Implementing Median Filter On CPU And MIC Using Histogram Approach

Elmasry, Mohamed Abbas

Abstract: The Median Filter (MF) is one of the image preprocessing approaches that require considerable computational resources to perform its operation in a moderate time. The MF can be implemented on traditional CPUs and Intel Many Integrated Core Architecture MIC such as Xeon-Phi coprocessors. This paper addresses the use of histogram algorithm to solve the MF on both traditional CPUs and MIC. Different r values and frame sizes are investigated. OpenMP has been deployed on CPUs and MIC. Experimental results show that histogram approach performs better than traditional insertion sort approach. It also shows that the use of both CPU and MIC architectures together can lead to much better results when proper scheduling strategy is used to assign the workloads.

Index Terms: Median Filter, CPU, MIC, OpenMP, Histogram, Image Processing, Histogram Approach,

1 Introduction

Computationally intensive problems need massive computing resources. It is impossible to perform sequential processing on a single machine. The widespread deployment of modern architectures such as Intel Xeon CPUs and Intel-Xeon-Phi coprocessors can play an important role in accelerating the computations of such computationally intensive problems. Implementing efficient parallel algorithms to exploit the capabilities of such architectures is a key to achieve considerable speed-up when compared to classical approaches[1]. The median filter MF is considered a common approach that is widely used to reduce salt and pepper and speckle noise in an image. Due to the fat that the MF preserves the edges, it can be performed before edge detection operation to achieve better results. The median filter is considered a non-linear digital filtering technique. 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 $P_{i,j}$ is replaced by the median of the set defined as $\{P_{i-r, j-r}, \ldots, P_{i+r, j+r}\}$. This set is sorted and the item of index $2r^2 + 2r$ is the median [2]. A MF with r = 2 is shown in Fig. **1**.

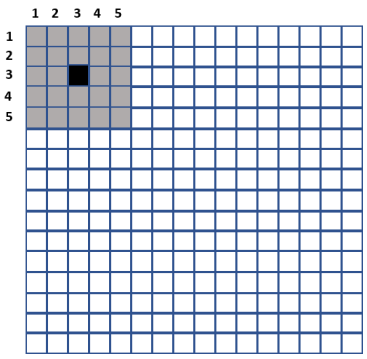

***Fig. 1**, Median Filter with r=2*

Many attempts have been investigated to optimize the MF[3],[4], [5], [6], [7]. Filtering video streams using MF is computationally intensive problem. This is clear when the MF is applied with r =5 on a video stream of only one second in 4K resolution (3840 x 2880) with a frame rate of 30 fps. This will result in about 20 billion sorting operations for a one-minute of 4K video stream. Each sorting operation will work on 121 data items. This paper introduces a histogram-based parallel implementation to the median filter on CPU and MIC. The implementation will use traditional CPUs, and Xeon-Phi coprocessors. The paper will also study the effect of using histogram approach instead of traditional insertion sort algorithm. The rest of the paper is organized as follows: section 2 discusses the MF operation. Section 3 describes the histogram approach. Section 4 presents the numerical results of the implementation. Section 5 presents the conclusion and directions for future work.

2 Median Filter Operation

The MF operation depends mainly on the sorting operation that can be repeated in parallel since there is no dependency. Each sorting operation will have a set of data items and a single thread will be activated for it. It is possible to activate multiple threads simultaneously to perform parallel sorting operations. The parallel implementation of the median filter can be as in Fig. ***2***.

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. 2**, The Parallel Implementation of the MF*

It is obvious that the sorting algorithm is essential in deciding the time required to perform the MF operation. Another important factor is the r size which determines the data items of each sorting operation. 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.

3 Histogram Approach

In the context of image processing, the histogram of an image usually refers to a histogram of the pixel intensity values. This

histogram is a graph shows the number of pixels in the image at each different intensity value presented in that image. If each pixel is represented by an 8-bit data, then there are 256 possible values. It is possible to use a histogram of 256 columns; each represents the frequency of the occurrence of the color having its value. Consequently, it is simple to cumulatively sum the frequencies of colors in an ascending order till the sum value reaches the index $2r^2+2r$. Pseudo code of the histogram approach is shown in Fig. 3.

4 Experimental Results

This section shows the results of implementing the MF on Intel-Xeon processor and Xeon-Phi coprocessor. Intel Composer Studio 2017 with OpenMPis used. Intel-Xeon processors has dual Intel x86_64 processors each has 12-core processors running at 2.4 GHz. It also has 96 GB memory, and Centos operating system. Xeon-Phi node has a total of 60 active cores and the processor is running at 1GHz. It is equipped with 96 GB memory. Experiments are conducted on a single frame having three different resolutions; VGA (640 x 480), HD (1280 x 960), and 4K (3840 x 2880).

```
ALGORITHM: MEDIAN FILTER FOR MIC
MEDIAN(IMAGE, RADIUS, WIDTH, HEIGHT)

INPUT:
   IMAGE: INPUT IMAGE (2D ARRAY)
   WIDTH: WIDTH OF THE INPUT IMAGE
   HEIGHT: HEIGHT OF THE INPUT IMAGE
   RADIUS: RADIUS OF MEDIAN FILTER
OUTPUT:
MEDIAN: OUTPUT IMAGE WITH APPLIED MEDIAN FILTER
 1:      BEGIN
 2:      FOR ROWFROM RADIUSTO HEIGHT-RADIUS
 3:        FOR COLFROM RADIUSTO WIDTH-RADIUS
 4:      HIST[256] = {0}
 5:          FOR RX IN ROW-RADIUS TO ROW+RADIUS
 6:            FOR RY IN COL-RADIUS TO COL+RADIUS
 7:      HIST[IMAGE[RX][RY] += 1
 8:            END FOR
 9:          END FOR
10:          COUNT = 0
11:          FOR IDX FROM 0 TO 255
12:            COUNT += HIST[IDX]
13:            IF COUNT > RADIUS THEN
14:              MEDIAN[ROW][COL] = IDX
15:      EXIT FOR
16:            END IF
17:          END FOR
18:        END FOR
19:      END FOR
20:      END
```

***Fig.** 3, Pseudo Code of the Histogram Approach*

***TABLE 1,** Implementation of Histogram algorithm on CPU and MIC versus the use of Insertion Sort for VGA frame*

Size	r	CPU (Sorting)	CPU (Histogram)	MIC (Histogram)
VGA	2	0.107944	0.131223	0.362253
	3	0.131347	0.13533	0.335598
	4	0.161886	0.156434	0.373549
	5	0.272879	0.175031	0.392383
	6	0.259636	0.206966	0.272821
	7	0.377391	0.182492	0.373906
	8	0.81152	0.192672	0.432415
	9	0.833513	0.166666	0.386287
	10	1.16794	0.212751	0.345792
	11	1.61054	0.180054	0.637261
	12	2.10354	0.196224	0.762281
	13	2.95465	0.191362	0.765866

Different r values ranging from 2 to 13 are also investigated. TABLE **1** shows the results for implementing the MF on a VGA frame using insertion sort versus the use of histogram approach on the same CPU architecture. The histogram algorithm performs better than the insertion sort algorithm when r exceeds 3. A speedup factor of 15.4 is achieved for r=13.Histogram algorithm implementation on MIC performs better than insertion sort when r exceeds 6. A speedup factor of 3.8 is achieved for r=13in this scenario.

***TABLE 2,** Implementation of Histogram algorithm on CPU and MIC versus the use of Insertion Sort for HD frame*

Size	r	CPU (Sorting)	CPU Histogram)	MIC (Histogram)
HD	2	0.400991	0.56954	0.819595
	3	0.**3755**43	0.520846	1.15746
	4	0.527809	0.582261	0.986819

	5	0.703423	0.638658	1.27088
	6	0.960508	0.629441	0.850553
	7	1.61261	0.69143	1.1354
	8	2.11007	0.64441	1.7546
	9	3.44712	0.661576	1.63117
	10	4.50732	0.78813	1.58411
	11	6.23596	0.806266	1.86377
	12	9.07234	0.805719	2.77015
	13	11.3755	0.750359	2.24823

	5	5.51453	5.44277	6.99446
	6	7.6007	5.57322	11.7384
	7	11.2081	6.08744	11.7305
	8	16.7907	5.50116	11.9003
	9	24.4226	6.17103	14.6318
	10	34.6824	5.78344	11.0968
	11	48.1041	6.73073	19.9042
	12	69.9485	6.55753	19.1639
	13	90.3898	6.77607	16.2178

For HD resolution,
TABLE **2** shows that the histogram algorithm performs better than the insertion sort algorithm on CPU when r exceeds 4. A speedup factor of 15.1 is achieved for r=13. Histogram algorithm implementation on MIC performs better than insertion sort when r exceeds 5. A speedup factor of 5 is achieved for r=13in this scenario. For 4K resolution,
TABLE **3**shows that the histogram algorithm performs better than the insertion sort algorithm on CPU when r exceeds 4. A speedup factor of 13.3 is achieved for r=13. Histogram algorithm implementation on MIC performs better than insertion sort when r exceeds 7. A speedup factor of 5.57 is achieved for r=13in this scenario.

TABLE 3, *Implementation of Histogram algorithm on CPU and MIC versus the use of Insertion Sort for 4K frame*

Size	R	CPU (Sorting)	CPU (Histogram)	MIC (Histogram)
4K	2	3.21055	4.39099	8.63384
	3	3.13806	4.40473	7.08617
	4	4.08433	5.12625	10.8289

Another set of experiments has been conducted to study the effect of distributing the workload between the CPU and MIC using speed-based scheduling strategy described in [8]. Based on this scheduling strategy,
TABLE **4** shows the percentage of sorting operations assigned to each architecture for VGA resolution. TABLE 5and TABLE 6 show this percentage for HD and 4K resolutions respectively.

TABLE 4*, The percentage of sorting operations assigned to CPU and MIC for VGA resolution.*

r	# of Sorting Operations	CPU%	MIC%
2	304964	73.40843	26.59157
3	303849	71.26312	28.73688
4	302736	70.4832	29.5168
5	301625	69.15286	30.84714

6	300516	56.86294	43.13706
7	299409	67.20118	32.79882
8	298304	69.17677	30.82323
9	297201	69.85892	30.14108
10	296100	61.90965	38.09035
11	295001	77.97006	22.02994
12	293904	79.52812	20.47188
13	292809	80.00873	19.99127

***TABLE 5**, The percentage of sorting operations assigned to CPU and MIC for HD resolution*

r	# of Sorting Operations	CPU%	MIC%
2	1224324	59.00039	40.99961
3	1222089	68.96597	31.03403
4	1219856	62.89157	37.10843
5	1217625	66.55432	33.44568
6	1215396	57.47003	42.52997
7	1213169	62.15138	37.84862
8	1210944	73.1385	26.8615
9	1208721	71.14482	28.85518
10	1206500	66.77697	33.22303
11	1204281	69.80318	30.19682
12	1202064	77.46788	22.53212
13	1199849	74.97626	25.02374

***TABLE 6**, The percentage of sorting operations assigned to CPU and MIC for 4K resolution*

r	# of Sorting Operations	CPU%	MIC%
2	11045764	66.28754	33.71246
3	11039049	61.66767	38.33233
4	11032336	67.87088	32.12912
5	11025625	56.23809	43.76191
6	11018916	67.80648	32.19352
7	11012209	65.83533	34.16467
8	11005504	68.38679	31.61321
9	10998801	70.33562	29.66438

10	10992100	65.7384	34.2616
11	10985401	74.72969	25.27031
12	10978704	74.50558	25.49442
13	10972009	70.53097	29.46903

TABLE **7**shows the speedup factor when using both CPU and MIC in implementing the MF with respect to the classical insertion sort on CPU for VGA, HD, and 4K resolutions.

Fig.* 4 *shows *that using both CPU and MIC led to better speedup factor rather than using the CPU only.*

TABLE 7, *Speedup factor when using both CPU and MIC for VGA, HD, and 4K*

r	Speedup (VGA)	Speedup (HD)	Speedup (4K)
2	1.120579286	1.193316231	1.103024242
3	1.361950192	1.045479433	1.155272482
4	1.468224623	1.441340777	1.173917536
5	2.254472824	1.654900808	1.801598436
6	2.206157853	2.655244587	2.011297021
7	3.077307066	3.752583587	2.796650991
8	6.088640585	4.477014748	4.46315843
9	7.158853766	7.323747778	5.626766623
10	8.867283512	8.564338458	9.122287736
11	11.47204576	11.0802555	9.563717896
12	13.47962896	14.53496622	14.31691127
13	19.29802859	20.21983483	18.91305445

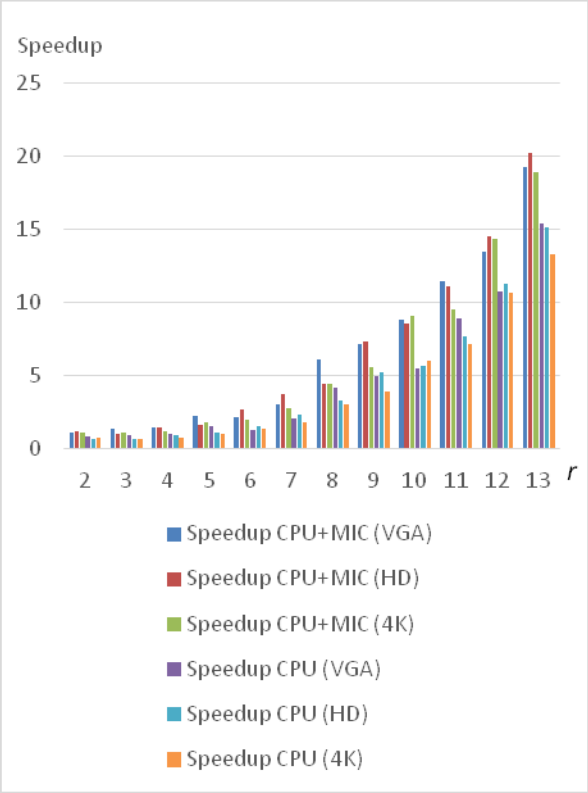

Fig. 4*, Speedup Factor for CPU only and CPU+MIC for Different r and resolutions using Histogram Approach*

5 Conclusion

The paper shows that the use of modern architectures in conjunction with traditional CPU can lead to better performance in solving computationally intensive problems such as the MF. Deploying both CPU and MIC in implementing the MF using speed-based scheduling strategy will certainly lead to better speedup factor in comparison with the use of CPU only. At least more than 20% speedup is achieved when using both Intel Xeon CPU and Intel Xeon-Phi coprocessor. It is also clear that the algorithm used plays an important role in enhancing the results. Histogram approach on CPU has led to better results under certain values of r. However, this paper has proved that using both CPU and MIC together will lead to better performance irrespective of r value. Assigning suitable

workload to each architecture type and using suitable scheduling strategy will eventually lead to better speedup factor. Due to the nature of the MF operations where interactions between tasks are minimal, the use of speed-based scheduling strategy can best fit the distribution of the workload among different architectures based on its individual speed. This paper is considered a step towards a proven system for solving computationally intensive problems on CPUs and MICs.

ACKNOWLEDGMENT

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).

REFERENCES

[1] H. M. Faheem and B. König-Ries, "A Multiagent-based Framework for Solving Computationally Intensive Problems on Heterogeneous Architectures," in Proceedings of the 16th International Conference on Enterprise Information Systems-Volume 1, 2014, pp. 526–533.
[2] D. S. Richards, "VLSI Median Filters," IEEE Trans. Acoust., 1990.
[3] I. Katib, "Implementing Median Filter on Heterogeneous Architectures," Int. J. Comput. Appl., 2020.
[4] S. Perreault and P. Hébert, "Median filtering in constant time," IEEE Trans. Image Process., 2007.
[5] G. Gupta, "Algorithm for Image Processing Using Improved Median Filter and Comparison of Mean, Median and Improved Median Filter," Int. J. Soft Comput., 2011.
[6] K. Verma, B. Kumar Singh, and A. S. Thokec, "An enhancement in adaptive median filter for edge preservation," in Procedia Computer Science, 2015.
[7] 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.
[8] 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.

Elmasry, Mohamed Abbas
Management Information Systems, National Egyptian E-Learning University, Giza, Egypt
malmasry@eelu.edu.eg