INVESTIGATING THE MEDIAN FILTER OPERATION ON CPU AND GPU

Iyad Katib
King Abdulaziz University, Jeddah 21577, Saudi Arabia
Email: iakatib@kau.edu.sa

https://doi.org/10.26782/jmcms.2020.08.00052

Abstract

The Median Filter (MF) is one of the problems that need massive computational resources to perform its operation in a moderate time. The MF can be implemented on traditional CPUs and GPUs. Investigating the performance in terms of processing time of the MF on different architectures can provide the researchers with wider vision to optimally select the computational resources that best fit the required time needed to remove salt and pepper noise. This paper shows the impact of different parameters affecting the MF processing time. Resolution of the frame, frame rate per second, and the MF r value are investigated in order to decide both the preferred architecture and algorithm. OpenMP has been deployed on CPUs and CUDA has been deployed on Nvidia GPGPU K20. Experimental results show that histogram approach and K20 using CUDA are the best choice for processing 4K resolution with r > 2 and HD resolution with r > 4. For VGA resolution and r > 6, histogram approach and CPU using OpenMP are the best choice. The paper provides a way to select the architecture-algorithm pair suitable for implementing the MF.

Keywords: CUDA, GPU, Histogram Approach, Median Filter, OpenMP

I. Introduction

The median filter MF is considered one of the important noise removal approaches in image processing field. It is considered a basic step in image pre-processing before implementing more deep and complex approaches. The MF can help in reducing salt and paper noise while preserving the edges. Calculations of the MF can be performed by replacing each pixel with the median of a window surrounding this pixel. This window consists of \( (2r + 1)^2 \) pixels where \( r \) is the number of pixels that surround the pixel of interest. A MF with \( r = 2 \) is shown in Fig. 1. Thus, the pixel \( P_{i,j} \) is replaced by the median of the set defined as \( \{ P_{i-r, j-r},..., P_{i+r, j+r} \} \). Consequently, the item of index \( 2r^2 + 2r \) is the median [II].
Optimization of the MF has been tackled in research several times [III], [IV], [V], [XII], [XIII]. This is due to the fact that the MF is a pillar in all image pre-processing applications. However, another important fact is that the MF is considered one of the computationally intensive operations. Many hardware approaches using FPGA [VIII] and ASIC [XI] have been used to speed up the MF operation. Real-time vision systems always tend to use hardware solutions [VII], [IX].

Using the MF in video streaming results in tremendous amount of sorting operations. For example; applying the MF with $r = 7$ on a video stream of only one second in HD resolution (1280 x 960) with a medium frame rate of 24 fps yields 28,743,264 sorting operations each working on 225 data items. This means that one-minute live streaming will result in more than 1.7 billion sorting operations. 4K (3840 x 2880) frames will lead of course to much higher operations.

The MF operations can be implemented using different approaches. Insertion sort is one of the well-known sorting algorithms that can be used in implementing the MF [I]. Histogram-based approaches can also be used in performing the MF operations as in [X]. This paper shows how to implement the MF using both CPU and GPU using different parallel algorithms. We will work on video streams of three different resolutions; VGA (640 x 480), HD, and 4K. We will also use different frame rates such as 15, 24, and 30 fps.

This paper provides a parallel implementation to the median filter on CPUs and GPGPUs. OpenMP will be used on the CPU while CUDA will be used on Nvidia GPGPU. Furthermore, the paper will provide recommendations related to the implementation of the MF based on a set of parameters such as frame rate, $r$, and resolution to decide both the architecture and algorithm that best fit the case.

The rest of the paper is organized as follows: section 2 discusses different factors that affect the performance (speed) of the MF. Section 3 discusses two well-known algorithms used to implement the MF. The first algorithm is the insertion sort. The second depends on the use of histogram approach. Section 4 presents the numerical results of the implementation. This section also provides explanations to these results. Section 5 presents the conclusion and directions for future work.

II. Factors Affecting the MF Performance

The MF performance is affected by many factors. These factors include the radius $r$, frame resolution, frame rate per second, algorithm used to implement the MF
operations, and the type of architecture on which the MF operations will be performed. In this section, we will study the impact of each parameter.

The Radius \( r \) determines the number of the window pixels that will be involved in sorting operations. Window pixels are \((2r + 1)^2\). As \( r \) increases, the number of pixels increases. Consequently, the individual execution time of the sorting operation increases. Frame resolution affects the number of operations required. VGA, HD, and 4K frame sizes are investigated in this paper. As the resolution increases, the number of operations increases.

Frame rate per second also affects the number of operations. As the frame rate increases, the number of operations increases. We will investigate the effect of different frame rates such as 15, 24, and 30 fps. Algorithm used to implement the MF operations is considered a key factor in affecting the overall performance. The complexity of the algorithm and its suitability to be implemented in parallel determines the time needed to implement individual operations related to each pixel. This time is the basic factor in deciding the total implementation time. The ability of the architecture to implement concurrent operations (batch) decides the total required time. The number of batches decides the total time required for the MF to perform its function. As the number of batches increases, the total time increases.

### III. The MF Operations

In this section, we will describe two algorithms used to perform the MF operation; the insertion sort and histogram-based approach. Insertion sort can be implemented in parallel for each pixel position since there is no task dependency. Pseudo code of the MF implementation using insertion sort is shown in Error! Reference source not found.

```plaintext
Parallel Median Filter
For each pixel position \( P_{i,j} \) do
  Use Insertion Sort to sort the values of the pixel positions ranging from \( P_{i-r, j-r} \) to \( P_{i+r, j+r} \)
  Index_\( i \) ← 1
  while Index_\( i \) < length(A)
    Index_\( j \) ← Index_\( i \)
    while Index_\( j \) > 0 and A[Index_\( j-1 \)] > A[Index_\( j \)]
      swap A[Index_\( j \)] and A[Index_\( j-1 \)]
      Index_\( j \) ← Index_\( j \) - 1
    end while
    Index_\( i \) ← Index_\( i \) + 1
  end while
  Select the middle one in the sorted list A[2r^2 + 2r]
```

Copyright reserved © J. Mech. Cont.& Math. Sci.

Iyad Katib
Cumulative histogram-based approach [VI] depends mainly on calculating the frequency of occurrence of the colour in the array of pixels surrounding the pixel position. If each pixel holds 8-bit data, there are 256 possible colours for a given pixel. The median is calculated simply by integrating the frequencies of colours till the value of integration reaches the index of the centre of the window (cut-off), which is \(2r^2 + 2r\).

IV. Experimental Results

To study the impact of different factors affecting the MF operation, we conducted a set of experiments on traditional CPUs and GPGPUs. All the experiments on the traditional CPU have been implemented using Intel Composer Studio 2017 with OpenMP-enabled. GPU experiments are implemented using CUDA 8. Traditional CPU has dual Intel x86_64 processors each has 12-core processors running at 2.4 GHz, 96 GB memory, and CentOS (Linux) operating system. GPGPU node is a traditional CPU node equipped with NVIDIA K20 GPU card having a total global memory of 5120 MB, and 2496 CUDA cores. Two different approaches; insertion sort and cumulative histogram are implemented on the two architectures. Different \(r\) values are investigated.

The first set of experiments works on 15 fps with VGA, HD, and 4K resolutions. In general, histogram approach performs better than insertion sort for \(r\) value greater than 4. For \(r = 13\) and VGA resolution, histogram approach provides around 16X speedup compared with insertion sort when implementing both approaches on CPU using OpenMP and around 5X speedup when implementing both approaches on K20 using CUDA. The speedup factor decreases on CPU and increases on K20 for higher resolutions. For histogram approach, Fig. 2, Fig. 3, and Fig. 4 show that OpenMP implementation performs better than CUDA on VGA resolution while OpenMP and CUDA are almost similar on HD resolution, and CUDA performs better than OpenMP on 4K resolution respectively. A speedup factor of 2 is achieved when using histogram approach on K20 with respect to the use of CPU for \(r = 13\).

![Fig. 2: The MF Implementation on VGA frames (15fps)](image-url)
The second set of experiments works on 24 fps with VGA, HD, and 4K resolutions. For histogram approach, Fig. 5 and Fig. 6 show that OpenMP and CUDA are almost similar on VGA and HD resolutions. Fig. 7 show that CUDA-based performs better than OpenMP-based implementation using histogram approach on 4K resolution. Again, a speedup factor of 2 is achieved when using histogram approach on K20 with respect to the use of CPU for $r = 13$. 

Fig. 3: The MF Implementation on HD frames (15fps)

Fig. 4: The MF Implementation on 4K frames (15fps)

Fig. 5: The MF Implementation on VGA frames (24fps)
The third set of experiments works on 30 fps with VGA, HD, and 4K resolutions. In principle, the same concept of the similarity between OpenMP and CUDA implementations for histogram approach is clearly shown in Fig. 8 and Fig. 9 for VGA and HD resolutions. Fig. 10 show that CUDA-based performs better than OpenMP-based implementation using histogram approach on 4K resolution. Again, a speedup factor of 2 is achieved when using histogram approach on K20 with respect to the use of CPU for \( r = 13 \).
As we can see from the results, different combinations of architectures and algorithms can be used in different scenarios. Fig. 12 shows the flowchart that describes the selection criteria to select the architecture and algorithm for both 4K and HD resolutions.
Fig. 11: Selection Flowchart of Architecture and Algorithm for 4K and HD frames

V. Conclusion

The MF operation can be optimized by selecting both the architecture and the algorithm that best suits the implementation time requirements. Frame size, frame rate per second, and $r$ parameter of the MF are essential in deciding the architecture and the algorithm. The paper shows that K20 GPU and histogram approach can be used independently when there is a video stream with high frame rate and high resolution such as 4K. Histogram approach provides a steady state behavior irrespective of the resolution, frame rate, and $r$ value. This paper has provided a way to describe the optimum implementation approach for the MF. Future work will involve investigating other types of GPUs, CUDA frameworks, and algorithms. We believe that this work is a step towards an intelligent mechanism to optimally select architectures and algorithms for image processing functions.

Acknowledgement

Computation for the work described in this paper was supported by King Abdulaziz University’s High Performance Computing Center (Aziz Supercomputer) (http://hpc.kau.edu.sa).

Copyright reserved © J. Mech. Cont.& Math. Sci.
Iyad Katib
References

I. C. M. Wu and Y. C. Chiang, “Insertion Sort Circuit Design Applied on the Median Filter,” in 2018 IEEE International Conference on Consumer Electronics-Taiwan, ICCE-TW 2018, 2018.

II. D. S. Richards, “VLSI Median Filters,” IEEE Trans. Acoust., 1990.

III. G. Gupta, “Algorithm for Image Processing Using Improved Median Filter and Comparison of Mean, Median and Improved Median Filter,” Int. J. Soft Comput., 2011.

IV. H. M. Faheem and B. König-Ries, “A New Scheduling Strategy for Solving the Motif Finding Problem on Heterogeneous Architectures,” Int. J. Comput. Appl., 2014.

V. K. Verma, B. Kumar Singh, and A. S. Thokec, “An enhancement in adaptive median filter for edge preservation,” in Procedia Computer Science, 2015.

VI. L. Hayat, M. Fleury, and A. F. Clark, “Two-dimensional median filter algorithm for parallel reconfigurable computers,” IEE Proc. Vision, Image Signal Process., 1995.

VII. M. Fayez, H. M. Faheem, I. Katib, and N. R. Aljohani, “Real-time image scanning framework using GPGPU – Face detection case study,” in Proceedings of the 2016 International Conference on Image Processing, Computer Vision, and Pattern Recognition, IPCV 2016, 2016.

VIII. M. Vega-Rodríguez and J. Sánchez-Pérez, “An FPGA-based implementation for median filter meeting the real-time requirements of automated visual inspection systems,” Proc. 10th IEEE Mediterr. Conf. Control Autom. (MED ’02), 2002.

IX. N. A. Sabour, H. M. Faheem, and M. E. Khalifa, “Multi-agent based framework for target tracking using a real time vision system,” in 2008 International Conference on Computer Engineering and Systems, ICCES 2008, 2008.

X. O. Green, “Efficient scalable median filtering using histogram-based operations,” IEEE Trans. Image Process., 2018.

XI. R. Medhat, H. M. Faheem, and M. E. Khaleefa, “Efficient parallel architecture of median filter,” in Proceedings of the 9th IASTED International Conference on Parallel and Distributed Computing and Networks, PDCN 2010, 2010.

XII. S. Perreault and P. Hébert, “Median filtering in constant time,” IEEE Trans. Image Process., 2007.

XIII. Y. He, P. Liu, Z. Wang, Z. Hu, and Y. Yang, “Filter pruning via geometric median for deep convolutional neural networks acceleration,” in Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2019.