The International Journal Of Engineering And Science (IJES)
|| Volume || 5 || Issue || 2 || Pages || PP -45-53 || 2016 ||
ISSN (e): 2319 – 1813 ISSN (p): 2319 – 1805

A Hardware-Aware Mean Filtering Algorithm

Abdullah Baz
Computer Engineering Department
College of computer and information systems
Umm Al-Qura University
aobaz01@uqu.edu.sa

ABSTRACT

The mean filter is a multipurpose spatial technique extensively used in image processing and pre-processing applications. Its execution time strongly depends upon the image and kernel size. Therefore, inefficient implementation of the mean filter wastes the time and energy of the computing hardware. According to the literature, researchers in this field rely only on static analysis techniques (such as counting the number of mathematical operations) to determine the performance of any algorithm or to compare the performance of different algorithms. This article investigates the state of the art mean filtering techniques and proved theoretically and experimentally that such analysis techniques are not accurate in measuring the performance of any implementation. Interestingly, this paper proposes a new mean filtering implementation that outperforms the optimum existing technique for large image sizes, although it does more number of mathematical operations.

Index Terms—Mean filtering, performance, algorithm, instruction.

Date of Submission: 15 February 2016 Date of Accepted: 05 March 2016

I. INTRODUCTION

Mean filtering filters the image by replacing the value of each pixel with the mean (average) value of its neighbours that are located within a window (mask or kernel). Therefore, it can be directly used to smooth images, reduce sharp transitions in grey levels, remove small details, and reduce noise & blurring. Moreover, many other advanced image processing techniques such as image segmentation are built upon mean filter. Mean filtering can be thought of as a convolution filter. Often a square kernel is used for mean filtering where small kernel size is used for fine filtering and large kernel size is used for severe filtering [1-5].

The mean filter can be defined mathematically as follows without any loss of generality. Let the grey levels of all pixels in a source image are stored in a $2D$ array called I and the location of any pixel is determined by its Cartesian coordinates x & y, where the origin of the coordinate system is the topmost leftmost pixel of the image [1-5]. Then, the grey level value of any pixel can be determined by $I(x, y)$ and the pixels values of the image filtered by $m \times n$ window is:

$$(x, y) = \frac{1}{m \times n} \sum_{j=x-\frac{m-1}{2}}^{x+\frac{m-1}{2}} \sum_{k=y-\frac{n-1}{2}}^{y+\frac{n-1}{2}} I(j, k) \qquad (1)$$

, where m and n must be an odd number so that the window is centred at (x, y). If the width and height of the image I is W and H pixels, respectively, then the simplest algorithm for implementing this filter is:

```
Algorithm 1: Basic algorithm
Input: Image $I$ of width $W$ pixels & height $H$ pixels, and the window size ($m$ pixels in rows and $n$ pixels in columns)
Output: Filtered image $O$
$For\ x\ from\ 0\ to\ W-1$
    $For\ y\ from\ 0\ to\ H-1$
        $Sum \leftarrow 0$
        $For\ j\ from\ x-(m-1)/2\ \ to\ x+(m-1)/2$
            $For\ k\ from\ y-(n-1)/2\ \ to\ y+(n-1)/2$
                $Sum \leftarrow Sum + I(j,k)$
            $End$
        $End$
        $O(x,y) \leftarrow Sum/(m \times n)$
    $End$
$End$
```

The straightforward inspection of the above equation and algorithm shows that its execution time has a direct relationship with the image size (W & H) and the filter size (m & n). Many researchers [6,7] in the field of DSP algorithms rely only on the number of mathematical operation involved in an algorithm to measure its execution time.

The main contribution of this paper is investigating the existing techniques of execution time estimation of mean filtering algorithms. Based upon the results obtained from this investigation, this article exploits the knowledge about the working mechanism of modern hardware to propose a new fast mean filtering algorithm.

The remainder of this paper is organized in six sections. Second section covers the state of the art mean filtering algorithms and techniques. Third section explains the time cost of conditional instructions, which is used in the optimum existing mean filtering algorithm. Fourth section figured out the bottleneck of the optimum mean filtering algorithm. Then, we propose a new hardware-aware mean filtering algorithm in section five and compare its performance with the optimum algorithm in section six. Finally, we summarize the article and states our future work in section seven.

II. BACKGROUND

The authors of [6] noticed that the basic mean filtering algorithm repeatedly computes the sum of all neighbourhoods of the index pixel every time it moves from one position to the next. This causes a major redundancy because the filter only needs to add the last column of the new kernel and subtract the first column of the previous kernel to the current sum as shown in Fig. 1. Considering this point in the implementation allows the authors to decrease the number of mathematical additions and hence the computation time of the filter by at least 50% when the kernel size is 5×5.

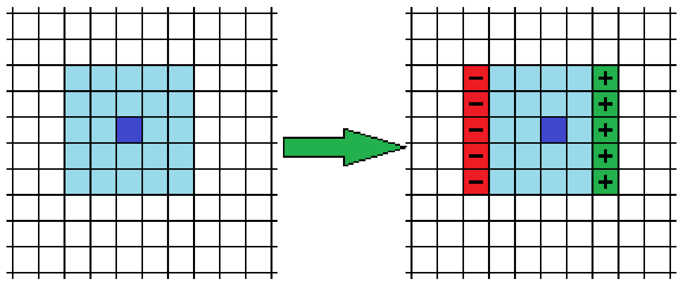


Fig. 1. A fast method of calculating the mean filter by adding the last column of the new kernel and subtracting the first column of the old kernel to the current sum.

The authors of [7] went one step further by completely eliminating the division operation in addition to further decreasing the number of addition operations involved in the filter. They called this method Zero Division Method (ZDM) and they proved that its execution time is faster than all other existing techniques.

Many researchers [6-7] in this field measured the execution time of any algorithm based upon only static parameters such as the number of involved mathematical operations (e.g. additions, divisions) or complexity (e.g. computational, time). However, the execution time of any algorithm is the summation of the execution time of each instruction. Therefore, it is not accurate to limit the execution time of the whole algorithm to some instructions only and ignoring the rest. This can be considered as an approximation only if the execution time of other instructions can be mathematically neglected with respect to the total execution time [8]. In order to judge whether the execution time of an instruction can be ignored or not, we need to understand how the execution of that instruction is conducted inside the hardware. This is a must because the execution techniques of some instructions are more complicated than they look. Moreover, the execution time of some instructions is completely dynamic and cannot be captured by any static analysis technique [8-10]. In order to demonstrate this fact, we will investigate the ZDM algorithm to show that one of the ignored instruction (conditional statement) take a significant amount of the total execution time, which cannot be ignored in any calculation. Firstly, we start by explaining how modern processors execute conditional instructions.

III. COST OF CONDITIONAL INSTRUCTIONS

Modern computing units are multicores, which work based on pipeline principle. Thus, each instruction is divided into several phases (e.g. fetch, decode, execute) and each phase is executed in a different pipeline stage. This technique will keep all pipeline stages busy most of the time, which optimizes the computation performance [8-10].

Conditional instructions force the computing unit to execute different instructions depending upon whether predefined conditions evaluate to true or false. This causes a difficulty for the pipeline working mechanism [8-9]. For instance, the pipeline stage that is responsible of fetching the instructions will not be able to fetch the instruction next to the conditional instruction until the execution stage evaluates the condition. This will halt some pipeline stages and decreases the overall performance of the computing unit. Pipelined processors overcome this challenge via what is called branch predictor techniques, which attempt to predict whether the condition is true or false. Based upon the prediction, the fetching stage will fetch the instruction that it thinks (not sure) is the next one [9-10]. If the prediction was true, the processor will carry on its execution. However, if the prediction was wrong, the pipeline will discard the partially executed instructions, then fetch, decode, and execute the correct instructions. This situation is called Misprediction, which wastes the resources of the processing unit. Despite that possible wastage, it was proven that branch predictor optimizes the overall performance of multicore processing units [8-10]. The prediction accuracy strongly depends upon whether the involved condition has a predictable pattern or not. Simple patterns (e.g. always TRUE, always FALSE, or TRUE-FALSE-TRUE-FALSE...) is easy to predict by most existing branch prediction techniques. However, complex pattern (e.g. TRUE-TRUE-TRUE-FALSE-FALSE-TRUE-TRUE-TRUE-FALSE-FALSE…) is difficult to predict and the execution time of its conditional instruction could be several times slower than simple pattern conditional instructions. Therefore, in order to accurately estimate or compare the execution time of any algorithm that has a condition instruction, impact of branch Mispredictions must be taken into account.

IV. BOTTLENECK OF ZDM ALGORITHM

ZDM algorithm has only one condition instruction, which is crucial to ensure the circular nature of the array. The involved condition compare a counter with the kernel size, therefore, this instruction strongly depends upon the size of the utilized kernel. Table 1 lists the pattern for different kernel sizes (n), assuming square kernel.

TABLE I
THE CONDITION PATTERN OF THE CONDITIONAL INSTRUCTION IN ZDM ALGORITHM.

n	Pattern	Predictability
3	FFFT-FFFT…	↓
5	FFFFFT-FFFFFT…	
7	FFFFFFFT-FFFFFFFT…	
9	FFFFFFFFFT-FFFFFFFFFT…	
11	FFFFFFFFFFFT-FFFFFFFFFFFT…	
13	FFFFFFFFFFFFFT-FFFFFFFFFFFFFT…	
15	FFFFFFFFFFFFFFFT-FFFFFFFFFFFFFFFT…	

The pattern of the condition is difficult to predict and its predictability increases as the kernel size increases. This increases the overall execution time of the algorithm. In order to demonstrate this experimentally, we conducted an experiment to investigate this issue in the algorithm. The aim of the experiment is to figure out how much that condition instruction consumes from the total execution time. The experiment involves a software implementation of ZDM algorithm in C# programming language. Then the implementation is used to filter several images of different sizes utilizing different kernels. During the execution, we measured the processing time of each instruction, which can be defined as how long the execution of each instruction has kept the CPU busy. This performance metric does not only measure the time but also it gives an indication about how much energy the CPU consumes in order to complete the execution of the instruction. Our experiment guarantees that the measured time are not distorted by other processes running in the backend and the time spent while execution is waiting (e.g. in a sleep mode) is not counted. We also count whether the execution utilizes single or several cores of the processing unit.

The results of the experiments are depicted in Fig. 2, which plots the normalized execution time of only the condition instruction with respect to the total execution time versus the kernel size.

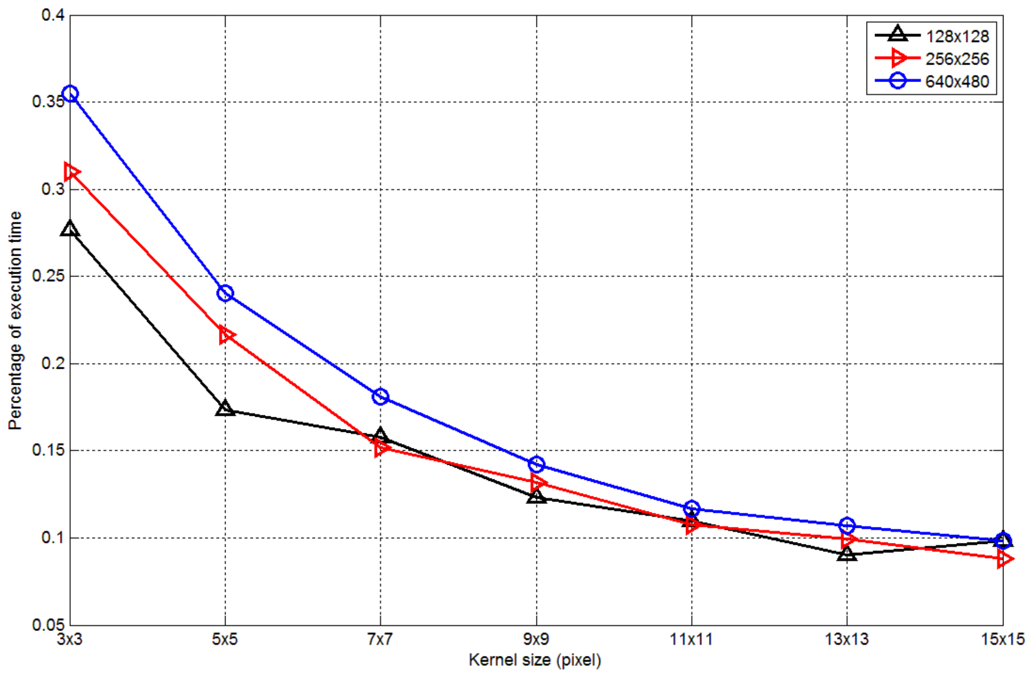


Fig. 2. Normalized execution time of condition instruction in the ZDM algorithm.

The data demonstrates that the processor spends between 10% and 35% of total execution time only in executing the condition instruction. This is mainly due to the unpredictable patterns involved in the condition of this instruction. Interestingly, as the kernel size increases the normalized execution time of the instruction decreases, which occurs due to the increasing in the predictability as explained in TABLE I. Despite this high percentage of the total execution time, many researchers [6-7] ignored such instructions and do not count them during the evaluation and comparison of algorithms.

Therefore, it is worth highlighting that in order to implement an efficient algorithm or to estimate its execution time, we need to understand how the targeted hardware executes that algorithm.

V. A HARDWARE-AWARE MEAN FILTERING TECHNIQUE

The previous results guided us to implement the mean filter using a new hardware-aware algorithm via exploiting the required knowledge about how modern processors function.

As mentioned earlier, repeating the sum operation of all neighbourhood pixels from scratch every time the index pixel moves from one pixel to the next results in a major redundancy. This type of redundancy was addressed in [6-7] for the case when the index pixel moves horizontally. However, the redundancy occurred when the index pixel moves vertically and horizontally has not yet been addressed.

Here, we propose the following working mechanism to address this redundancy. After manipulating the boundary rows and columns, we start from the topmost leftmost pixel in the source image and calculate the sum of all neighbourhood pixels in the window. Then we repeat the following steps until the index pixel reaches the bottommost row of the source image.

1. Move the index pixel one position towards the bottom of the image. Then subtract from the sum the values of pixels in the topmost row of the old kernel and add to the sum the values of pixels in the bottommost row of the new kernel. Finally, average the sum and assign its value to the middle pixel.
2. Move the index pixel one position towards the right of the image. Then subtract from the sum the values of pixels in the leftmost row of the old kernel and add to the sum the values of pixels in the rightmost row of the new kernel. Finally, average the sum and assign its value to the middle pixel. This step is repeated until the index pixel reaches the rightmost pixel in the current row.

3. Move the index pixel one position towards the bottom of the image. Then subtract from the sum the values of pixels in the topmost row of the old kernel and add to the sum the values of pixels in the bottommost row of the new kernel. Finally, average the sum and assign its value to the middle pixel.
4. Move the index pixel one position towards the left of the image. Then subtract from the sum the values of pixels in the rightmost row of the old kernel and add to the sum the values of pixels in the leftmost row of the new kernel. Finally, average the sum and assign its value to the middle pixel. This step is repeated until the index pixel reaches the leftmost pixel in the current row.

Starting from this working mechanism, we developed an algorithm that only utilizes low cost instructions such as assignments, mathematical operations, and iteration instructions. This results in a hardware-aware algorithm that is capable of addressing the mentioned redundancy as listed blow in algorithm 2.

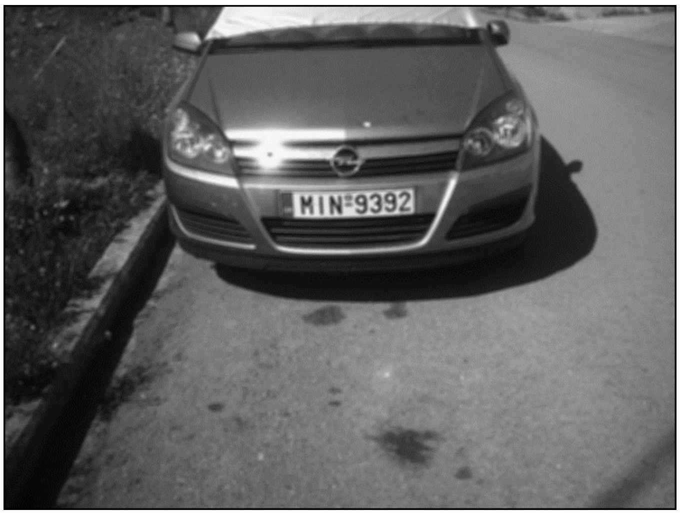
Algorithm 2: Hardware-aware algorithm

Input: Image I of width W pixels & height H pixels, and the size of the square window (n pixel)
Output: Filtered image O

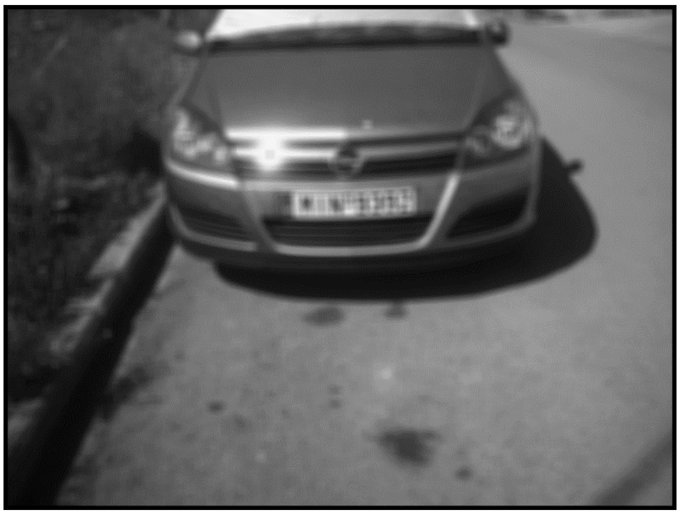
```
$sum \leftarrow all\ pixels\ in\ the\ kernel$
$for\ y\ from\ 0\ to\ H-1$
        $x \leftarrow 0$
        $sum \leftarrow sum - topmost\ row\ of\ the\ old\ kernel$
        $sum \leftarrow sum + bottommost\ row\ of\ the\ new\ kernel$
        $O(x+(n-1)/2, y+(n-1)/2) = sum/(n \times n)$
        $for\ x\ from\ 1\ to\ W-1$
                $sum \leftarrow sum - leftmost\ column\ of\ the\ old\ kernel$
                $sum \leftarrow sum + rightmost\ column\ of\ the\ new\ kernel$
                $O(x+(n-1)/2, y+(n-1)/2) = sum/(n \times n)$
        $end$
        $y \leftarrow y+1$
        $sum \leftarrow sum - topmost\ row\ of\ the\ old\ kernel$
        $sum \leftarrow sum + bottommost\ row\ of\ the\ new\ kernel$
        $O(x+(n-1)/2, y+(n-1)/2) = sum/(n \times n)$
        $for\ x\ from\ w-1\ to\ 1$
                $sum \leftarrow sum - rightmost\ column\ of\ the\ old\ kernel$
                $sum \leftarrow sum + leftmost\ column\ of\ the\ new\ kernel$
                $O(x+(n-1)/2, y+(n-1)/2) = sum/(n \times n)$
        $end$
$end$
```

We implemented the above algorithm in C# programming language and tested it using sample images. Fig. 3 and 4 shows some outputs of the implemented algorithm.

(a)

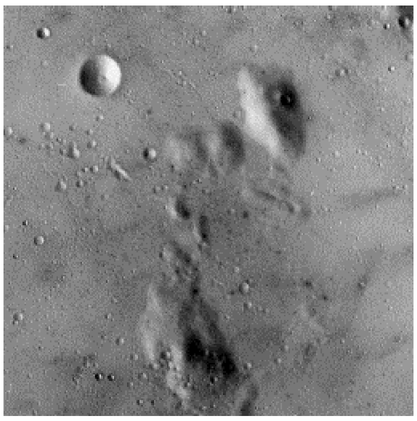


(b)

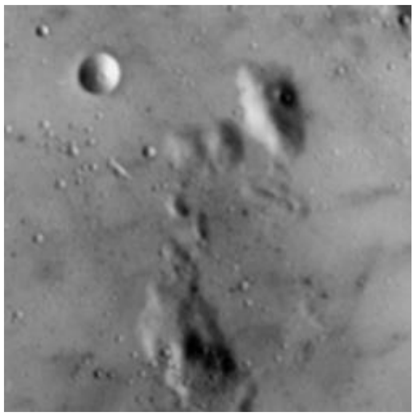


(c)

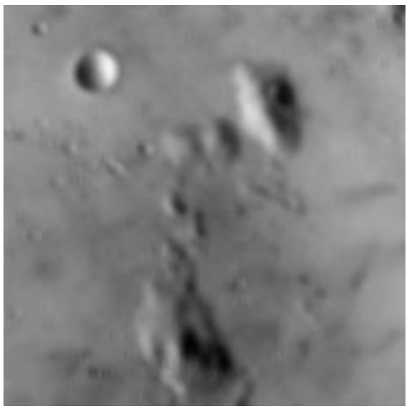
Fig. 3. (a) A test sample obtained from [11] used to test our algorithm, (b) the output for 5×5 mean filter, and (c) the output for 11×11 mean filter.



(a)



(b)



(c)

Fig. 4. (a) A test sample obtained from [12] used to test our algorithm, (b) the output for 3×3 mean filter, and (c) the output for 7×7 mean filter.

VI. COMPARATIVE EVALUATION

In order to demonstrate the difference between hardware-aware and unaware algorithms, this section compares the execution time of our proposed algorithm and ZDM algorithm. Both algorithms were implemented in C# programming language and then all implementations are compiled and run on the same machine under the same operating conditions using the same testing samples. The samples are images of different sizes, which include 128×128, 256×256, 512×512, 640×480, 800×600, 1024×768, 1280×800, 1792×1312, 2304×1728 & 2048×2048, where each image is filtered using different kernels including 3×3, 5×5, 7×7, 9×9, 11×11, 13×13 & 15×15. The execution time for each algorithm is calculated by averaging the execution time of 5000 runs. The results of this experiment are shown in Fig. 5, which plots the normalized execution time versus the image size for different kernel sizes. The normalized execution time is the execution time of the ZDM algorithm divided by the execution time of our proposed algorithm. Consequently, values above 1 mean that our algorithm is faster than ZDM and values below 1 mean that ZDM is faster than our algorithm.

Based on the plotted data, our algorithm is 25% slower than ZDM for small image sizes. However, as the image dimension increases beyond 1000 pixels the execution time of the ZDM starts to increase due to the effect of condition instruction. As an upper limit, the execution time of ZDM is more than twice (205%) the execution time of our algorithm. The effects of the kernel size on the performance, which was described earlier in Table I, appears in plot since the performance of ZDM gets worse as the kernel size decreases.

Furthermore, our experiment involved counting the number of additions in both algorithms. Some of the data is listed in Table II below. Interestingly, the number of addition operations involved in our proposed algorithm is more than that in ZDM algorithm. Our algorithm has between 16% and 75% more addition operations than ZDM for all image and kernel sizes mentioned above. This supports the theory mentioned above: the number of mathematical operations is not an accurate tool to judge the algorithm performance.

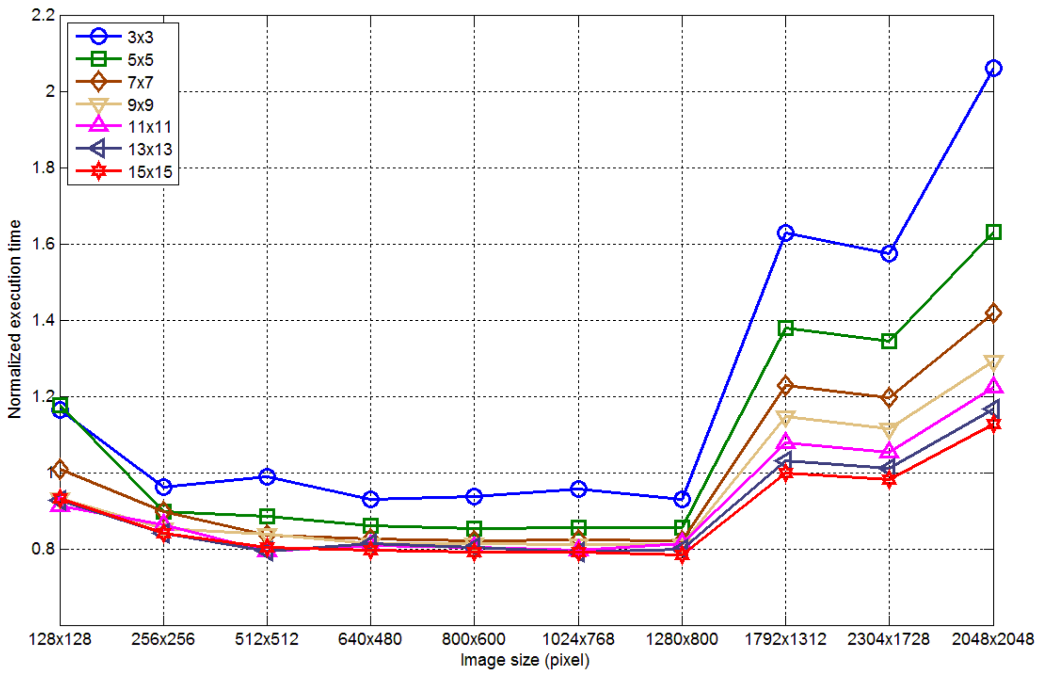


Fig. 5. Normalized execution time of ZDM with respect to our algorithm for different image and kernel sizes.

TABLE II
THE NUMBER OF ADDITIONS INVOLVED IN ZDM AND OUR PROPOSED ALGORITHM.

Image width (pixel)	Image height (pixel)	Kernel size (pixel)	Number of additions in ZDM	Number of additions in our algorithm	Normalized number of additions (Our/ZDM)
128	128	3	81920	95256	1.16
128	128	15	278528	389880	1.39
256	256	3	327680	387096	1.18
256	256	15	1114112	1756920	1.57
512	512	3	1310720	1560600	1.19
512	512	15	4456448	7440120	1.66
640	480	3	1536000	1829784	1.19
640	480	15	5222400	8751480	1.67
800	600	3	2400000	2863224	1.19
800	600	15	8160000	13817880	1.69
1024	768	3	3932160	4697112	1.19
1024	768	15	13369344	22846200	1.70
1280	800	3	5120000	6119064	1.19
1280	800	15	17408000	29852280	1.71
1792	1312	3	11755520	14069400	1.19
1792	1312	15	39968768	69235320	1.73
2304	1728	3	19906560	23839512	1.19
2304	1728	15	67682304	117751800	1.73
2048	2048	3	20971520	25116696	1.19
2048	2048	15	71303168	124114680	1.74

VII. CONCLUSION AND FUTURE WORK

Due to the importance of the mean filter for most image processing and pre-processing applications, this paper investigated the existing mean filtering algorithms. According to the literature, the optimum one is called Zero Division Method (ZDM). The researchers in this field rely only on the number of mathematical operations involved in any algorithm to measure its performance and to compare it with others. However, we proved (theoretically and experimentally) in this paper that this is an inaccurate method. Furthermore, we figured out the main bottleneck of the ZDM algorithm and we conducted an experiment to support our argument. According to the obtained results, we proposed a new hardware-aware algorithm that has more mathematical operations than ZDM, however, its execution time is half that of ZDM for some image and kernel sizes.

Therefore, we conclude that proposing an efficient algorithm requires a good knowledge about how real hardware function, which is needed to avoid instructions that wastes processors resources. The work proposed in this paper opened a new research dimension for optimizing the performance of existing DSP algorithms via exploiting the required knowledge about how the targeted hardware functions. Our future work will concentrate in investigating other important processing techniques to improve their performance.

REFERENCES

[1] R. Gonzalez, R. Woods, S. Eddins, Digital Image Processing Using MATLAB, Pearson Education, 2004.

[2] R. Gonzalez, R. Woods, Digital Image Processing, second ed., Pearson Education, 2002.

[3] W Pratt, Digital Image Processing, third ed.,Wiley, NewYork, 2001.

[4] A. Jain, Fundamentals of Digital Image Processing, Prentice-Hall, 1989.

[5] A. Rosenfeld, A. Kak, Digital Picture Processing, Academic Press, New York, 1982.

[6] J. PAN, Y. TANG, B. PAN, the Algorithm of Fast Mean Filtering, the 2007 International Conference on Wavelet Analysis and Pattern Recognition, Beijing, China, 2-4 Nov. 2007.

[7] S. Rakshit, A. Ghosh, B. Uma Shankar, Fast mean filtering technique (FMFT), Pattern Recognition 40 (2007) 890 – 897.

[8] X. Wu, Performance Evaluation, Prediction and Visualization of Parallel Systems, Springer Science & Business Media.

[9] P. Chang, E. Hao, T. Yeh, Y. Patt, Branch Classification: a New Mechanism for Improving Branch Predictor Performance, the 27th Annual International Symposium on Microarchitecture, 1994.

[10] D. Parikh, K. Skadron, Y. Zhang, M. Stan, Power-aware branch prediction: characterization and design, IEEE Transactions on Computers, Vol.53, (2), pp.168-186.

[11] [Online]. Available:http://www.medialab.ntua.gr/research/LPRdatabase.html

[12] [Online]. Available:http://sipi.usc.edu/database/