Proceedings of the 2007 IEEE Symposium on Computational
Intelligence in Image and Signal Processing (CIISP 2007)

OPPOSITION-BASED WINDOW MEMOIZATION FOR MORPHOLOGICAL ALGORITHMS

Farzad Khalvati [1], *Hamid R. Tizhoosh* [2], *Mark D. Aagaard* [3]

[1,3] Department of Electrical and Computer Engineering
[2]Department of System Design Engineering
University of Waterloo, 200 University Ave. West, Waterloo, ON, N2L 3G1, Canada

ABSTRACT

In this paper we combine window memoization, a performance optimization technique for image processing, with opposition-based learning, a new learning scheme where the opposite of data under study is also considered in solving a problem. Window memoization combines memoization techniques from software and hardware with the repetitive nature of image data to reduce the number of calculations required for an image processing algorithm. We applied window memoization and opposition-based learning to a morphological edge detector and found that a large portion of the calculations performed on pixels neighborhoods can be skipped and instead, previously calculated results can be reused. The typical speedup for window memoization was 1.42. Combining window memoization with opposition-based learning yielded a typical increase of 5% in speedups.

I. INTRODUCTION

Many digital image processing algorithms are used in real time applications. The data intensive nature of image processing, combined with the performance requirements of real time applications, makes it both crucial and challenging to optimize the performance of image processing algorithms. In this paper, we present *opposition-based window memoization (OB-WM)*, a performance optimization technique for image processing algorithms.

Opposition-based window memoization combines two concepts: window memoization (WM) and opposition-based learning (OBL). The main idea behind window memoization is to decrease the number of unnecessary calculations by reusing the previously calculated results. Window memoization uses a *reuse table* to store previously computed windows and the corresponding results. Subsequent windows are compared against the reuse table; if a matching window is found, the previously computed result is reused and the actual computation is skipped. Opposition-based learning, introduced by Tizhoosh [1], is a new learning scheme where the *opposite* of data under study is also considered in solving a problem. The opposite of any quantity is defined with respect to the problem at hand. For example, in a set-theoretical context for image data, opposite of gray-levels can be defined as logical complement of the gray-levels.

Michie [2] introduced the general technique of memoization in 1968. In hardware, memoization techniques have been proposed for microprocessors where the results of previously executed instructions, functions, or blocks are reused [3], [4], [5], [6], [7]. However, none of these techniques have been implemented in a real design yet and thus, the reported speedups (i.e. 1.15 as a typical speedup) are based on theoretical results. Handling the dependencies between instructions in a program and maintaining the coherency of the memoization table would require large content-addressable memory arrays with four or more write ports. The complexity of these memory arrays and the difficulty of pipelining them overshadow the theoretical gains in performance [8].

In embedded software, research in memoization has led to typical speedups ranging from 1.2 to 1.4 [9],[10]. However, the proposed techniques require detailed profiling information about runtime behaviour and transform the program to a new code that benefits from the value locality of data.

We chose morphological gradient as our case study since gray-scale morphological algorithms are vastly used in many applications. Optimizing the morphological algorithms is an ongoing research and many techniques have been proposed in this regard. For instance, for morphological algorithms that use flat (single-value non-zero) structuring elements (SEs), efficient algorithms have been presented in [11]. Moreover, researchers have been working on decomposing arbitrary non-flat (non-uniform) SEs to 3×3 bases to improve the performance [12], [13]. Regarding that several techniques have been proposed to decompose SEs to 3×3 size, our goal is to improve the performance of the morphological algorithms that use 3×3 non-flat SEs.

The 3×3 non-flat SEs contain up to 9 operands (one for each pixel used in the calculations). This large number of operands decreases the probability of having repeating windows, which leads to low reuse rates and makes it difficult to look up a result in the reuse table quickly.

After studying the characteristics of typical images and the behavior of morphological algorithms, we developed a lookup technique that uses *multi-thresholding* to increase the reuse rate with an insignificant loss of accuracy in the resulting image. The lack of data dependencies between morphological gradient operations guarantees that the reuse table remains coherent, which allowed us to optimize the reuse table. The reuse table is a form of a hash table, and so requires the computation of a hash key to look up a result. We decreased the time required to look up a result by updating the hash key incrementally as the convolution mask moves across the image, rather than computing an entirely new key for each window. We also bene□ted from the concept of opposite window to improve the performance further. We de□ne the opposite of each window based on the SE used. By having a window and its response to the mask, the mask response for the opposite window will be known without a need for performing the mask operations. This leads to further decrease in the number of calculations required for the algorithm and hence, higher speedup is gained. We evaluated the performance for 52 natural images of 512×512 pixels. In almost all cases, we achieved signi□cant speedups while the error in the results was so small that it could not be detected by the human eye.

The outline for the rest of the paper is as follows: In section II, we propose a detailedness algorithm. Section III presents an overview of morphological gradient. In section IV, we present window memoization. In section V, we discuss how we extend window memoization to exploit opposition-based learning scheme. Finally, we present the conclusion and future work in section VI.

II. IMAGE *"DETAILEDNESS"*

To analyze the speedups achieved for different images, we classify images based on their complexity. The classi□cation will give us an estimation of performance gain for an image, before applying the performance improvement technique on the image. In addition, it may help us customize the performance improvement technique for individual classes of images. We present an algorithm that calculates how complicated (or detailed) an image is. For a given image, the algorithm generates $n \times n$ seed pixels spread across the image, which are in equal distances (horizontally and vertically) far from each other. Four derivatives are calculated on 3×3 neighborhoods around each seed pixel and the percentage of the instances that the maximum value of the four derivatives is larger than a threshold, ϵ, is calculated. The □nal result of the algorithm for an image is a number between 0 (the least detailed) and 100 (the most detailed) indicating the level of complexity of the image (η). We have experimentally determined n and ϵ to be 102 and 15, respectively. Table I shows the detailedness algorithm.

Figure 1 shows the results generated by applying the algorithm to extreme cases. As it is seen from the □gure, the algorithm generated $\eta = 1\%$ and $\eta = 90\%$ as the indicator of image detailedness for the very simple and complicated images, respectively. We veri□ed that the detailedness algorithm produces intuitively reliable results by applying the algorithm on different natural images and demonstrating the results to 5 observers. The proposed detailedness algorithm has been designed for natural images. It is possible to achieve counter-intuitive detailedness rates by applying the algorithm to some synthetic images with speci□c patterns aimed to defeat the algorithm. An example would be an image that contains alternating black and white lines, which would have a detailedness of roughly 100%, but could be viewed as having very little complexity or detail.

Table I. Algorithm for calculation of detailedness

```
1. input an image 'I'
2. initialize counter k
3. generate $n \times n$ seed pixels, which are $\sigma$ pixels
(horizontally and vertically) far from each other
4. for each seed pixel at (i,j) calculate:
      $\Delta 1 = abs(I(i-1,j) - I(i+1,j))$
      $\Delta 2 = abs(I(i,j-1) - I(i,j+1))$
      $\Delta 3 = abs(I(i-1,j-1) - I(i+1,j+1))$
      $\Delta 4 = abs(I(i-1,j+1) - I(i+1,j-1))$
5. $\Delta max = max(\Delta 1, \Delta 2, \Delta 3, \Delta 4)$
6. if $\Delta max > \epsilon$, then increment k
7. calculate detailedness $\eta$: $\eta = \frac{k}{n^2} \times 100\%$
```

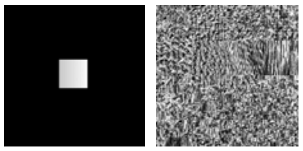

Fig. 1. Detailedness algorithm results for extreme cases: a. A simple image: $\eta = 1\%$ b. A complicated image: $\eta = 90\%$

Using the detailedness algorithm, we are able to classify the images based on the level of variation of the gray levels. We will use this classi□cation in analyzing our proposed performance improvement techniques in upcoming sections.

As input images for our simulations, we have randomly chosen 52 different images of 512×512 pixels and run the detailedness algorithm. The results for detailedness are between 1.38% and 77.65%. Figure 2 shows the distribution of the images over detailedness.

Table II. Detailedness classes of low, medium and high for 52 images

Class	range of η	Average η	images
Low	$\eta < 17\%$	9%	11
Medium	$17\% \leq \eta < 55\%$	38%	34
High	$\eta \geq 55\%$	66%	7

To categorize the images, we calculated the average detailedness minus/plus the standard deviation as two boundary points, which gave us 17% and 55%, respectively. As a result, all the images with detailedness below 17% were categorized as class Low. Class Medium contains the images

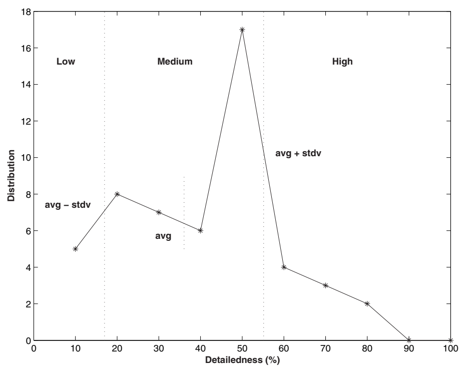

Fig. 2. Images distribution over detailedness

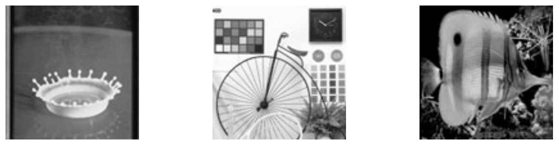

Fig. 3. Left to right: low, medium and high detailedness

with detailedness between 17% and 55% and finally, class High includes the images that have detailedness higher than 55%. Table II shows the range of detailedness, average detailedness and number of images in each class of detailedness. Figure 3 shows typical images for each class of detailedness.

III. GRAY-SCALE MORPHOLOGY

In this section, we briefly review the two basic mathematical operators of gray-scale morphology: Dilation (grows image regions) and Erosion (shrinks image regions). These two operators are fundamental building blocks for gray-scale morphology, based on which many morphological algorithms have been developed. Let $f(x, y)$ and $k(x, y)$ be input image function and SE function, respectively where $f : Z^2 \rightarrow Z$ and $k : Z^2 \rightarrow Z$. Gray-scale dilation, denoted by $f \oplus k$ is defined as [14]:

$$(f \oplus k)(x, y) = max\{f(x - s, y - t) + k(s, t) \\ |(x - s), (y - t) \in D_f; (s, t) \in D_k\} \quad (1)$$

Gray-scale erosion, denoted by $f \ominus k$ is defined as [14]:

$$(f \ominus k)(x, y) = min\{f(x + s, y + t) - k(s, t) \\ |(x + s), (y + t) \in D_f; (s, t) \in D_k\} \quad (2)$$

where D_f and D_k are the domains of f and k, respectively. The morphological gradient is computed as:

$$g(x, y) = (f \oplus k)(x, y) - (f \ominus k)(x, y). \quad (3)$$

In this paper, we consider non-flat 3×3 SEs.

IV. WINDOW MEMOIZATION TECHNIQUE

In this section, we present window memoization technique without considering opposite windows. The main idea behind window memoization is that if a calculation has been performed on a pixel neighborhood (window) then when we encounter an *identical* window in the future, we can reuse the previously computed result. The goal is to increase performance by reducing the total number of calculations that are performed on an image. To maximize the performance improvement, we want to maximize the percentage of windows that are able to reuse previously computed results (reuse rate) and minimize the cost of reusing a result.

As with memoization techniques in software or hardware, window memoization uses a memory array (reuse table (RT)) to store the neighborhoods and their results after performing the calculations. As shown in Figure 4, pixels of each new window are compared to the pixels stored in the reuse table. If the new window matches a window stored in the reuse table (hit), the result is looked up from the reuse table and the calculations for the new window are skipped. Otherwise (miss), the calculations are performed on the new window and the reuse table is updated with the produced result.

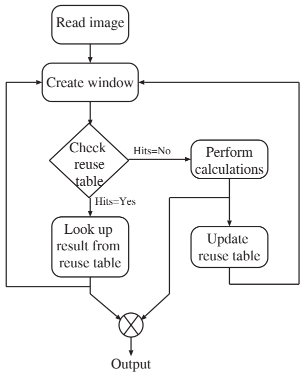

Fig. 4. Flowchart of window memoization

Each reuse table element contains three fields: valid bit, full-key and result. The valid bit indicates whether the data stored in this address is valid. The full-key represents the stored neighborhood. Finally, the outcome of applying the mask operations to the window is stored in the result field. To make full-key unique for each window of size $n \times n$ in the image, we shift and OR the pixels in the window such that they build a $8 \times n \times n$ bit number (8 bits per pixel). Table III shows the window memoization algorithm.

RT-size is the size of reuse table, '$\ll n$' represents an operator that shifts the operand n-bits to the left, and 'OR' is a logical *or*. To gain better performance for window memoization, it is crucial to speed up the lookup operations. To generate the full-key for each window faster, we benefit from the overlap between the neighboring windows to build the full-key incrementally as the convolution mask moves

Table III. Algorithm for Window Memoization

```
1. input an image 'I'
2. initialize full-key
3. create $n \times n$ window, W
4. update full-key:
       for each pixel W(i,j) in the window calculate:
          full-key = (full-key $\ll$ 8 ) OR W(i,j)
5. calculate the hash-key:
       hash-key = mod (full-key, RT-size)
6. if RT[hash-key].valid = 0; go to 9
7. if RT[hash-key].full-key $\neq$ full-key; go to 9
8. look up the result:
       result = RT[hash-key].result; go to 3
9. apply mask operation on W
10 update reuse table:
       RT[hash-key].result = result
       RT[hash-key].full-key = full-key
       go to 3
```

across the image.

For every window, another key (hash-key) is produced by the hash-key generator by calculating the remainder of the full-key divided by the size of the reuse table. The hash-key is used as an address to the reuse table. Whenever a new neighborhood is available, the full-key is compared to the related field of the reuse table where the hash-key points to. If they match, the result is looked up from the table. Otherwise, the mask calculations are performed on the original window (i.e. with 256 gray levels) and the corresponding location in the hash table is updated with the new full-key and result.

With a naive matching algorithm, the reuse rate will be low, since it is unlikely that a window of pixels will exactly match a previously encountered window. We increase the reuse rate through *multi-thresholding*: rather than require that the pixels match exactly, we reduce the number of gray levels of the windows when doing the comparison. Decreasing the precision of the match (using fewer gray levels when comparing pixels) increases the reuse rate and thereby the performance but at the potential cost of producing incorrect results. The fewer the gray levels used in testing for a match, the greater the probability that the matching result found in the reuse table will differ from the actual result that would be calculated for the window. To minimize the loss of accuracy, we reduce the number of gray levels only for matching the windows; the actual calculations are done with a full range of 256 gray levels. To apply multi-thresholding to window memoization, step 4 of the above mentioned algorithms is modified as shown in Table VI where 2^p is the number of gray levels used for matching and '$\gg n$' represents an operator that shifts the operand n-bits to the right.

To find the optimal values for the number of gray levels for matching and the reuse table size for our technique, we evaluated the performance and the results accuracy with respect to different numbers of gray levels for matching (2 to 256) and hash table sizes (4 entry up to 64k entry). We found that using 16 gray levels for comparing the windows leads to high reuse rates (and thus high speedups) while preserving the accuracy of the results. Although lower gray levels for matching results in very high speedups but it increases the error in the results. Also, our experiments with different sizes of reuse tables showed that 4K entries is an optimal size. Smaller reuse tables decrease the reuse rate and thus the speedup. Although using larger reuse tables increases the reuse rate, it decreases the actual speedups (larger reuse tables tend to be located in larger cache memories, which belong to the lower levels of the cache hierarchy).

Table IV. Step 4 of Window Memoization Algorithm (Table III), modified for multi-thresholding

```
4. update full-key:
       for each pixel W(i,j) in the window calculate:
       $W'(i,j) = W(i,j) \gg 8 - p$
       full-key = (full-key $\ll$ p ) OR $W'(i,j)$
```

We measure the accuracy of window memoization for an algorithm (e.g. morphological gradient) by comparing the output image against a reference image calculated by a conventional implementation of the algorithm (e.g. morphological gradient without memoization). To measure the difference between two images, we use the misclassification error (ME, Equation 4). The misclassification error calculates the percentage of the background pixels that have been assigned to foreground incorrectly and vice versa [15].

$$ME = 1 - \frac{|B_{Ref} \cap B_{Test}| + |F_{Ref} \cap F_{Test}|}{|B_{Ref}| + |F_{Ref}|}. \quad (4)$$

In Equation 4, B_{Ref} and F_{Ref} are the reference edge map background and foreground, respectively; and B_{Test} and F_{Test} are the background and foreground of the result produced by window memoization, respectively. For our simulations, we implemented a morphological gradient based on an arbitrary non-flat 3×3 SE and we ran the simulations on a $2GHz$ Pentium 4 server. The average reuse rates, accuracy and the corresponding speedups are given in Table V. In the table, the average elapsed time in millisecond for morphological gradient without and with window memoization technique are shown as T_1 and T_2, respectively.

Table V. Results for WM for different classes of 512×512 images

Class	T_1	T_2	Speedup	Reuse Rate(%)	Accuracy(%)
Low	40	17	2.50	92.76	99.90
Medium	46	34	1.42	57.76	99.86
High	49	46	1.17	32.58	99.70

Figure 5 shows the result of morphological edge detector without memoization (conventional algorithm) and with memoization (our algorithm) for sample images.

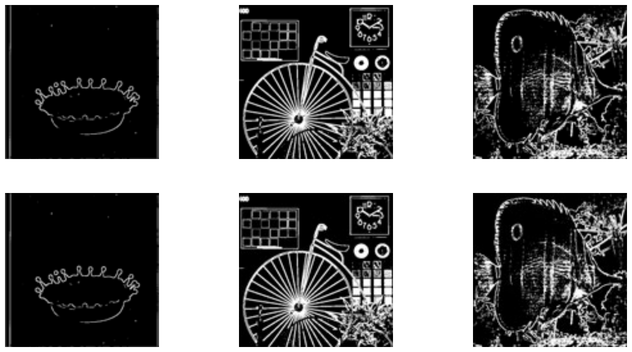

Fig. 5. Edge maps of sample images generated by conventional algorithm (top) and algorithm with WM (bottom)

V. OPPOSITE WINDOWS

Opposition-Based learning (OBL) scheme was introduced by Tizhoosh [1] to enhance the learning algorithms by considering the opposite estimation, opposite weight and counter-action for the problem under study. For example, in reinforcement learning, OBL requires that whenever an agent takes an action, it should take into account the opposite action and/or opposite state. This shortens the state-space traversal and accelerates the converge [16]. OBL is deﬁned as follows:

Opposition-based learning: Let $f(x)$ be the function under study and $g(.)$ be an evaluation function. If $x \in [a, b]$ is an initial guess and $\breve{x}$ is its opposite value, then in every iteration both $f(x)$ and $f(\breve{x})$ are calculated. If $g(f(x)) \geq g(f(\breve{x}))$ the learning continues with x, otherwise with $\breve{x}$. As an example, the opposite of a number is deﬁned as [1]:

Definition: Let $x \in \mathcal{R}$ be a real number deﬁned on a certain interval: $x \in [a, b]$. The *opposite number* $\breve{x}$ is deﬁned as: $\breve{x} = a + b - x$.

We would like to extend the window memoization technique in such a way that it beneﬁts from OBL. The goal is to increase the reuse rate, by considering opposite windows, whenever the lookup is performed. To improve the speedups achieved by opposition-based window memoization, it is necessary to minimize the cost required to calculate the mask response for the opposite of the window in focus. We deﬁne the opposite of a window from spatial perspective, which means the opposite of a given window has the same pixels as the given neighborhood, with different assigned locations. The idea is that if we know the response of a mask on a window, we would like to know the response of the mask on the opposite window without applying the mask on the opposite window. This will lead to less number of calculations required by window memoization technique.

The opposite relations is deﬁned based on the nature of the algorithm under study. Opposition can be understood in a set-theoretical context (opposite gray-levels≡logical complement), as a directional category (opposite window≡window with opposite gradient), or in any other meaningful way (e.g. a homogenous window is the opposite of a noisy/edgy window). For our case study, which is morphological gradient with non-ﬂat SE, we consider special cases of the SE function, where the function is symmetric, diagonally, horizontally and vertically. In each case, we deﬁne opposite window accordingly. For a 3 × 3 SE that is vertically symmetric, we have: $k(s, t) = k(-s, t)$ where $-1 \leq s, t \leq 1$ and $k(s, t)$ is the corresponding SE function. In this case the simplest way to deﬁne opposite window is such that both the original window and its opposite gives the same response to the SE function. This leads to the following deﬁnition:

Opposite window for vertically symmetric windows: Assuming that $W(x, y)$ is a vertically symmetric window from an image $f(x, y)$, the opposite window, $\breve{W}(x, y)$, is deﬁned as:

$\forall i, j$ *($i = x$ or $i = -x$) & ($j = y$)* $\Rightarrow \breve{W}(i, j) = W(x, y)$

This means that if we swap either of pixels in top row with the ones in bottom row along the same column, we get an opposite window of the original neighborhood. Based on this deﬁnition, for each window there are 8 different opposite windows (3 pixels to exchange: 2^3). Thus far, we have deﬁned opposite window ($\breve{W}(x, y)$) such that for a SE function, $k(s, t)$ we have:

$$\begin{aligned}(W \oplus k)(x, y) - (W \ominus k)(x, y) &= \\ (\breve{W} \oplus k)(x, y) - (\breve{W} \ominus k)(x, y) & \qquad (5)\end{aligned}$$

In other words, the morphological gradient gives the same result to both window and its opposite window. From window memoization point of view, this means that a hit occurs when either a window matches the one previously stored in the reuse table or an opposite window matches the original window previously stored in the reuse. In other words, for a given window W and its opposite $\breve{W}$, identical full-keys must be generated, resulting in accessing the same location in the reuse table for both W and $\breve{W}$. Theoretically, this will increase the reuse rate in comparison to WM without OBL, and hence the speedup will increase as well. To consider opposite windows in window memoization, step 4 of the algorithm in table III is modiﬁed as shown in Table VI. In the table, 2^p is the number of gray levels used for matching.

Table VI. Step 4 of Window Memoization Algorithm (Table III), modiﬁed for opposite windows

```
4. update full-key:
       sort each pairs of pixels in top row and bottom row,
       W(i,j) and W(-i,j) in the window:
     for each pixel W(i,j), in the window calculate:
       $W'(i,j) = W(i,j) \gg 8 - p$
       full-key = (full-key $\ll$ p ) $OR$ $W'(i,j)$
```

Using this full-key generator, we applied window memoization to morphological gradient, which uses a vertically symmetric non-flat SE. We ran the simulations for the same set of images on the same workstation as for our first simulations. The average reuse rates, the accuracy and the corresponding speedups are given in Table VII. In the table, the average elapsed time in millisecond for morphological gradient without and with OB-WM are shown as T_1 and T_2, respectively.

Table VII. Results for OB-WM for different classes of 512×512 images

Class	T_1	T_2	Speedup	Reuse Rate (%)	Accuracy(%)
Low	40	15	2.80	93.32	99.86
Medium	46	34	1.47	61.36	99.80
High	49	47	1.18	36.61	99.59

As it can be seen from tables V and VII, considering opposite windows in window memoization has increased the reuse rate by 0.56%, 3.61% and 4.03% for class low, medium and high, respectively. The corresponding increase rates for speedups are: class low: 30%, medium: 5% and high: 1%. We implemented the OB-WM based on diagonally and horizontally symmetric SEs. Although we achieved up to 10% increase in reuse rate, due to implementation cost, speedups did not improve.

Implementation Issues - We implemented the WM and OB-WM techniques in C++ to measure the relative performance of morphological edge detection with and without the proposed techniques. We performed inline expansions of the functions, and maximized the compiler optimization settings. We observed that reuse rates do not necessarily indicate the actual speedups obtained in software implementation of the window memoization technique. The reason is that any implementation of the technique will require a memoization mechanism that performs searching, storing and retrieving the results, which consumes time.

For our results, it is seen that for reuse rates less than about 15%, the speedup becomes less than 1. To overcome the performance overhead caused by WM or OB-WM, the memoization mechanism must be very efficient and fast. Otherwise, the time that it takes to reuse a result will be longer than the original calculations. Our current version of memoization mechanism is efficient for both WM (for any non-flat SE) and OB-WM for vertically symmetric SEs.

VI. CONCLUSION

The window memoization technique reveals the fact that image data locality can be exploited to improve the overall performance of gray-scale morphological algorithms that use non-flat SEs significantly with negligible penalty in accuracy. Using 16 gray levels for matching with 4k entry reuse table, we obtained speedups of up to 2.84 for 512×512 images. We also applied OBL, a new learning scheme to window memoization, which led to speedups of up to 3.38. Window memoization can be applied to any convolution algorithm in the spatial domain where identical input pixels produce the same outputs. However, depending on the complexity of the algorithms and the results type (e.g. binary or non-binary), the performance gain and the result accuracy might not be identical for different algorithms. In future work, we will probe different hashing functions to improve the memoization mechanism for both WM and OB-WM. We will apply the technique to other class of spatial domain algorithms (e.g. filters) to investigate the result reuse rate and accuracy for algorithms that produce non-binary results.

VII. REFERENCES

[1] H. R. Tizhoosh, "Opposite-based learning: A new scheme for machine intelligence," in *International Conference on Computational Intelligence for Modelling Control and Automation - CIMCA-2005*, 2005, vol. I, pp. 695–701.

[2] D. Michie, "Memo functions and machine learning," *Nature*, vol. 218, pp. 19–22, 1968.

[3] S. Richardson, "Exploiting trivial and redundant computation," in *11th Symposium on Computer Arithmetics*, 1993, pp. 220–227.

[4] A. Sodani and G. S. Sohi, "Dynamic instruction reuse," in *ISCA-97*, 1997, pp. 194–205.

[5] D. Citron, D. Feitelson, and L. Rudolph, "Accelerating multi-media processing by implementing memoing in multiplication and division units," in *ASPLOS-VIII*, 1998, pp. 252–261.

[6] J. Huang and D. J. Lilja, "Extending value reuse to basic blocks with compiler support," *IEEE Trans. on Computers*, vol. 49, pp. 331–347, 2000.

[7] K. M. Kavi and P. Chen, "Dynamic function result reuse," in *ADCOM-03*, 2003.

[8] J. P. Shen and M. H. Lipasti, *Modern Processor Design*, McGraw-Hill, 2004.

[9] W. Wang, A. Raghunathan, and N. K. Jha, "Profiling driven computation reuse: An embedded software synthesis technique for energy and performance optimization," in *VLSID-04 Design*, 2004, p. 267.

[10] Y. Ding and Z. Li, "Operation reuse on handheld devices," in *LCPC-03*. 2003, vol. 2958 / 2004, pp. 273–287, Springer-Verlag GmbH.

[11] J. Y. Gil and R. Kimmel, "Efficient dilation, erosion, opening, and closing algorithms," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24-12, pp. 1606–1617, 2002.

[12] H. Park and R. T. Chin, "Decomposition of arbitrary shaped morphological structuring elements," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 17-1, 1995.

[13] O. I. Camps and T. Kanungo, "Gray-scale structuring element deomposition," *IEEE Transactions on Image Processing*, vol. 5-1, pp. 111–120, 1996.

[14] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, Prentice Hall, 2002.

[15] M. Sezgin and B. Sankur, "Survery over image thresholding techniques and quantitiative performance evaluation," *Electronic Imaging*, vol. 13(1), pp. 146, 2004.

[16] H. R. Tizhoosh, "Opposition-based reinforcement learning," in *Journal of Advanced Computational Intelligence and Intelligent*, 2006, vol. 10, no. 4, pp. 578–585.