Implementation Efficiency of Binary Morphology

Dan Bloomberg
Leptonica
ISMM April 2002

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.

What???

Binary Morphology? Are you kidding? Everyone knows how to do that, n'est-ce pas?

Well, yes, it's pretty straightforward. But there are some little twists along the road that you might find interesting.

Such as?

The boundary conditions are a bit tricky to get exactly right. The order in which the computations are done, and in which the results written to the destination, matter significantly. And the size of the image affects the overall speed as well.

What speeds are attainable?

As you just read in the abstract, with small images, where most of the image data is in the cache, and using the destination word accumulation method, up to 12 destination pixels can be determined for each "care" in the structuring element *in each CPU clock cycle*! For a run-of-the-mill 1 GHz Pentium III, that's about 12 billion pixel operations per second. Less than 1 millisecond for an 8 Mpixel scanned text image.

How do I find out about all this?

The details are in the paper, and all the software is available at **leptonica.com**.

A bit of history

Haven't people achieved these speeds before, and how hard is it to do the implementation?

One thing at a time. Let's start with previous work. How about this: there have been several papers over the past 10 years that work on *one pixel at a time*, with a pipelined finite state machine for each different SE, and that take about 100 cycles/pixel, independent of the size of the SE. But this is very slow (about 1 *EPO/cyc* for a SE with 100 hits) and exceedingly complicated to program.

Scratch that. What else?

For very small SEs you can use an exhaustive neighborhood table lookup, on each foreground pixel. This might be useful for special operations on sparse images, such as thinning, but it is also very slow.

What about more general methods?

Well, you can use full image rasterops, where you bitblt the full image for each "care" in the SE. Similarly, Boomgaard and Balen implemented a general DWA (destination word accumulation) method that was not customized to specific SEs.

history, cont.

That is the standard method. What is the drawback?

These run at about the same rate, between 1 and 3 *EPO/cyc*, because they share the computational overhead that goes with the generality. You see, each word that is written to the destination must be computed by a general routine that figures out how to compose it from the source image using the shift given by each "care" in the SE. Full image rasterops do all words for each "care"; DWA does all "care"s for each word. The switch is in the order of "summing": full image rasterops has the SE "care"s in the outer loop; DWA puts the iteration over destination words in the outer loop.

Can you speed it up with special hardware?

Yes, to some extent. Recently York et al have used DWA morphology on a multimedia processor with 64-bit registers, using DMA to get data between main memory and the processor. They do a little better than on a standard general CPU, but not much. The general CPU performance is such a fast-moving target that it's hard to compete using special hardware and systems.

Making DWA fast

So how do you improve on this?

We get to the implementation part of your question. In essence, you write special DWA routines for dilation and erosion for each SE. These routines have, as a fully unrolled inner loop, exactly what you must do to compose a full destination word from various source words.

Can you give me an example?

Sure. Here's the inner loop for erosion by a symmetrical horizontal SE of width 3:

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

The source and dest pointers are initialized to the corresponding words in source and dest images. General purpose computers do these operations very efficiently. Including pointer setup, this takes about 12 machine cycles for each word, because it runs at 7.6 *EPO/cyc*, or 7.6/32 elementary *word* operations per cycle, which we divide into 3 elementary operations for this SE to get (3)(32)/7.6 cycles/word.

Making DWA easy

That's very simple. But where's the code for pixels near the boundary?

We *could* write special code for boundary words. But there are many different cases, depending on the location of the boundary word and the size and shape of the SE. The code would bloat out by at least a factor of 10, and things would get ugly. If you want to do it, be my guest!

But if you have no special boundary code, what do you do, skip the boundary pixels altogether?

No. We embed the source image in a larger image with a border of 32 OFF pixels on all sides. We compute destination words only for the interior, corresponding to the original image. If we will be using the dest later as a source image for another morphological operation, we give it a 32 pixel border as well. At the end of all morphological operations, we remove any remaining borders.

That's easy, but what is the computational overhead?

Very small. To put a border on the source, you allocate 1 MB for a full page at 300 pixels/inch, and do a fast memcpy() because with a 32 pixel border, the word boundaries are aligned for the copy.

Comparing full image rasterop and DWA implementations

How does the specialized DWA method compare with using full image rasterops?

The best way I can show this is by using the *EPO/cyc* criterion. Have a look at two tables. The times depend strongly on the size of the image. For the small 512x512 pixel image, we can assume that there are few cache misses. For the larger images, the smaller rates are due to slow fetches from main memory.

The first table gives the *EPO/cyc* for full image rasterops. The speeds are in the range of 1 - 3 *EPO/cyc*, and are independent of the size of the SE and whether the operation is dilation or erosion, but the vertical operation is a bit faster than the horizontal one.

	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

comparing, cont.

By contrast, the second table gives the *EPO/cyc* for our DWA implementation.

	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

Here the speeds are in the range of 3 - 12 *EPO/cyc*. They differ a bit for erosion and dilation for horizontal SEs, but not for vertical SEs. Also for horizontal SEs, use of longer SEs is more efficient, particularly for larger images where memory fetches can be amortized over more elementary operations. The DWA shows the same deterioration in performance with image size as full image rasterops.

Typically, DWA is about 4 times faster than full image rasterops. And I should add that all possible efficiencies have been used in the rasterop implementation; it cannot be significantly improved. We use packed data, high-level clipping, and special case the code when source and dest are 32-bit aligned. But you just can't beat the unrolled loops in DWA, plus the fact that rasterop requires extra masking to compose words from partial words.

Making DWA correct

OK. Suppose you write DWA code for a morphological operation using some SE. How do you know that you haven't made a mistake?

A fair question. Remember, we also have a general implementation of binary morphology with full image rasterops. We just do the operation on some large image with each method, XOR the results, and check that all pixels are OFF.

And are the results exactly the same?

Yes, if you handle the boundary conditions in the same way. For the rasterop implementation, we don't need to embed the source in a larger image because we clip each rasterop to image boundaries. But getting the rasterop *erosion* correct is a little tricky. Remember, with DWA we're pulling OFF pixels in from the border. To do the equivalent with rasterop, you need to zero (set to OFF) those same pixels. The number on each edge depends on the maximum distance of a "care" from the SE center on the corresponding side of the SE.

Corresponding side??

Suppose the SE has 2 hits, one on the center and one 2 pixels to the left. The erosion does a full image rasterop with a shift of 2 pixels to the right. This leaves 2 pixels on the left edge of the dest that must be zeroed.

Going for the holy grail: fast and easy

But for larger SEs, isn't it a lot of work to write that inner code, and do it correctly.

Yes, we can check that the operation is correct, but who wants to write erosion code such as this, for a simple diagonal line of length 5? (wpls2 = 2 * wpls)

```
*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));
```

And things get rapidly worse for larger elements, such as non-separable balls.

What we really want is the "holy grail" of having a general method (so that it is easy to use, like a rasterop implementation) that is also very fast (like a SE-specialized DWA implementation).

Making DWA really easy

Can this be done?

Yes, you can write a program that writes the DWA code. This was once done in special languages like Lisp, and more recently in interpreted text-handling languages like Perl. But it's easy to do in C as well.

What is the input to this code-writing routine?

We have a function that takes an array of SEs, which are in memory, and generates two C files: a high level, simple interface that you call in your program to do a specific morphological operation, and a low level interface that dispatches to the low level implementation for the requested SE.

How do you avoid namespace conflicts if you link many pairs of such files?

The generator function takes an integer as a parameter, that is both used in naming the generated C files and is embedded in the name of each generated function. To invoke a morphological operation, you need to know both that integer, which is in the name of the high level function, and the name of the SE.

Using DWA

Can you give me a practical example of how to use DWA?

Here are the guts of a little C program that computes the erosion with the above diagonal SE, both with full image rasterops and DWA, to verify that the results are the same.

```
...
int        same;   /* result: 1 if same; 0 if different */
int        index;  /* ignored */
PIX       *pixs;   /* source image in memory */
PIX       *pixt1, *pixt2, *pixt3, *pixt4, *pixt5; /* temp images */
SEL       *sel;    /* one SE */
SELA      *sela;   /* array of SEs */

      /* a function that returns an array of SEs */
    sela = selaAddBasic(NULL);

     /* get the desired SE from its name and the array */
    selaFindSelByName(sela, "sel_5dp", &index, &sel);

     /* erode with full image rasterops */
    pixt1 = pixErode(NULL, pixs, sel);

     /* add border of OFF pixels; erode with SWA; remove border */
    pixt2 = pixAddBorder(pixs, 32, 0);
    pixt3 = pixFMorphopGen_1(NULL, pixt2, EROSION, "sel_5dp");
    pixt4 = pixRemoveBorder(pixt3, 32);

     /* compare the results */
    pixt5 = pixXor(NULL, pixt1, pixt4);
    pixZero(pixt5, &same);
...
```

Odds and ends

What about the hit-miss operation in DWA?

That is not specifically implemented in the auto-generated DWA. For now, you need to compose the fg and bg DWA erosions separately and AND them together. However, the SE can have both "hits" and "misses", so it could be added within the existing framework. If you need this and want help implementing it, ask me: **bloomberg@ieee.org**.

I've been asking you a lot of questions. Is there anything you've left out?

There are some boundary condition issues we didn't cover, that you should be aware of, such as how the closing can lose pixels near the boundary. There are also formal definitions for DWA, as well as for the similar *source word accumulation* (SWA) method that is inferior to DWA, in case you want to see them. These are all discussed in the paper.

Oh, and if you have a Unix system, download the software at **leptonica.com** and try it out!

Bibliography

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.

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.

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.

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

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

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.

bib, cont.

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.

Glossary

EPO/cyc: elementary pixel operations per cpu cycle (Note: in the paper this is written "EPO/Hz", which is technically incorrect)

SE: structuring element

"care": a hit or a miss; anything but a "don't-care" (!)

rasterop: also called *bitblt*, it performs a logical operation on a rectangle of a "dest" image, using the pixel values in that rectangle and optionally those of a similar sized rectangle in a "source" image. The operation between the two images can be any combination of AND, OR, XOR and NOT.

DWA: destination word accumulation

SWA: source word accumulation