
Electrical Engineering

New Mexico Institute of Mining and Technology

# GPU Implementation of Adaptive Median Filter

April 21, 2011

EE552

Andrew Ronquillo
Ian Johnson

**Abstract**:
An adaptive median filter is a great tool to have to remove salt and pepper noise. The problem with implementing the adaptive median filter is the amount of time it takes to perform all the necessary calculations on all the layers of the image. One method to help decrease the amount of time to complete all processes is to implement the algorithm on a GPU rather than a CPU. Results acquired show that there was an improvement in processing time, but the improvement was not as much as expected. The results show a 1.5x times increase in speed as opposed to the theoretical 10x-50x increase in speed.

**Introduction**:
The adaptive median filter was implemented using MATLab. In order to have MATLab perform at its peak performance all the code had to be vectorized. Once all the code was vectorized, GPU software was selected. The selected GPU software was Jacket by Accelereyes since the software was easy to implement with MATLab.

Jacket is a GPU software package that allows the user to implement MATLab on the GPU. Jacket supports many MATLab functions and includes the image processing toolbox with pre-written MATLab functions for use. Jacket makes the transition from performing calculations on the GPU and the CPU extremely easy. To perform the processing on the GPU the key 'g' is put in front of most functions, i.e. gsingle() or gdouble(). Once the data is put on the GPU with Jacket, the computations are performed on the GPU until the data is transferred back to the CPU by using regular MATLab functions, i.e. single() or double().

**Background**:
An adaptive median filter algorithm is a great tool to have to remove salt and pepper noise from images due to the simplicity of the algorithm, the robustness of the algorithm, and the conditionals that are inside the algorithm. The conditionals are simple enough to calculate quickly and easily, but each conditional calculation has to be made at each iteration in the algorithm so the overall computing cost can be quite expensive when dealing with larger images and all layers of the image. The GPU could decrease the time of overall filtering because the GPU is amazing at doing parallel computing. The GPU will calculate each loop iteration and every layer of the image simultaneously rather than the CPU which calculates everything sequentially. The algorithm used for the adaptive median filter is shown in Figure 1.

$z_{min}$ = minimum intensity value in $S_{xy}$
$z_{max}$ = maximum intensity value in $S_{xy}$
$z_{med}$ = median of intensity values in $S_{xy}$
$z_{xy}$ = intensity value at coordinates $(x, y)$
$S_{max}$ = maximum allowed size of $S_{xy}$

The adaptive median-filtering algorithm works in two stages, denoted stage $A$ and stage $B$, as follows:

Stage $A$:
$A1 = z_{med} - z_{min}$
$A2 = z_{med} - z_{max}$
If $A1 > 0$ AND $A2 < 0$, go to stage $B$
Else increase the window size
If window size $\leq S_{max}$ repeat stage $A$
Else output $z_{med}$

Stage $B$:
$B1 = z_{xy} - z_{min}$
$B2 = z_{xy} - z_{max}$
If $B1 > 0$ AND $B2 < 0$, output $z_{xy}$
Else output $z_{med}$

**Figure 1: Adaptive Median Filter**

**Results**:
The first test was to compare the final results of the GPU calculations to the CPU calculations to make sure the algorithm was still being performed correctly. The final results show that there is no visual difference between the two results so the algorithm still works as it is supposed to. The filtered image results of one layer of the original image are shown in Figure 2.

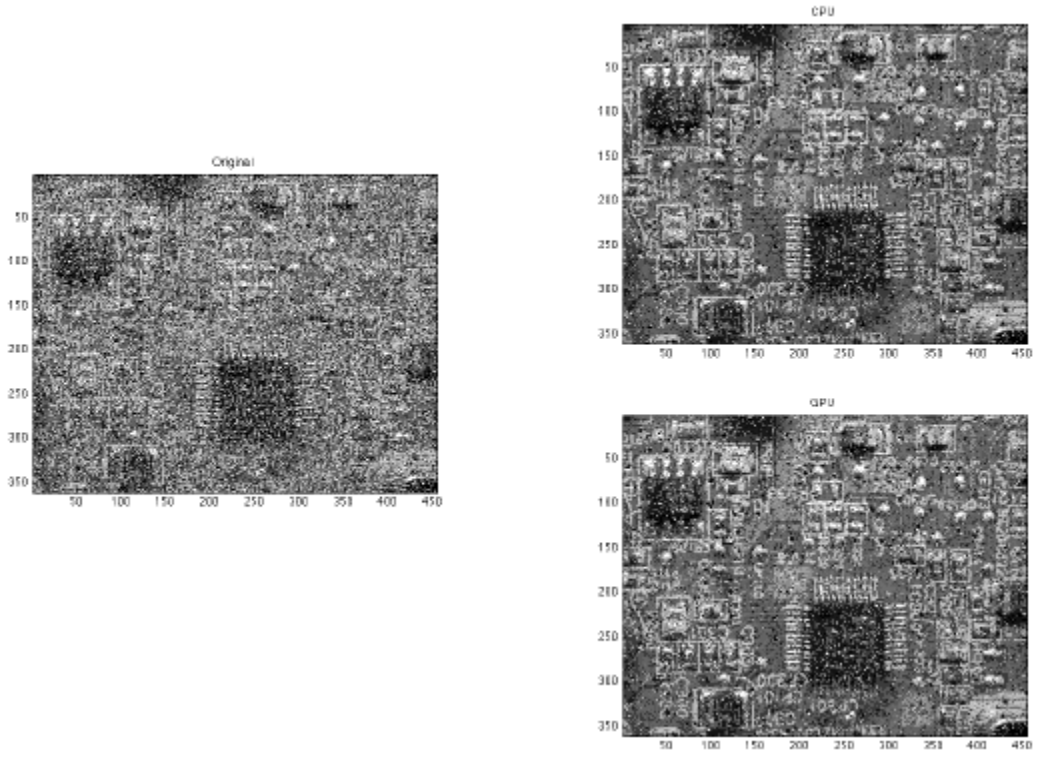


**Figure 2: original (left); CPU (upper right); GPU (lower right)**

The GPU used in this experiment was on a laptop so it was not a dedicated GPU and was only single-precision. In order to get better results with the theoretical 10x-50x results, a dedicated double-precision GPU will have to be used. The algorithm will have to be completely vectorized in MATLab as well for highest potential.

To get the full spectrum on the advantages and disadvantages of the GPU vs. CPU, several tests were performed. The tests were performed with one layer of the image, and then with all three layers of the image. At first the tests were proven inconclusive as shown in Figure 3. The inconclusive results were from 50 iterations of the entire algorithm and were very confusing at first.

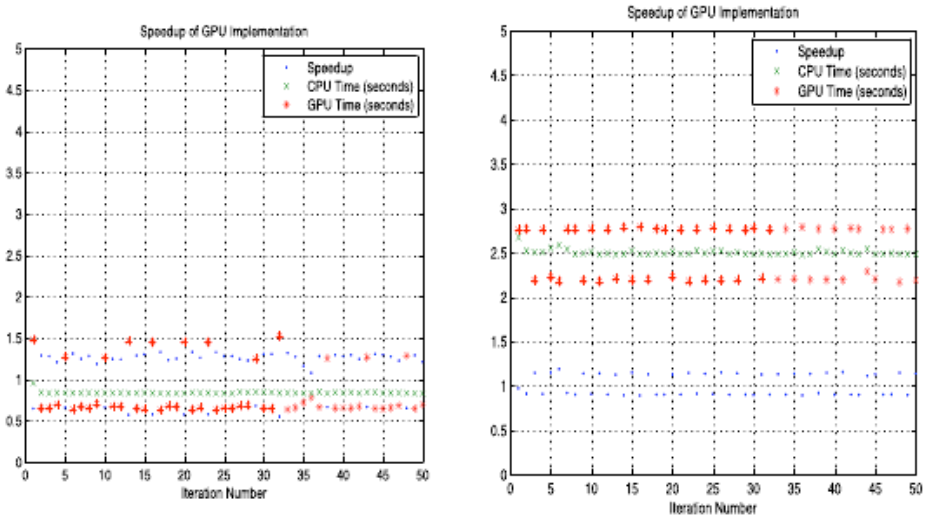


**Figure 3: Inconclusive Results**

After some slight changes in the algorithm, some different functions were used with jacket, and more testing, conclusive results were finally recorded and are shown in Figure 4. Figure 4 shows results from 500 iteration tests.

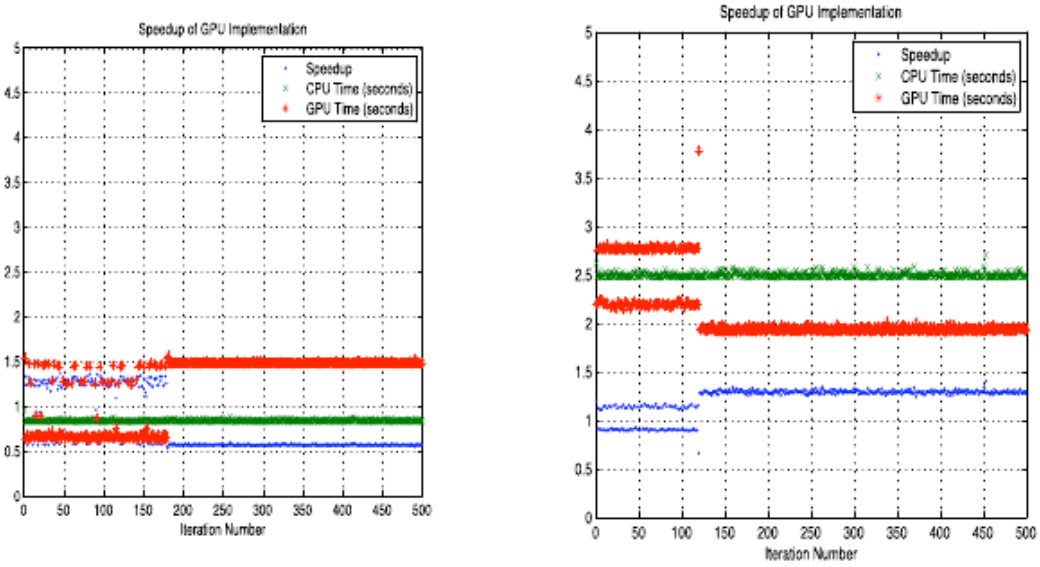


**Figure 4: Total Time of Calculations**

**Data Analysis**:
The results presented were acquired after many different trials and discoveries. In order for the GPU to become truly successful in dominating the CPU, all three layers of the image had to be processed. One reason for these results is due to all the constant data transition from GPU to CPU which takes up a lot of time. The time differences shown in the results are when the laptop used went to sleep therefore allowing the GPU to be completely dedicated to calculations. The difference between the 1-D (single color plane) and 3-D (three color plane) analysis completely makes sense. The double line of the GPU time is the result when the laptop is not asleep and still having to process information to send to the monitor, etc.

In the 1-D analysis, the GPU is generally slower due to the constant switching of CPU calculations and GPU calculations. The reason for the constant processing switching is due to the Jacket unable to support break in loops, nested for loops, and other functions used in MATLab. The reason for the lack of support is due to the GPU processing all the for loop iterations simultaneously and the conditionals of the adaptive median filter rely on other calculations of the for loop.

In the 3-D analysis the GPU starts to show its teeth and begins to dominate the CPU. The CPU time is exactly three times as long to complete one iteration of the algorithm and the GPU shows less than a 3x increase allowing the time decrease to increase immensely. The GPU is able to process all three layers of image at once decreasing the time by at least one third and the for loops are then calculated in parallel as well, decreasing the processing time even more.

**Conclusion**:
In conclusion, Jacket is great software to use for GPU processing and has much potential. The reason why this experiment did not produce the results claimed by Accelereyes (10 to 20 times speedup) is because a GPU that was used was not a dedicated device for processing data. It was the general GPU for the entire computer. Also, the way the filter was implemented required sending data back to the CPU for computation before continuing on the GPU. This data transferring cost a lot of process time on the GPU, if we could implement all the functions on the GPU, eliminating the data transfer problem, the process would be much faster. Overall, the GPU is faster at producing similar results to the CPU due to its ability to process several algorithms simultaneously instead of the sequentially.

**References**:

Gonzalez, Rafael C., and Richard E. Woods. *Digital Image Processing*. Upper Saddle River, NJ: Prentice Hall, 2008. Print.

Gonzalez, Rafael C., Richard E. Woods, and Steven L. Eddins. *Digital Image Processing Using MATLAB*. Upper Saddle River, NJ: Pearson Prentice Hall, 2004. Print.

*Jacket Documentation - Fast GPU Software for MATLAB and C/C*. Web. 18 Apr. 2011. <http://wiki.accelereyes.com/wiki/index.php?title=Main_Page>.

## **Appendix**:

### *General MATLAB code*

```
clear all
close all
clc

im = double(imread('Noisy_PCB.jpg'));
[Y X Z] = size(im);

figure; imagesc(uint8(im(:,:,2))); colormap('gray'); title('Original')

for kk = 1:1
    k(kk) = kk;
    tic
    % for ii = 1:1
    output(:,:,1) = adaptive_med_vect(im(:,:,1),7);
    output(:,:,2) = adaptive_med_vect(im(:,:,2),7);
    output(:,:,3) = adaptive_med_vect(im(:,:,3),7);
    % end
    cpu(kk) = toc;

    im_filt = double(output(:,:,2));
    figure; imagesc(uint8(im_filt)); colormap('gray'); title('CPU')
    %%%
    %%% GPU implementation
    %%%

    gsync;
    tic
    gpu_im = gsingle(im);
    output2_1 = adaptive_med_vect_gpu(gpu_im(:,:,1),7);
    output2_2 = adaptive_med_vect_gpu(gpu_im(:,:,2),7);
    output2_3 = adaptive_med_vect_gpu(gpu_im(:,:,3),7);
    geval(output2_1);
    gsync;
    gpu(kk) = toc;

    output_gpu = double(output2_2);
    figure; imagesc(uint8(output_gpu)); colormap('gray'); title('GPU')
end

%%% ============
%%% Plotting
%%% ============
speedup = cpu./gpu;
figure
plot(k,speedup,'.',k,cpu,'x',k,gpu,'*')
grid on
title('Speedup of GPU Implementation')
xlabel('Iteration Number');
legend('Speedup','CPU Time (seconds)','GPU Time (seconds)')
ylim([0 5])
```

### *CPU Filter*

```
function f = adaptive_med_vect(g, Smax)
%ADAPTIVE_MED_VECT Perform adaptive median filtering.
% F = ADAPTIVE_MED_VECT( G, SMAX ) performs adaptive median filtering of
% image G.  The median filter starts at size 3-by-3 and iterates up to
% size SMAX-by-SMAX.  SMAX must be an odd integer greater than 1.
%
% SMAX must be an odd, positive integer greater than 1.
```

```
if (Smax <= 1) || (Smax/2 == round(Smax/2)) || (Smax ~= round(Smax))
    error('SMAX must be an odd integer > 1.')
end
% [M, N] = size(g);

%Initial setup
f = g;
f(:) = 0;
alreadyProcessed = false(size(g));

%Begin Filtering
for k = 3:2:Smax
    zmin = ordfilt2(g, 1, ones(k, k), 'symmetric');
    zmax = ordfilt2(g, k * k, ones(k, k), 'symmetric');
    zmed = medfilt2(g, [k k], 'symmetric');

    processUsingLevelB = (zmed > zmin) & (zmax > zmed) & ~alreadyProcessed;

    zB = (g > zmin) & (zmax > g);
    outputZxy = processUsingLevelB & zB;
    outputZmed = processUsingLevelB & ~zB;
    f(outputZxy) = g(outputZxy);
    f(outputZmed) = zmed(outputZmed);

    alreadyProcessed = alreadyProcessed | processUsingLevelB;
    if all(alreadyProcessed(:))
        break;
    end
end

% Output zmed for any remaining unprocessed pixels.  Note that this
% zmed was computed using a window of size Smax-by-Smax, which is
% the final value of k in the loop.
f(~alreadyProcessed) = zmed(~alreadyProcessed);
```

*GPU Filter*

```
function f = adaptive_med_vect_gpu(g, Smax)
%ADAPTIVE_MED_VECT Perform adaptive median filtering.
% F = ADAPTIVE_MED_VECT( G, SMAX ) performs adaptive median filtering of
% image G.  The median filter starts at size 3-by-3 and iterates up to
% size SMAX-by-SMAX.  SMAX must be an odd integer greater than 1.
%
% SMAX must be an odd, positive integer greater than 1.

if (Smax <= 1) || (Smax/2 == round(Smax/2)) || (Smax ~= round(Smax))
    error('SMAX must be an odd integer > 1.')
end
% [M, N] = size(g);

%Initial setup
f = g;
f(:) = 0;
alreadyProcessed = false(size(g));

%Begin Filtering
for k = 3:2:Smax
    g_d = double(g);
    zmin = ordfilt2(g_d, 1, ones(k, k), 'symmetric');
    zmax = ordfilt2(g_d, k * k, ones(k, k), 'symmetric');
    zmin = gdouble(zmin);
    zmax = gdouble(zmax);
    zmed = medfilt2(g, [k k], 'symmetric');
%     zmed = gdouble(zmed);
```

```
    processUsingLevelB = (zmed > zmin) & (zmax > zmed) & ~alreadyProcessed;

    zB = (g > zmin) & (zmax > g);
    outputZxy = processUsingLevelB & zB;
    outputZmed = processUsingLevelB & ~zB;
    f(outputZxy) = g(outputZxy);
    f(outputZmed) = zmed(outputZmed);

    alreadyProcessed = alreadyProcessed | processUsingLevelB;
    if all(alreadyProcessed(:))
        break;
    end
end

% Output zmed for any remaining unprocessed pixels.  Note that this
% zmed was computed using a window of size Smax-by-Smax, which is
% the final value of k in the loop.
f(~alreadyProcessed) = zmed(~alreadyProcessed);
```