P-HGRMS: A Parallel Hypergraph Based Root Mean Square Algorithm for Image Denoising

Tejaswi Agarwal
Undergraduate Student
VIT University, Chennai, India
tejaswi.agarwal2010@vit.ac.in

Saurabh Jha
Undergraduate Student
VIT University, Chennai, India
saurabh.jha2010@vit.ac.in

B. Rajesh Kanna
Advisor
VIT University, Chennai, India
rajeshkanna.b@vit.ac.in

ABSTRACT

This paper presents a parallel Salt and Pepper (SP) noise removal algorithm in a grey level digital image based on the Hypergraph Based Root Mean Square (HGRMS) approach. HGRMS is generic algorithm for identifying noisy pixels in any digital image using a two level hierarchical serial approach. However, for SP noise removal, we reduce this algorithm to a parallel model by introducing a cardinality matrix and an iteration factor, k, which helps us reduce the dependencies in the existing approach. We also observe that the performance of the serial implementation is better on smaller images, but once the threshold is achieved in terms of image resolution, its computational complexity is also increased drastically. We test P-HGRMS using standard images from the Berkeley Segmentation dataset on NVIDIAs Compute Unified Device Architecture (CUDA) for noise identification and attenuation. We also compare the noise removal efficiency of the proposed algorithm using Peak Signal to Noise Ratio (PSNR) to the existing approach. P-HGRMS maintains the noise removal efficiency and outperforms its sequential counterpart by 6 to 20 times (6x - 18x) in computational efficiency. Based on our experiments, we also observe that an increase in SP noise does not impact the performance of the proposed algorithm.

Categories and Subject Descriptors

I.3 [**Computer Graphics**]: Hardware Architecture –*Graphics Processor, Parallel Processing*

Keywords

Parallel Processing, GPGPU, Salt and Pepper Noise

1. INTRODUCTION

Spatial image filtering techniques proposed over the years for applications are generally based on a statistical approach rather than the syntactic consideration. To overcome limitations in the existing noise removal algorithms, Kanna [1] proposed the HGRMS approach which uses Hypergraph based Root Mean Square approach for Salt and Pepper (SP) noise removal. HGRMS is a generic approach, which models the input image using Image Neighborhood hypergraph (INHG) [9] parameters alpha, beta (α, β) to identify the noisy pixels. The identification of noisy hyperedge (NH) involves two sequential operations such that the NH should satisfy the property of isolated hyperedge and its cardinality equal to one. This causes severe performance penalties in computation.

We observed that the performance of the sequential implementation of HGRMS good with image resolutions up to 128 × 128 but once this threshold is achieved, the run time increases considerably. Motivated by the high data parallelism available on the GPU and parallelism in the Hypergraph model, we studied HGRMS in detail, and removed its data dependencies over loop carried dependence.

Our noise reduction algorithm works on the following criterion: binary classification of hyperedge of image (H0 noisy hyperedge and H1 no noisy hyperedge) and filtering the noisy parts. To modify the existing algorithm we include a cardinality matrix, and update the image to remove the noise based on the values in the cardinality matrix. The cardinality matrix is generated based on the alpha, beta (α, β) neighborhood of the pixels in an image. Since the cardinality matrix is independent of the original image, it serves as a platform to exploit parallelism using CUDA threads to update the original image with the Root Mean Square value of its corresponding beta neighborhood pixels. We preserve the image adaptive nature of the existing algorithm by introducing a new parameter, known as iteration factor, *k*.

The rest of the paper is organized as follows: Section 2 describes the proposed algorithm for our work and section 3 describes the results of P-HGRMS as compared to its serial implementation.

2. PROPOSED ALGORITHM

The serial HGRMS algorithm is based on finding the beta neighborhood of each pixel and finding the open neighborhood defined by alpha in beta neighbors based on the Helly Property [8]. This leads to repetitive computations as each pixel value is computed several times during the iterative process. The algorithm is not inherently parallel as it has a high dependence on the previously modified pixel value. In our approach, to parallelize this algorithm, we iterate over the modified pixel values until there is no alpha-beta noise remaining in the image.

This section presents our proposed two-phased parallel approach to noise removal and explains each algorithm in detail.

Algorithm 1 is used to form the cardinality matrix of the image I. The cardinality matrix is formed by finding the cardinality of each pixel, i.e. counting the number of closed neighborhood alpha-beta pixels with respect to that pixel in I. The cardinality number of each pixel is saved in another matrix, which we call the cardinality matrix, C.

Each CUDA thread computes the cardinality value of a pixel in image in parallel with other CUDA threads depending on its threadIdx and blockIdx value This Cardinality matrix, C is generated in parallel.

Algorithm 1: findIsolatedPixels to generate cardinality matrix

```
findIsolatedPixels(*d_image, *d_card, col, row)
1. do in parallel on each thread
2.          c := blockDim.x*blockIdx.x+threadIdx.x
3.          r := blockDim.y*blockIdx.y+threadIdx.y
4.          if r<row and c<col
5.                    for i:=r-beta to r+beta+1
6.                              for j:=c-beta to c+beta+1
7.    if i<row and i>=0 and j<col and j>=0
8.       if  d_image[i*col+j]  <  d_image[r*col+c]+alpha  and
                 d_image[i*col+j]>d_image[r*col+c]-alpha
9.                              d_card[i*col+j]:= d_card + 1
10. end do
11. cudaThreadSynchronize()
```

Algorithm 2: Remove noise for image enhancement

```
removeNoise(*d_image,*d_tempImage, *d_card, col, row)
1.  j=blockDim.x*blockIdx.x+threadIdx.x;
2.  i=blockDim.y*blockIdx.y+threadIdx.y;
3.  if i<row and j<col
4.          sum:=0
5.          pix_count:=0
6.          flag:=0
7.          if d_card[i*col+j]<=2
8           for i1:=i-beta to i1<i+beta+1
9.                    for j1:=j-beta to j1<j+beta+1
10.                             pix_count = pix_count + 1
11.                   if i1<row  and i1>=0 and j1<col and j1>=0
12.  if not (d_image[i1*col+j1] < d_image[i*col+j]+alpha+1 and
                  d_image[i1*col+j1]>d_image[i*col+j]-alpha )
13.     sum = sum + d_image[i1*col+j1] * d_image[i1*col+j1]
14.                           flag = flag + 1
15.         if flag>pix_count-3
16.                   d_image'[i*col+j]=sum/flag
17.  cudaThreadSynchronize()
```

Algorithm 3: Iterate over Algorithm 1 and Algorithm 2 to converge to the optimum solution

```
findOptimumSolution()
1. for K := 1 to 5
2.          findIsolatedPixels(*d_image, *d_card, col, row)
3.    removeNoise(*d_image,*d_tempImage,*d_card, col, row)
4.          d_image = d_tempImage
5. end  for
```

Each CUDA thread works on a cell in the cardinality matrix C in parallel with other CUDA threads.

For each cell in C, if cardinality is found to be less than the threshold value, 3, we find the open neighborhood alpha-beta of the pixel in I, and count the number of pixels which exceed this value. If the count exceeds threshold value, we replace identified noisy pixel by Root Mean Square value of open neighborhood of this pixel in a temporary image matrix, I'. The temporary matrix enables us to remove data race conditions. Once all the CUDA threads are synchronized, we copy temporary image matrix to original image matrix. We repeat the above steps until no further noise remains with respect to alpha and beta. We iteratively call algorithm 1 and algorithm 2 one after another until no further noise remains with respect to alpha and beta. This process is done through algorithm 3 as shown above.

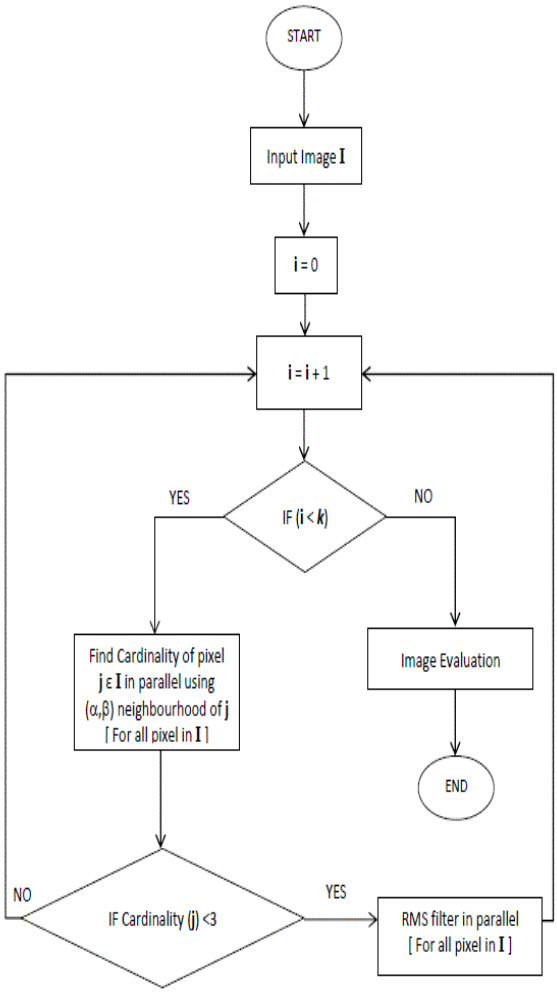

Figure 1: Proposed Algorithm

In our proposed method, to control the number of iterations, we introduce new parameter known as the iteration factor, k. The number of times Algorithm 1 and Algorithm 2 are called in succession depends on this iteration factor, k.

This factor is necessary to preserve the image adaptive nature of HGRMS algorithm. The value of k is dependent of beta parameter as follows. The value of K is dependent on the INHG parameter, β and the HGRMS model. To preserve the noise removal efficiency the number of noisy pixel replacements of our proposed algorithm should be equal to number of noisy pixel replacements of HGRMS model. The computational complexity of HGRMS model is $p^2 \times n \times m$ and that of our proposed system is $k \times (2 \times p \times n \times m)$. We find the value of *k* by equating the two computational complexities.

$k \times (2 \times p \times n \times m) = p^2 \times n \times m$, where m is the height and n width of the image and

$p = (2 \times beta + 1)^2$ where p denotes the chessboard matrix.

Therefore,

$k = \left\lceil \frac{p}{2} \right\rceil$

In our case, beta is equal to 1, and hence the value of p is 9 and *k* is computed as 5. Setting *k* value beyond $\left\lceil \frac{p}{2} \right\rceil$ does not yield any enhancement in the noise reduction capability of P-HGRMS.

If k is set lower than $\frac{p}{2}$, it helps in reduction of number of computations but with greater noise.

The above algorithm is highly parallel and the time complexity is $k \times (18 \times n \times m)$ for a linear implementation and $\frac{k}{c} \times (18 \times n \times m)$ for a parallel implementation where *c* is the number of cores available of the hardware. The next section discusses our implementation results.

3. EXPERIMENTAL RESULTS

We run this algorithm with k =1, 2, 3 and 4 on different sets of salt and pepper images with noise ranging from 5% to 20%. The images are acquired from the Berkeley Segmentation dataset [11].

The existing algorithm HGRMS was implemented and tested on a system with an Intel Core i5 Processor with 4 GB of RAM. For the GPU implementation we used and Amazon AWS cluster, having 2 X NVIDIA Tesla M2050 GPUs having 1690 GB of instance storage and 22 GiB of memory.

Table 1: Execution time in milliseconds to various levels of noise

Noise	1.png (Time in ms)		2.png (Time in ms)		3.png (Time in ms)		4.png (Time in ms)		5.png (Time in ms)	
	CPU	GPU	CPU	GPU	CPU	GPU	CPU	GPU	CPU	GPU
5%	14	2.23	14	2.21	17	1.91	14	1.81	14	1.92
10%	15	2.03	14	1.86	14	1.95	16	1.91	15	1.94
15%	14	2.1	16	1.90	15	2.01	15	1.95	16	2.00
20%	16	2.12	17	1.99	16	2.04	16	2.00	16	2.01

Table 1 shown above gives us the run time of images, with noise levels ranging from 5 % to 20 %. It is evident from the results that a change in noise level does not impact the performance of the proposed algorithm. This was expected as our algorithm is dependent on the size of the image, irrespective of any other factors

Table 2: PSNR values of HGRMS and PHGRMS with different levels of noise.

Noise / Image	**1.png**		**2.png**		**3.png**		**4.png**		**5.png**	
	P HGRMS	HGRMS	PHGRMS	HGRMS	PHGRMS	HGRMS	PHGRMS	HGRMS	PHGRMS	HGRMS
5%	26.94	26.97	29.12	29.15	31.70	31.69	29.76	29.91	28.97	28.96
10%	24.60	24.55	30.78	30.76	27.96	27.91	26.82	26.83	26.11	26.06
15%	22.17	22.11	25.89	25.86	24.30	24.31	23.64	23.63	23.46	23.39
20%	19.67	19.64	22.45	22.40	21.19	21.18	20.66	20.62	20.76	20.74

Restoration performance is quantitatively measured by the peak signal-to-noise ratio (PSNR). Table 2 shown above compares PSNR values of the existing HGRMS algorithm and the proposed P-HGRMS algorithm.

Figure 2: Original Images from the Berkeley Dataset used for PSNR calculation

5 % 10% 15% 20% 20%

Figure 3: Images contaminated with various levels of noise

Figure 4: Restored images using P-HGRMS algorithm

Figure 2 above shows the original five images which we used for testing our algorithm. Figure 3 shows images contaminated with various levels of noise and Figure 3 gives their resultant restored images using P-HGRMS. The PSNR values with various levels of noise show that the noise removal efficiency of HGRMS is maintained in the parallel version proposed in this paper as shown in Table 2.

Finally, we report in Table 3, the speedup of the sequential version of the algorithm running on the CPU and the proposed GPU-parallel version using Tesla architecture. As noted, even with a small image size of 128 X 128 we receive a speed up of 6x. The CUDA implementation is 18x faster with an image resolution of 2048x2048.

Table 3: Speedup of PHGRMS

Method	Image Resolution				
	128x128	256x256	512x512	1024x1024	2048x2048
HGRMS	4.00	11.00	40.00	191.0	1062
PHGRMS	0.64	1.42	4.61	16.73	56.25
Speedup	6.21	7.74	8.67	11.41	18.88

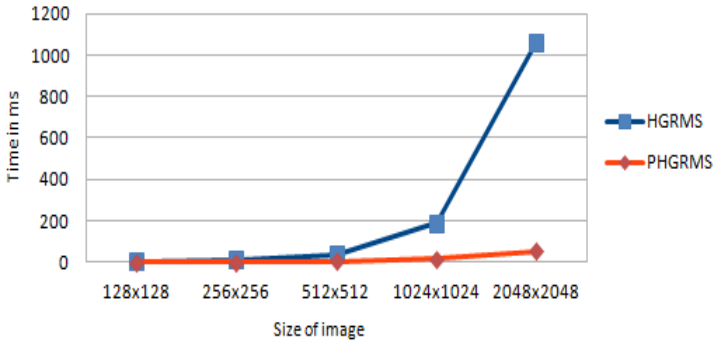

Figure 5: Plot of HGRMS vs PHGRMS

4. CONCLUSION

In this paper, we proposed a two phase parallel algorithm for noise filtering in images with preserving the image details. We test our algorithm on NVIDIA CUDA and results show that the proposed algorithm outperforms its existing Hypergraph based root mean square approach. In this work, we use images contaminated with noise ranging from 5% to 20 % for PSNR for comparisons with HGRMS algorithm. It is worth mentioning that when images are contaminated with high SP noise, the proposed scheme is able to clean images quiet efficiently without loss of image details. The proposed algorithm in this work is highly performance efficient as compared to its serial implementation. Even at higher noise ranges of 20 % the algorithm performed as efficiently as its serial counterpart.

This algorithm represents a very fast and promising approach in denoising salt and pepper noise in digital images. This is because it obviates the needs of two sequential operations involved in identification of noise and introduces parallelism in image modeling and denoising. Future work includes reducing processing time further by testing with the MPI implementation and superior hardware capabilities. We also plan to devise a mechanism where the algorithm would itself decide the iteration factor, *k*, based on the input image, rather than a pre-defined iteration factor.

REFERENCES:

[1] K. Kannan, B. Rajesh Kanna, and C. Aravindan (2010), Root mean square filter for noisy images based on hypergraph model, Image and Vision Computing, 28 (9), 1329-1338, Elsevier, DOI: 10.1016/j.imavis.2010.01.013, 5- Year impact factor- 1.84

[2] Boukerrou and K. L. Kurz. Suppression of salt and pepper noise based on youden designs. Information Science, 110:217–235, 1998.

[3] R. H. Chan, C. W. Ho, and M. Nikolova. Salt and pepper noise removal by median type noise detectors and detail-preserving regularization. IEEE Transactions on Image Processing, 14(10):1479–1485, 2005.

[4] I. Frosio and N. A. Borghese. Human visual system modelling for real time salt and pepper noise removal. Biological and Artificial Intelligence Environments, pages 337–342, 2005.

[5] H. S. L. H. C. Young and K. M. Poon. Modified CPI filter algorithm for removing salt and pepper noise in digital images in visual communications and image processing. In Proceedings of SPIE, pages 1439–1449, 1996.

[6] Rioulo. A spectral algorithm for removing salt and pepper noise from images. In Proceedings of Digital Signal Processing Workshop, Lone, pages 275–278, 1996.

[7] R. Dharmarajan, K. Kannan, "A hypergraph-based algorithm for image restoration from salt and pepper noise", International Journal of Electronics and Communication, AEU, 64 (2010) 1114–1122.

[8] E. Helly. Ueber mengen konvexer koerper mit gemeinschaftlichen punkter, Jahresber. Math. Verein., 32:175–176, 1923.

[9] Bretto, A., Cherifi, H.: Noise detection and cleaning by hypergraph model. In: IEEE Computer Sciences (ed.): International Symposium on Information Technology: Coding and computing. IEEE Computer Sciences (2000) 416–419

[10] Rital, S., Bretto, A., Cherifi, H., Aboutajdine, D.: Application of Adaptive Hypergraph Model to Impulsive Noise Detection. In: Skarbek, W. (ed.) CAIP 2001. LNCS, vol. 2124, pp. 34–42. Springer, Heidelberg (2001)

[11] D. Martin, C. Fowlkes, D. Tal, and J. Malik, "A Database of Human Segmented Natural Images and its Application to Evaluating Segmentation Algorithms and Measuring Ecological Statistics. ICCV 2001.