# Implementing Median Filter on Heterogeneous Architectures

Iyad Katib
King Abdul Aziz University, KSA

## ABSTRACT
The Median Filter (MF) is considered one of the computationally intensive problems in the image processing domain. MF can be implemented on heterogeneous clusters consisting of CPUs, Nvidia GPUs, and Xeon-Phi coprocessors (MIC) architectures. This heterogeneity adds more complexity to the problem and is considered a challenging one. This paper deploys a speed-based scheduling strategy to implement the MF on a heterogeneous cluster. The strategy is used to schedule tasks on heterogeneous architectures based on their speed. Basically, suitable parallel computing paradigms such as OpenMP, and CUDA can be used on individual architectures to perform sample set of tasks. Then, the actual number of tasks will be assigned to each one based on its actual speed. The MF operation can be implemented such that the total run time will be significantly improved in comparison to pure CPU-based implementation. The paper shows that the speedup factor is significantly improved when using CPU, GPU, and Xeon-Phi. The paper then shows how the different cluster structures can process different 4K frame rates per second.

## Keywords
Heterogeneous Architectures, Median Filter, Scheduling of Tasks.

## 1. INTRODUCTION
The median filter is one of the well-known approaches used to perform a high degree of noise reduction in an image. It is normally used before performing more complex algorithms on images such as edge detection. The median filter is considered a non-linear digital filtering technique. It is very useful in reducing salt and paper noise and speckle noise. The major advantage of the median filter is that it preserves the edges. The median filter is calculated by replacing each pixel with the median of a window surrounding this pixel. The radius 'r' of the window is a parameter to the filter. Thus, the pixel Pi,j is replaced by the median of the set defined as {Pi-r, j-r ,…, Pi+r, j+r}. This set is sorted and the item of index 2r2 + 2r is the median [1]. The median filter can operate on a single-color channel. For RGB support, each channel should be processed separately.

Due to the importance of the median filter, many attempts to optimize it has been performed [2], [3], [4], [5]. The median filter is a corner stone of all higher-level image-processing applications, and yet is computationally intensive. It has also been revisited from a targeted hardware approach due to its requirement in real-time image processing and portable cameras which have the need for an efficient ASIC to perform the filter [6]. Real-time vision systems always tend to use hardware solutions [7], [8].

The MF is considered computationally intensive problem specially when used in filtering video streaming. Applying the MF with r =5 on a video stream of only one second in 4K resolution (3840 x 2880) with a frame rate of 30 fps means that it is required to perform 329,763,000 sorting operations. Each operation will work on 121 data items. This means that the MF is approaching one billion sorting operations for each 3 seconds of 4k video stream.

This paper introduces a novel parallel implementation to the median filter on heterogeneous architectures. The implementation will use traditional CPUs, Nvidia GPGPUs, and Xeon-Phi coprocessors. Furthermore, the scheduling strategy developed in [9] will be deployed to assign workloads to different types of architectures based on its speed.

The rest of the paper is organized as follows: section 2 discusses the task dependency analysis of the parallel median filter. Section 3 shows how to use the speed-based scheduling strategy in solving the median filter. Section 4 presents the numerical results of the implementation. Section 5 presents the conclusion and directions for future work.

## 2. TASK DEPENDENCY ANALYSIS
Having an image of row x column pixels, the median filter sequential operation can be described as follows:

1) Start with an initial pixel position $\boldsymbol{P}_{i,\,j}$ and a radius '*r*' value in pixels such that $i = j = r+1$. For example, r = 2 means a window of size $(2r+1) \times (2r+1)$ pixels, i.e., 5 x 5 pixels. The initial window starts from $\boldsymbol{P}_{1,\,1}$ and ends up with $\boldsymbol{P}_{5,\,5}$. In general, for a pixel position $\boldsymbol{P}_{\,i,\,j}$, the window will start from pixel position $\boldsymbol{P}_{i-r,\,j-r}$ and will end up with pixel position $\boldsymbol{P}_{i+r,\,j+r}$.
2) Determine the median of the set consisting of $(2r+1)^2$ elements. This can be achieved by sorting the one-dimensional array of elements and finding the median element which is the element of index $2r^2 + 2r$.
3) Put the value of the median element in pixel position $\boldsymbol{P}_{i,\,j}$.
4) Slide the window horizontally to a new pixel position $\boldsymbol{P}_{i,\,j+1}$ and then repeat steps from 1 to 3 till the pixel position $\boldsymbol{P}_{i,\,column-r}$ which indicates the last possible window in the row.
5) Slide the window vertically to a new row and repeat sliding window technique.
6) The process ends when the median of pixel position $\boldsymbol{P}_{\,row-r,\,column-r}$ is calculated.

Having a look at the operation of the median filter, it is found that the sorting operation can be repeated in parallel which means the possibility of parallel implementation. Since each sorting operation will have a set of data items to work on, then multiple threads can be activated to perform multiple sorting

operations simultaneously. The parallel implementation of the median filter can be as in Fig. 1.

```
Parallel Median Filter
For each pixel position P_{i, j} do
Sort the values of the pixel positions ranging from P_{i-r, j-r} to P_{i+r, j+r}
Select the middle one in the sorted list
Replace the pixel value with the middle one
End
```

**Fig. 1: Parallel Median Filter Pseudo Code**

Consequently, the time required to implement the parallel sorting depends mainly on the sorting algorithm used. The time required to perform the parallel sorting is equivalent to the time needed to perform sorting operation on one window having $(2r+1)^2$ items. In this paper, insertion sort will be used to avoid memory contentions associated with assigning extraneous buffers for the parallel implementation of the sorting operation. Hence, $O(n \log(n))$ is considered the complexity of the sorting algorithm [10]. Of course, different sorting approaches can be used such as merge sort, quick sort, etc. [11], [12].

## 3. USING SCHEDULING STRATEGY TO SOLVE THE MF

According to the speed-based scheduling strategy described in [9], the main program ***MedianFilterTaskScheduler*** divides the problem space into smaller subspaces called chunks. Each chunk consists of a set of tasks. The chunk will run on the corresponding architecture. If the architecture is not able to process at least one chunk in a time less than the total run time of the fastest architecture, then this architecture will be excluded. Fig. 2 shows the pseudo code of the task scheduler used to schedule the MF operations on different architectures. Having the total problem space and the speed of each architecture in processing a single chunk, the scheduling strategy will assign the chunks that will be provided to each architecture. Problem space here is the number of sorting operations required. This depends on many factors such as the video stream time, frames per second, and the resolution of the image. Workload distribution will be assigned to each architecture with a start index and end index of the sorting operation.

1. ***PROGRAM MedianFilterTaskScheduler***
2. *Input : S[1, , Frames] ; input the frames*
3. *Input : r*
4. *Input : row, column ; resolution of the image*
5. ***BEGIN***
6. $t[t_1, \dots, t_p] \leftarrow$ *load single chunk execution time for architectures*
7. $t_{min} \leftarrow MIN_{i=1}^{p}(t_i)$ ; find the smallest run time
8. ***FOR*** $i = 1$ ***to*** $p$
9. ***IF*** $t_i \leq t_{min}$ ***THEN***
10. $R_i \leftarrow Problem\_Space / t_i$ ; *find the weight of each architecture*
11. ***ELSE***
12. $R_i \leftarrow 0$ ; *this architecture is very slow and will be ignored*
13. ***END***
14. ***END***
15. $R_{total} \leftarrow R_1 + R_2 + \cdots + R_p$ ; *sum the weights*
16. $R_u \leftarrow Problem\_Space / R_{total}$ ; *find the tasks assigned to each weight unit*
17. $offset = 0$
18. $start_i = 0$
19. ***FOR*** $i = 1$ ***to*** $p$
20. $C_i = R_i * R_u$ ; *sorting operations (tasks) assigned to architecture*
21. $start_i = start_i + offset$ ; *determine the start index of tasks*
22. $end_i = start_i + C_i - 1$ ; *determine the end index of tasks*
23. $offset = C_i$
24. $Median \leftarrow$ ***SPAWN*** $Algorithm_i(r, row, column, C_i, start_i, end_i)$
25. ***END***
26. ***return*** $Filtered\ Frames$
27. ***END***

**Fig. 2: Scheduling routine pseudo code to assign workloads to architectures**

## 4. EXPERIMENTAL RESULTS

In this section the results of implementing the MF on different architectures will be shown. Currently, an HPC cluster has 492 CPU nodes (11,808 cores), 2 GPGPU nodes, and 2 MIC nodes exist on "Aziz" cluster at King Abdulaziz University, KSA. All the experiments on the traditional CPU and Xeon-Phi systems have been implemented using Intel Composer Studio 2017 with OpenMP-enabled. GPU experiments are implemented using CUDA 9. 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. Xeon-Phi node has a total of 60 active cores and the processor is running at 1GHz. GPGPU node is equipped with NVidia K20 GPU card having a total global memory of 5120 MB, and 2496 CUDA cores.

Applying the MF on (3840 x 2880) 4K image will be investigated. Also, the speed of the MF with r = 5 will be checked. Video images with different frame rates will be investigated such as 15, 24, and 30 fps. Insertion sort will be used as a basic sorting algorithm. Parallel instances of the insertion sort will be concurrently activated. Chunk size will depend on the number of frames per second. Table 1 shows the speed of different architectures in processing one second of video stream with different frame rates. It is found that CPU can process 30 4K frames in 167 seconds which means that it is impossible to use it alone to process real-time video streaming. The ratio for each architecture will be as shown in Table 2. This ratio will be used to assign different workloads to each architecture.

For 30 fps video, The CPU will take the responsibility of processing around 32.2% of the workload while the GPU will take the responsibility of 42.8% and the MIC will take the responsibility of processing 25.0% of the workload. The MF can filter a one-minute video streaming of a 4K image, 30 fps (329,763,000 x 60 sorting operations) in 2.79 hours when using one traditional CPU. Implementing the MF on CPU, GPU, and MIC results in 3156 sec. for the same video, which means a speedup factor of more than 3X is achieved when compared to pure CPU implementation. Chunk size here is equivalent to the number of sorting operations needed to process one second video streaming. Chunks assigned to each architecture based on its speed and the speedup factor S are listed in Table 3.

In order to reduce the total time required to process video streaming, the chunk size should be reduced. The chunk size can be selected to be a specific number of sorting operations. Processing power should be able to process all the sorting operations of one second video stream in only one second. For example; 329,763,000 sorting operations (mentioned earlier in this paper) should be performed in one second to support real-time video streaming applications. Table 4 shows the maximum number of sorting operations that can be implemented in one second by each architecture. Consequently, a chunk size 6,107,955 sorting operations can be implemented in one second on a cluster comprising of one CPU, one GPU, and one MIC. Table 5 shows the heterogeneous cluster structure having equal number of CPUs, GPUs, and MICs required to apply MF on real-time video streaming for 4K image with different frame rates.

**Table 1: MF Run Time on Different Architectures (1 Sec. Video)**

| fps | r | $t_{CPU}$ (Sec.) | $t_{GPU}$ (Sec.) | $t_{MIC}$ (Sec.) |
|---|---|---|---|---|
| 30 | 5 | 167.577 | 126.278 | 216.2 |
| 24 | 5 | 135.369 | 100.161 | 173.236 |
| 15 | 5 | 83.7992 | 63.8055 | 109.542 |

**Table 2: Percentage of the Architectures**

| fps | # of Sorting Operations per second | CPU Ratio | GPU Ratio | MIC Ratio |
|---|---|---|---|---|
| 30 | 329,763,000 | 32.2% | 42.8% | 25.0% |
| 24 | 263,810,400 | 31.9% | 43.1% | 25.0% |
| 15 | 164,881,500 | 32.5% | 42.6% | 24.9% |

**Table 3: Speedup factor for One Min Video Stream (chunk size = No. of sorting operations in one second)**

| fps | CPU Time | Chunks Assigned | | | CPU+ GPU+ MIC | S |
|---|---|---|---|---|---|---|
| | | CPU | GPU | MIC | | |
| 30 | 10,054 | 18 | 25 | 17 | 3,156 | 3.18 |
| 24 | 8,122 | 10 | 13 | 7 | 1,353 | 6 |
| 15 | 5,028 | 10 | 13 | 7 | 838 | 6 |

**Table 4: Max. No. of Sorting Operations per Sec. for CPU, GPU, and MIC**

| Arch. | Max. No. of Sorting Operations per Sec. |
|---|---|
| CPU | 1,948,824 |
| GPU | 2,633,863 |
| MIC | 1,525,268 |

**Table 5: Heterogeneous Cluster Structure with Equivalent No. of Nodes of each Architecture Type**

| fps | Heterogeneous Cluster Structure | |
|---|---|---|
| | Total # of Nodes | # of Nodes for Each Arch. Type (CPU, GPU, and MIC) |
| 30 | 162 | 54 |
| 24 | 132 | 44 |
| 15 | 81 | 27 |

The MF experiment has been conducted on different number of nodes as shown in Table 6. Since "Aziz" has a limited number of GPU and MIC nodes on the cluster, it is obvious that as the number of CPU nodes increases, the number of frames that can be processed in a second increases. As seen, 165 CPU nodes (3960 cores) are needed in addition to 2 Nvidia GPU nodes and 2 Xeon-Phi nodes to process real-time video streaming of 30 4K frames per second. 15 fps can be processed using 81 CPU nodes (1944 cores) in addition to 2 Nvidia GPGPU nodes and 2 Xeon-Phi nodes.

**Table 6: MF Experimental Results Using Different No. of CPU Nodes**

| Cluster Structure | | | Processed fps |
|---|---|---|---|
| CPU | GPU | MIC | |
| 2 | 2 | 2 | 1 |
| 19 | 2 | 2 | 4 |
| 53 | 2 | 2 | 10 |
| 81 | 2 | 2 | 15 |
| 132 | 2 | 2 | 24 |
| 165 | 2 | 2 | 30 |

# 5. CONCLUSION

The MF implementation on heterogeneous architectures can significantly improve the speed of its operation. The time required to perform median filtering depends mainly on the number and type of architectures used to perform the sorting operations. Proper scheduling strategy and suitable parallel computing paradigms are used to perform the MF function. In order to provide real-time MF operation on video streaming, more hardware should be deployed to reduce the processing time of sorting operations. In doing so, the frame should be divided among many architectures such that the time required to process it is reduced. Future work may include the use of different sorting technique that may result in reducing the total time required to perform the MF functionality. The

proposed MF implementation on heterogeneous architectures can be used either in real-time or off-line applications. This paper is considered a step towards a complete system to solve computationally intensive problems of image processing domain on HPC clusters comprising of heterogeneous architectures.

## 6. 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).

## 7. REFERENCES

[1] D. S. Richards, "VLSI Median Filters," IEEE Trans. Acoust., 1990.

[2] S. Perreault and P. Hébert, "Median filtering in constant time," IEEE Trans. Image Process., 2007.

[3] G. Gupta, "Algorithm for Image Processing Using Improved Median Filter and Comparison of Mean, Median and Improved Median Filter," Int. J. Soft Comput., 2011.

[4] K. Verma, B. Kumar Singh, and A. S. Thokec, "An enhancement in adaptive median filter for edge preservation," in Procedia Computer Science, 2015.

[5] 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.

[6] 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.

[7] 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 & Systems, 2008, pp. 355–363.

[8] 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 International Conference on Image Processing, Computer Vision, and Pattern Recognition (IPCV), 2016, p. 147.

[9] 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.

[10] Y. Yang, P. Yu, and Y. Gan, "Experimental study on the five sort algorithms," in 2011 2nd International Conference on Mechanic Automation and Control Engineering, MACE 2011 - Proceedings, 2011.

[11] M. McCool, A. D. Robison, and J. Reinders, "Merge Sort," in Structured Parallel Programming, 2012.

[12] H. M. Mahmoud, "Quick Sort," in Sorting, 2011.