed afterwards.

3.2 Low-level implementation

To indicate the simplicity of the code required for *dwa* morphology, we give the C code for computing a destination word in dilation and erosion with a horizontal SE of size 3 with centered origin. `*sptr` and `*dptr` are 32-bit source and destination words, respectively.

Dilation:

```
        *dptr = (*sptr >> 1) | (*(sptr - 1) << 31) |
                *sptr |
                (*sptr << 1) | (*(sptr + 1) >> 31);
```

Erosion:

```
        *dptr = ((*sptr >> 1) | (*(sptr - 1) << 31)) &
                *sptr &
                ((*sptr << 1) | (*(sptr + 1) >> 31));
```

Dilation is simpler and slightly more efficient, because erosion requires the intersection of fully populated 32-bit entities that are composed of shifted source words and have been aligned with the destination word.

As a slightly more complicated example that shows both row and column addressing, here is the inner loop for an erosion using an upward-slope diagonal SE of size 5 with centered origin. The variables `wpls and wpls2` are the words/line and words/(2 lines) in the source.

```
*dptr = ((*(sptr - wpls2) << 2) | (*(sptr - wpls2 + 1) >> 30))&
        ((*(sptr - wpls) << 1) | (*(sptr - wpls + 1) >> 31)) &
        *sptr &
        ((*(sptr + wpls) >> 1) | (*(sptr + wpls - 1) << 31)) &
        ((*(sptr + wpls2) >> 2) | (*(sptr + wpls2 - 1) << 30));
```

Each line gives the contribution from one of the five hits in the SE. Because the structure has simple regularities, it is straightforward to generate such low-level code automatically for any SE. This would allow one to build correct operations with minimum effort that achieve maximum computational efficiency.

4 Comparison of full image rasterops with dwa

4.1 General considerations

High performance commercial 32-bit microprocessors such as the Intel P3 have multiple levels of caching, both on and off chip. A 256 KB off-chip cache can hold two binary images of size 10^6 pixels. Thus it is expected that images much smaller than 1 Mbit will have all read accesses through the cache, whereas images much larger than 1 Mbit will be impacted by cache misses causing reads from main memory. To determine the size-dependence, we have measured performance on images of size 0.25 Mbit, 1 Mbit, and 8 Mbit. It is generally a good idea to minimize writes to main memory.

Measurments are taken with an 866 MHz P3 processor with 256 KB of off-chip cache, running on Linux kernel 2.2.18. The implementation language is C; the compiler is GNU C, version 2.95.2, and level 1 of optimization (-O) was used. We assume that all images reside in main memory (256 MB), and that no disk acess is necessary. All measurements are normalized to CPU speed (*EPO/cyc*), but overall performance may not scale with faster CPUs because latency in memory access is not tied to CPU speed.

The full image rasterops implementation can potentially require many writes to memory. For example, if the SE has N elements, a dilation will require N rasterop writes to each destination word. For the *dwa* implementation, the working set of active memory traverses the image only once. Loops are unrolled, and the compiler keeps most intermediate results in registers, writing each destination word out to main memory only once. The gain over full image rasterops is primarily due to (1) avoiding masking operations in inner loops, (2) using unrolled inner loops to avoid test and branch, (3) using a small number of variables, allowing intermediate data to remain in registers, and (4) avoiding inefficient reads and writes to main memory.

4.2 Performance

We tested all operations on three binary images, of size 0.25 Mbit, 1.0 Mbit and 8.0 Mbit. It appears that performance on the small 0.25 Mbit image is CPU limited, whereas the speed

	Linear structuring element	
	horizontal	vertical
0.25 Mbit	1.6	3.2
1.0 Mbit	1.0	1.5
8.0 Mbit	0.7	0.9

Table 1: *Full image rasterops performance in EPO/cyc for implementing binary morphology using horizontal and vertical linear SEs, for three image sizes.*

	Linear structuring element					
	3x1	5x1	7x1	9x1	1x3	1x9
0.25 Mbit dilation	9.6	9.7	10.8	11.2	12.0	11.8
1.0 Mbit dilation	4.0	5.5	6.6	7.3	5.2	8.0
8.0 Mbit dilation	2.9	4.0	5.1	6.0	3.2	6.6
0.25 Mbit erosion	7.6	6.2	6.9	7.5	12.2	11.8
1.0 Mbit erosion	3.3	4.4	4.8	5.9	5.2	8.4
8.0 Mbit erosion	2.7	3.5	4.5	5.3	3.2	6.6

Table 2: *Destination word accumulation performance in EPO/cyc for implementing binary dilation and erosion with horizontal (columns 1-4) and vertical (columns 5-6) linear SEs, for three image sizes.*

on the 8.0 Mbit image is bandwidth limited, due to reads and writes to main memory. The variance in measurements is typically about 2 percent, except for *dwa* morphology on the intermediate 1.0 Mbit image, which has a large variance of about 10 percent.

Table 1 gives the image size dependence of full image rasterops performance for morphological operations using horizontal and vertical SEs. The performance of full image rasterops has, as expected, little dependence on the specific morphological operation. Performance on the small image closely follows the algorithmic complexity. The vertical operations are about twice as fast as the horizontal, and all operations are from 2 to 3 times faster than that those on the large image.

Table 2 gives the image size dependence of *dwa* performance for dilation and erosion using different linear SEs. Consider first the small image, which has an *EPO/cyc* between 2 and 4 times faster than the large image, and where the speed reflects algorithmic complexity. The performance is nearly independent of SE size, and erosion is 30 to 40 percent slower than dilation for horizontal SEs. Dilation for horizontal SEs is almost as fast as for vertical SEs; each destination word is computed and saved in about 3 CPU cycles. The large image performance improves with larger SEs, is comparable for dilation and erosion, and is comparable for horizontal and vertical SEs of the same size.

Comparing Table 2 with Table 1, performance of *dwa* is seen to be between 4 and 7 times faster than an efficient implementation of full image rasterops. This efficiency, combined with simplicity of implementation, makes *dwa* the method of choice for binary morphology with linear, separable or decomposable SEs, or in applications where only a small number of SEs need to be hand-coded. Availability of full image rasterops morphology is useful to check the correctness of any *dwa* implementation.

References

[1] R. van den Boomgaard and R. van Balen, "Methods for Fast Morphological Image Transforms using Bitmapped Binary Images," *Graphical Models and Image Processing*, **54**(3), pp. 252–258, 1992.

[2] R. Hack, F. M. Waltz and B. G. Batchelor, "Software implementation of the SKIPSM paradigm under PIP," *SPIE Conf. Machine Vision Applications, Architectures and Systems Integration VI*, **3205**, pp. 153–162, Pittsburgh, PA, Oct. 1997.

[3] R. M. Haralick, S. R. Sternberg and X. Zhuang, "Image Algebra Using Mathematical Morphology," *IEEE Trans. Pattern Recognit. Mach. Intelligence*, **PAMI-9**, pp. 532–550, July 1987.

[4] B. K. Lien, "Efficient implementation of binary morphological image processing," *Optical Engineering*, **33**(11), pp. 3733–3738, November 1994.

[5] J. Serra, *Image Analysis and Mathematical Morphology*, Acad. Press, 1982.

[6] F. M. Waltz and J. W. V. Miller, "Execution speed comparisons for binary morphology," *SPIE Conf. Machine Vision Systems for Inspection and Metrology VIII*, **3836**, pp. 2–9, Boston, MA, Sept. 1999.

[7] G. York, R. Managuli and Y. Kim, "Fast Binary and Gray-scale Mathematical Morphology on VLIW Mediaprocessors," *SPIE Conf. Real-time imaging IV*, **3645**, pp. 45–55, San Jose, CA, Jan. 1999.