Perfect Window Memoization: A Theoretical Model of an Optimization Technique for Image Processing Algorithms

Farzad Khalvati[1] **and Hamid R. Tizhoosh**[2]
[1,2]Department of Systems Design Engineering, University of Waterloo
Waterloo, Ontario, Canada
Email: [1] farzad.khalvati@uwaterloo.ca, [2] tizhoosh@uwaterloo.ca

Abstract— *This work presents* Perfect Window Memoization*; a high-level processing model that gives an estimation (upper-bound) of performance gain for an image processed in software or hardware, obtained by eliminating the computational redundancy of the image. We show mathematically, supported by experimental data, that the computational redundancy of an image is, in fact, inherited from two basic data redundancies of the image; coding and interpixel redundancy. This is a simple, yet a revealing concept to use in practice by which images can be categorized based on their potential performance gain in software and hardware by only their fundamental redundancies, with no need to implement a mechanism to actually exploit the computational redundancy in software or hardware. The proposed model can be used as a useful tool in analyzing images from the performance perspective in the early stages of designing an optimization technique.*

Keywords: Coding redundancy, interpixel redundancy, window memoization, Performance optimization, Markov Model

1. Introduction

High-performance implementations of image processing algorithms are an important issue in real-time applications. The high volume of image data combined with the requirements of real-time systems makes it both challenging and crucial to optimize the image processing algorithms. As the image resolution and the complexity of algorithms increase, the conventional approaches of software implementations of image processing algorithms are not able to handle these challenges. One clear example is medical imaging where high performance image processing is a necessary requirement to maintain high patient throughput and thus, high quality care. The urgent need for optimizing the medical imaging algorithms have been frequently reported from both industry and academia. As an example, the processing times of 1 minute [1], 7 minutes [2], and even 60 minutes [3] are quite common for processing volume data in medical imaging. Considering the high volume of data to be processed, these reported long processing times introduce significant patient throughput bottlenecks that have a direct negative impact on access to timely quality medical care. Therefore, accelerating medical imaging algorithms will certainly have a significant impact on the quality of medical care by shortening the patients' access time. Other examples include security, navigation, multimedia, industrial inspection, and astrophysics where high-performance image processing is crucial to meet performance requirements.

Generally, there are two fundamental methods for algorithm optimization: algorithmic optimization and parallel processing. Algorithmic optimization which is a fundamental technique aims at reducing the number of operations used in an algorithm by decreasing the number of steps that the algorithm requires to take. Parallel processing breaks down an algorithm into processes that can be run in parallel while preserving the data dependencies among the operations. In both schemes, the building blocks of the optimization methods are the operations where the former reduces the number of operations and the latter runs them in parallel. In both cases, the input data is somewhat ignored. In other words, the algorithmic optimization will be done in the same way regardless of the input data type. Similarly, running the operations in parallel is independent of the input data. *Window Memoization*, introduced in [4], is a new optimization technique which can be used in conjunction with either of these two optimization methods. This new technique considers the characteristics of the input data, image data, to reduce the number of operations.

Window memoization is based on *reuse* or *memoization* concept. Memoization is an optimization method for computer programs which avoids repeating the function calls for previously seen input data. Window memoization applies the general concept of memoization to image processing algorithms by exploiting the image data redundancy and increases the performance of image processing significantly [4]. The main initiative behind window memoization is to reduce the amount of computations that an image processing algorithm must perform. The underlying basis of reduction is to remove the *redundant computations*; the computations that are not necessary to perform in order to complete an image processing task. Analogous to image compression algorithms that exploit data redundancy to reduce the size of images, computational redundancy can be used to reduce the amount of computations and hence, to improve the performance of image processing algorithms.

In this work, we investigate the underlying roots of computational redundancy of image data. In doing so, first we introduce a theoretical model for window memoization, *Perfect Window Memoization*, which predicts the upper-bound for the potential performance gain achieved by window memoization in software and hardware. We define computational redundancy as a characteristic of an image which is exploited by perfect window memoization to improve performance. Image data has two principal types of redundancy: coding redundancy and interpixel redundancy. Coding redundancy is due to the fact that images are usually represented using more bits per pixel than is actually needed. Interpixel redundancy is a result of the correlation among neighboring pixels. Our studies show that the coding and interpixel redundancy of an image has a mathematical relationship with the computational redundancy of the image. In other words, we show that the coding and interpixel redundancy of an image are the root causes of having similar windows across the image which is exploited by the window memoization technique to improve the performance of image processing in both software and hardware. This is a simple, yet revealing concept to use in practice. Images can be categorized based on their potential performance gain in software and hardware by only their coding and interpixel redundancy, with no need to actually implement the window memoization technique. Coding and interpixel redundancy are fundamental characteristics of images and hence, they are independent of any implementation artifact. Therefore, the relationship between the coding/interpixel redundancy and the performance improvement obtained by window memoization can be used as a useful tool in analyzing images from the performance perspective in the early stages of designing an optimization technique.

The organization of the remaining of this paper is as follows. Section 2 provides a brief background review on local image processing algorithms, image data redundancy, and the window memoization technique. Section 3 presents Perfect Window Memoization. Sections 4 discusses the relationship between the computational redundancy of an image versus the data redundancy of the image. Sections 5 and 6 present the results and conclusion of the paper, respectively.

2. Background and Related Work

In this section, a brief background review on the following subjects are presented: local image processing algorithms, image data redundancy, and window memoization technique.

2.1 Local Image Processing Algorithms

Although window memoization can be applied to any image processing algorithm, local algorithms, in particular, are the first to adopt the technique. Local processing algorithms that mainly deal with extracting local features in image (*e.g.* edges, corners, blobs) are increasingly used in many image processing applications such as medical imaging, texture recognition, scene classification, and robot navigation. The reason for popularity of these algorithms is that using local features of an image overcomes the need for high-level algorithms where a semantic-level understanding of the image is required [5]. The main drawback for local algorithms is that they are usually computationally expensive; a set of calculations must be repeated numerous times. The local algorithms use a small neighborhood of a pixel in an image to process and produce a new gray level for the image in the location of the pixel. The size of local windows can vary based on the algorithm but for most algorithms the local windows contain 9 pixels (3×3 pixels) or 25 pixels (5×5 pixels). A local processing algorithm applies a set of operations, which is called *the mask operations set* (f), to each window of pixels (w_{ij}) in the image to produce the *response* (r_{ij}) of the window.

$$r_{ij} = f(w_{ij}) \tag{1}$$

As equation 1 indicates, in local processing algorithms, the response of each local window r_{ij} only depends on the pixels in the local window w_{ij}.

2.2 Image Data Redundancy

Image processing exploits two major types of redundancy in image data which are quantifiable: *coding redundancy*, and *interpixel or spatial redundancy* [6]. In the following sections, we present an overview of these data redundancies.

2.2.1 Coding Redundancy

An image is said to contain coding redundancy if the number of bits per pixel that is required to represent the image is higher than is necessary [6]. To remove coding redundancy in an image, the gray levels of the image are encoded in a way such that a higher number of bits are assigned to less probable gray levels and vice versa. The coding redundancy of an image is measured based on the *entropy* of the image. Entropy is the average information of an image per pixel, which is calculated as [6]:

$$H = -\sum_{i=0}^{GL-1} P_i \times log_2 P_i \tag{2}$$

where GL and P_i are the number of gray levels and the probability of occurrence of gray level i in the image, respectively. For an image with GL gray levels, the sum of the average coding redundancy per pixel (C_r) and average information per pixel (entropy H) is a constant value [7]:

$$C_r + H = log_2(GL) \tag{3}$$

In other words, for an image with a given number of gray levels (GL), the entropy of the image (H) determines the coding redundancy of the image (C_r).

$$C_r = log_2(GL) - H \tag{4}$$

Coding redundancy has been exploited in several image compression algorithms to reduce the size of the image. Huffman coding, Golomb coding, and arithmetic coding are among the common lossless compression algorithms that benefit from coding redundancy in images [6].

2.2.2 Interpixel Redundancy

When measuring the coding redundancy of an image, it is assumed that the pixels in the image are uncorrelated. However, in real-world images, the neighboring pixels are correlated because they usually belong to one object or background with similar gray levels. The correlations among image pixels, which result from the structural or geometrical relationships between the objects in the image lead to interpixel redundancy [6]. As a result, for many pixels, much of information that the pixel carries is redundant and it can be predicted with a reasonably high accuracy from the values of its neighboring pixels.

Markov model is usually used to measure interpixel redundancy in an image. Markov model assumes that the probability that a pixel at a location has a certain gray level is a function of the gray levels of some number of neighbouring pixels [8]. The number of neighbouring pixels is the order of the Markov source. When pixels in an image are assumed to be independent random variables, in fact it is assumed that the image is a Markov source of 0^{th} order. This is the case when coding redundancy is calculated. A k^{th} order Markov source considers each window of $k+1$ pixels in an image as a *($k+1$)-dimension gray-level vector*. If the $k+1$ neighbouring pixels share information with each other, then the entropy of the k^{th} order Markov source will be less than that of the 0^{th} order Markov source, which is the original image. The decrease in entropy is considered to reflect the removal of interpixel redundancy among the pixels of windows with $k+1$ pixel size:

$$IP_r = H - H^k \tag{5}$$

In equation 5, IP_r is the interpixel redundancy, H and H^k are the entropies of the original image and the k^{th} order Markov source of the image, respectively. The interpixel redundancy calculated based on Markov model depends on the size of windows of pixels, based on which gray-level vectors are built.

There are many lossless image compression algorithms that take advantage of interpixel redundancy in images including run-length coding, difference coding, lossless predictive coding, LZW coding, and vector coding. Lossy predictive coding is an example for lossy image compression algorithms, which exploit interpixel redundancy [6] [8].

2.3 Window Memoization Technique

Introduced in [4], window memoization is an optimization technique for local image processing algorithms that exploits the data redundancy of images to reduce the processing time. The general concept of memoization in computer programs has been around for a long time where the idea is to speed up the computer programs by avoiding repetitive/redundant function calls [9] [10] [11]. Nevertheless, in practice, the general notion of memoization has not gained success due to two main reasons: 1) the proposed techniques usually require detailed profiling information about the runtime behaviour of the program which makes it difficult to implement [12], and 2) the techniques are usually generic methods which do not concentrate on any particular class of input data or algorithms [13]. In contrast, in designing the window memoization technique in [4], the unique characteristics of image processing algorithms have been carefully taken into account to enable window memoization to both be easy to implement and improve the performance significantly.

Window memoization minimizes the number of redundant computations by identifying similar groups of pixels in the image using a memory array, *reuse table*, to store the results of previously performed computations. When a set of computations has to be performed for the first time, they are performed and the corresponding results are stored in the reuse table. When the same set of computations has to be performed again, the previously calculated result is reused and the actual computations are skipped. This eliminates the redundant computations and leads to high speedups.

Implementing window memoization in software speeds up the computations required to complete an image processing task. In the previous work [4], an optimized software architecture for window memoization has been presented where the typical speedups range from 1.2x to 7.9x with a maximum factor of 40x. In hardware, window memoization yields high speedups with an overhead in hardware area that is significantly less than that of conventional performance improvement techniques. In the previous work [14], an optimized hardware architecture for window memoization has been presented where the typical and maximum speedup factors are 1.6x and 1.8x, respectively, with 40% less hardware in comparison to conventional optimization techniques.

3. Perfect Window Memoization

Perfect window memoization is an abstract version of window memoization which makes it easier to model the behaviour of data. It is a processing model for local image processing algorithms that detects and eliminates the computational redundancy of input image to improve the performance. In order to define this model, first, we define the concept of Gray-level vector or *symbol* in image. Afterward, we describe the model and as a result, we present a definition of computational redundancy which is more accurate than its previous definition in [4].

3.1 Symbols in Image

The windows of pixels are essential parts of designing or implementing local processing algorithms. When dealing with the windows of pixels in an image as the building blocks of the image, it is more convenient to consider the windows of pixels as a higher dimension gray levels or *gray-level vectors*. Gray-level vectors, which we call *symbols*, are defined based on the size of local windows. For windows of $m \times m$ pixels, an m^2-dimensional symbol represents all the windows in the image whose corresponding pixels are identical. In other words, a symbol represents all identical windows in the image. A window win belongs to (or matches) a symbol sym if each pixel in win is equal to the corresponding pixel in sym:

$$\forall pix \in win, \forall pix' \in sym,\ pix = pix' \implies win \in sym \tag{6}$$

where pix and pix' are corresponding pixels of window win and symbol sym, respectively. In the above definition, a window belongs to a symbol if all the pixels in the window are identical to the corresponding pixels in the symbol. We relax the equality requirement such that similar but not necessarily identical windows may belong to one symbol:

$$\forall pix \in win,\ \forall pix' \in sym, MSB(d, pix) = pix' \implies win \in sym \tag{7}$$

where $MSB(d, pix)$ represents d most significant bits of pixel pix in window win and pix' has d bits. By reducing d, windows that are similar but not identical are assigned to one symbol. This will potentially introduce inaccuracy in the result of the algorithm to which perfect window memoization is applied. However, in practice, for $d >= 4$, the accuracy loss in responses is usually negligible [4].

Ignoring the geometry of objects in an image, for a given local window size, the image can be characterized by the probability of occurrences of symbols in the image. Assume that a discrete random variable, s_i in the interval $[0, s)$, represents the symbols of an image. The probability of occurrence of each symbol (s_i) in the image is:

$$P(s_i) = \frac{n_i}{n},\ i = 0, 1, 2, ..., s-1 \tag{8}$$

where s is the total number of symbols in the image, n_i is the number of times that the i^{th} symbol appears in the image, and n is the number of total windows in the image.

3.2 Perfect Window Memoization

We define an abstract processing model, *perfect window memoization*, which minimizes the number of redundant mask operations sets performed on an image. Perfect window memoization is a high level model that gives an estimation (upper-bound) of performance gain in software and hardware, obtained by eliminating the redundant mask operations sets.

In local image processing algorithms, the response of the mask, r_{ij}, solely depends on the pixels in the local window, w_{ij}, covered by the mask (equation 1). All the windows in the image that contain similar pixels are identified by one symbol, s_i. Thus, for a given algorithm, a symbol s_i will produce the same response for n_i times. This means that much of the mask operations sets that are applied to symbol s_i are unnecessary or redundant.

Perfect window memoization uses a *reuse table* to store symbols, s_i, and the corresponding responses, r_i. The reuse table in perfect window memoization is in fact a *perfect cache*. The perfect cache never inserts the same data again. In other words, each symbol is inserted only once and when inserted, it is never evicted. Although perfect cache is a theoretical model, similar to processor cache modelings, it is a well accepted model for the behaviour of data with regard to redundancy (or locality).

When perfect window memoization receives a window for the first time, there is no matching symbol in the reuse table and thus, a *miss* occurs. Therefore, it applies the mask operations set to the window and inserts its matching symbol and its response into the reuse table. Upon encountering the same window again in the future, this time there is a matching symbol in the reuse table and thus, a *hit* occurs. As a result, perfect window memoization does not perform the mask operations set on the recent window. Instead, it looks up or *reuses* the corresponding response from the reuse table. In other words, each symbol is inserted into the reuse table only once and the mask operations set is applied to its corresponding windows only once regardless of the probability of occurrence of the symbol in the image. This means that when a symbol is inserted into the reuse table, it is never replaced with another symbol. Eventually, all symbols of an image along with their responses will be stored in the reuse table.

For a symbol whose windows appear in the image n_i times, $n_i - 1$ mask operations sets can be skipped and their corresponding responses can be reused. In other words, the number of hits for the windows of a symbol with the population of n_i will be $n_i - 1$. The number of hits for all windows in the image will be $n - s$ where n and s are the number of windows and symbols in the image, respectively. The *hit rate* (HR) is defined as the percentage of the times that the incoming windows find a matching symbol in the reuse table and therefore, reuse their previously calculated responses.

In order to insert all symbols of an image into the reuse table, the reuse table size must be equal to the number of symbols in the image. In practice, however, the required size may be too large to afford. Therefore, perfect window memoization explores inserting a variable portion of symbols in the image into the reuse table. The symbols with higher probabilities of occurrence will contribute more to performance gain if inserted into the reuse table because for such symbols, higher number of mask operations sets can be skipped.

For a reuse table of m entry size, the number of hits will be the sum of populations of the m most frequent symbols (the total number of windows that belong to m inserted symbols) minus the number of inserted symbols (or the number of misses), which is m.

$$HR(m) = \frac{n_0 + n_1 + ... + n_{m-1}}{n} - \frac{m}{n} \quad (9)$$

where $n_0, n_1, n_2, ..., n_{m-1}$ are the populations of m most frequent symbols in the image, and n and m are the total number of windows in the image and the number of inserted symbols (or the reuse table size), respectively. Although hit rate is a discrete function of m (the reuse table size), to simplify the analysis, we model its behaviour with a continuous function. Assume that P is a continuous function that models the probability density function (PDF) of the image symbols, which have been sorted in descending order. We can write:

$$HR(m) = \int_0^{m-1} P(x)dx - \frac{m}{n} \quad (10)$$

where $\int_0^{m-1} P(x)dx$ is the probability of m most frequent symbols (or inserted symbols).

In contrast to normal caches in processor architecture where the cache size is always smaller than the number of unique entries, in window memoization, when input images are simple (*i.e.* small number of symbols), it is possible that the reuse table size is larger than the total number of symbols in the image (*i.e.* $m > s$). In this case, the reuse table will not be filled with the symbols, which means that the total number of misses will be equal to the total number of symbols (s), rather than the reuse table size (m). Thus, a more accurate equation for hit rate is:

$$HR(m) = \int_0^{m-1} P(x)dx - \frac{min(m, s)}{n} \quad (11)$$

In other words, if $m \leq s$ then m indicates the total number of misses. Otherwise, the total number of misses will be s.

Ignoring some details of implementations of perfect window memoization in software and hardware, $HR(m)$ indicates what percentage of mask operations sets can be skipped and their corresponding responses can be reused, given that m most frequent symbols have been inserted into the reuse table. The hit rate of perfect cache is independent of the reuse mechanism implemented in software or hardware. It only depends on the characteristic of the image data and the size of perfect cache. For a given reuse table size, it represents the percentage of computations that can be skipped. This is in fact the definition of *computational redundancy* [4] with the exception that it is now defined for a given reuse table size[1]. This is a more accurate definition of the computational redundancy because not only it considers the image itself, it also takes into consideration the amount of memory space allocated for detecting and eliminating the redundant computations which is an inseparable part of any memoization technique. Thus, we define the computational redundancy, $Comp_r$ as:

$$Comp_r(m) = \int_0^{m-1} P(x)dx - \frac{min(m, s)}{n} \quad (12)$$

where m is the size of reuse table, $\int_0^{m-1} P(x)dx$ is the probability of m most frequent symbols (or inserted symbols), and s is the total number of symbols in the image.

4. Computational Redundancy versus Data Redundancy

Perfect window memoization benefits from a characteristic of image data, computational redundancy, to improve the performance of local image processing algorithms. Our goal in this section is to show mathematically that, for a given reuse table size, the sum of the coding and interpixel redundancy of an image has a positive relationship with the computational redundancy of the image or:

$$(C_r + IP_r) \propto^{+} Comp_r(m) \quad (13)$$

The relation above enables us to categorize images based on their potential performance improvement obtained by window memoization in software and hardware without actually implementing the window memoization technique. By combining equations 4 and 5, the left-hand-side of relation 13 can be written as:

$$C_r + IP_r \quad = \quad log_2(GL) - H^k \quad (14)$$

where GL is the number of gray levels of the image and H^k is the entropy the k^{th} order Markov source of the image. This gives:

$$(C_r + IP_r) \propto^{-} H^k \quad (15)$$

From relations 13 and 15, the problem to solve comes down to:

$$H^k \propto^{-} Comp_r \quad (16)$$

[1]In the previous definition of computational redundancy [4], the assumptions was $m >= s$ and thus, the computational redundancy would be independent of reuse table size (m) and equation 11 would become $Comp_r = HR = 1 - \frac{s}{n}$.

4.1 Exponential Model for the Probability of Symbols

P is the probability density function (or histogram) of symbols in the original image (Img), which are sorted in descending order. P has a peak at 0 and moving along X axis, it usually drops rapidly. P can be modeled by an exponential distribution with the area under curves of unity:

$$P(x) = \frac{1}{\lambda} e^{\frac{-x}{\lambda}} \tag{17}$$

where λ is the scale factor of the distribution. In order to verify that the probability density functions of image symbols are exponential (equation 17), we performed the curve fitting for all images of our dataset used in this research (i.e. 40 natural images of 512×512 pixels). The experimental data shows that the exponential curve models the probability density functions of image symbols very accurately. The average RMSE for 40 images is 0.54%.

4.2 Analytical Model of Entropy

Using a continuous model for the probability density function of symbols, P, we can calculate the continuous entropy of k^{th} order Markov source of the image based on its probability density function .

$$H^k = -\int_{-\infty}^{+\infty} P(x) \times log_2(P(x))dx \tag{18}$$

where $P(x) = \frac{1}{\lambda} e^{\frac{-x}{\lambda}}$. In equation 18, we can transform log_2 into the natural logarithm using:

$$\begin{aligned} log_2 P(x) &= log_2 e \times ln P(x) \\ &= log_2 e \times ln(\frac{1}{\lambda} e^{\frac{-x}{\lambda}}) \\ &= log_2 e \times [ln(\frac{1}{\lambda}) - \frac{x}{\lambda}] \end{aligned} \tag{19}$$

Substituting equations 17 and 19 into equation 18, we get:

$$\begin{aligned} H^k &= -\int_{0}^{+\infty} \frac{1}{\lambda} e^{\frac{-x}{\lambda}} \times log_2 e \times [ln(\frac{1}{\lambda}) - \frac{x}{\lambda}]dx \\ &= log_2 e \times (ln(\lambda) + 1) \end{aligned} \tag{20}$$

4.3 Analytical Model of Computational Redundancy

Substituting equation 17 in equation 12, we can calculate the computational redundancy of the image:

$$\begin{aligned} Comp_r(m) &= \int_{0}^{m-1} \frac{1}{\lambda} e^{\frac{-x}{\lambda}} dx - \frac{min(m,s)}{n} \\ &= (1 - e^{-\frac{m-1}{\lambda}}) - \frac{min(m,s)}{n} \end{aligned} \tag{21}$$

4.4 Final Relation

In this section, we show that the entropy of the k^{th} order Markov source of an image has a negative relationship with the computational redundancy of the image (i.e. relation 16). In other words, for two given images, we want to show:

$$H_1^k < H_2^k \implies Comp_{r1}(m) > Comp_{r2}(m) \tag{22}$$

Substituting equation 20 in the left hand side of the relation above gives:

$$\begin{aligned} log_2 e \times (ln(\lambda_1) + 1) &< log_2 e \times (ln(\lambda_2) + 1) \\ \implies \lambda_1 &< \lambda_2 \end{aligned} \tag{23}$$

For two given images and a given reuse table size, m, with the assumption that $m < s$, relation 23 and equation 21 leads to[2]:

$$\lambda_1 < \lambda_2 \implies Comp_{r1}(m) > Comp_{r2}(m) \tag{24}$$

which is the relation that we wanted to prove. In other words, the entropy of k^{th} order Markov source of an image has a negative relationship with the computational redundancy of the image for a given reuse table size m:

$$H^k \propto^- Comp_r(m) \tag{25}$$

Comparing the relation above and relation 15, we can conclude:

$$(C_r + IP_r) \propto^+ Comp_r \tag{26}$$

5. Results

For experimental results, we used 40 natural images of 512×512 pixels. The window size was 3×3 pixels. In equation 7, we chose d to be 4[3]. We ran experiments for all 40 images with different reuse table sizes of $1K$ to $512K$ (1K, 2K, 4K, 8K, 16K, 32K, 64K, 128K, 256K, and 512K). Each memory element contains a gray-level vector, which is 3×3 pixels of d bits. Figure 1 shows the results for two cases; reuse table sizes of $1K$ and $512K$ ($m = 1k$ and $m = 512k$).

In order to quantify the accuracy of relation 26 with respect to the experimental data, we calculate the percentage of cases that for any given pair of images out of 40 images, relation 26 holds[4]. In other words, for two given images, we verify that the following holds:

$$(C_{r1} + IP_{r1}) < (C_{r2} + IP_{r2}) => Comp_{r1} < Comp_{r2} \tag{27}$$

Table 1 shows the accuracy of experimental data with respect to relation 26. As it is seen, in the worst scenario, 87% of the cases matches the analytical model.

[2]It can be shown that the argument is true for $m \geq s$ as well.
[3]The whole discussion holds for $d \in \{2, 3, 4, ..., 8\}$
[4]The total number of pairs of images for 40 images is $\frac{40(40-1)}{2} = 780$

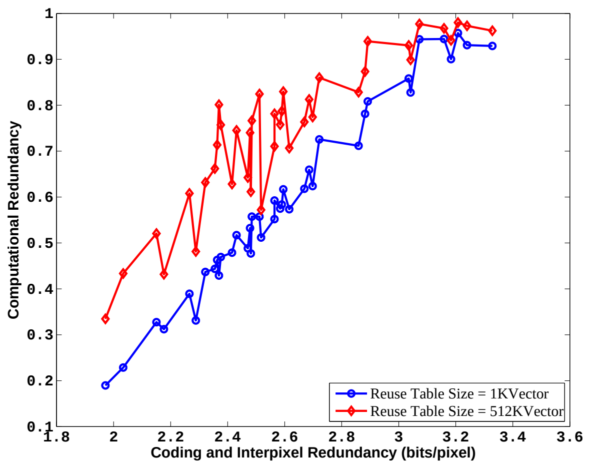

Fig. 1: Computational Redundancy versus Coding and Interpixel Redundancy

Table 1: Accuracy of experimental data for relation 26

Reuse Table Size	1K	2K	4K	8K	16K
Accuracy	96%	93%	91%	89%	88%
Reuse Table Size	32K	64K	128K	256K	512K
Accuracy	87%	87%	87%	87%	87%

6. Conclusion

In this paper, we introduced perfect window memoization; a high-level processing model for detection and elimination of redundant computations in local image processing algorithms. We refined the notion of computational redundancy in image data to present the behaviour of redundant computations in image processing more accurately. In previous work [4] [14], it has been shown that the computational redundancy of image data can be exploited, via the window memoization technique, to improve the performance of local image processing algorithms. Perfect window memoization gives the upper-bound of performance gain in software and hardware, obtained by the window memoization technique. We showed that the computational redundancy of an image is inherited from two principal redundancies in image data: coding redundancy and interpixel redundancy. It was shown that the amount of coding and interpixel redundancy of an image has a direct effect on the computational redundancy of the image. This leads to the fact that the coding and interpixel redundancy of an image has a positive relationship with the performance gain (speedup) obtained by exploiting the computational redundancy of the image in software and hardware. This is a simple, yet useful concept that can be utilized in design of data redundancy-based optimization techniques in image processing. The experimental data matches our analytical model of computational redundancy with very high accuracy.

References

[1] B. Haas, T. Coradi, M. Scholz, M. H. P. Kunz, U. Oppitz, L. Andre, V. Lengkeek, D. Huyskens, A. V. Esch, and R. Reddick, "Automatic segmentation of thoracic and pelvic CT images for radiotherapy planning using implicit anatomic knowledge and organ-specific segmentation strategies," *Phys. Med. Biol*, vol. 53, pp. 1751–1771, 2008.

[2] A. C. Hodgea, A. Fensterabc, D. B. Downeyb, and H. M. Ladakabcd, "Prostate boundary segmentation from ultrasound images using 2D active shape models: Optimisation and extension to 3D," *Computer methods and programs in biomedicine*, vol. 84, pp. 99–113, 2006.

[3] A. Gubern-Merida and R. Marti, "Atlas based segmentation of the prostate in MR images," in *MICCAI: Segmentation Challenge Workshop*, 2009.

[4] F. Khalvati, "Computational redundancy in image processing," Ph.D. dissertation, University of Waterloo, 2008.

[5] T. Tuytelaars and K. Mikolajczyk, "Survey on local invariant features," *FnT Computer Graphics and Vision*, vol. 1, no. 1, pp. 1–94, 2008.

[6] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*. Prentice Hall, 2008.

[7] M. Sonka, V. Hlavac, and R. Boyle, *Image Processing, Analysis, and Machine Vision*. PWS, 1999.

[8] R. Easton, *Basic Principles of Imaging Science*. Chester F. Carlson Center for Imaging Science, Rochester Institute of Technology, 2005.

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

[10] R. S. Bird, "Tabulation techniques for recursive programs," *ACM Computing Surveys*, vol. 12, no. 4, pp. 403–417, 1980.

[11] W. Pugh and T. Teitelbaum, "Incremental computation via function caching," in *ACM Symposium on Principles of Programming Languages*, 1989, pp. 315–328.

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

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

[14] F. Khalvati and M. D. Aagaard, "Window memoization: an efficient hardware architecture for high-performance image processing," *Journal of Real-Time Image Processing*, vol. DOI 10.1007/s11554-009-0128-y, 2009.