# Bayer Bilateral denoising on TriMedia 3270

Harold Phelippeau, Mohamed Akil, Breno Dias Rodrigues, Hugues Talbot, Stefan Bara

# Bayer Bilateral Denoising on TriMedia3270

H.Phelippeau[ab], M.Akil[a], B. Dias Rodrigues[c], H.Talbot[a], S.Bara[b]
[a]Université Paris-Est, Labinfo, ESIEE 93162 Noisy-le-Grand Cedex France;
[b]NXP Semiconductors, 2 Esplanade Anton Philips Campus Effiscience, Colombelles BP 2000 14906, Caen Cedex 9, France
[c]Universidade Federal de Minas Gerais - UFMG, Belo Horizonte, Minas Gerais, Brazil

## ABSTRACT

Digital cameras are now commonly included in several digital devices such as mobile phones. They are present everywhere and have become the principal image capturing tool. Inherent to light and semiconductors properties, sensor noise [10] continues to be an important factor of image quality [12], especially in low light conditions. Removing the noise with mathematical solutions appears thus unavoidable to obtain an acceptable image quality. However, embedded devices are limited by processing capabilities and power consumption and thus cannot make use of the full range of complex mathematical noise removing solutions. The bilateral filter [6] appears to be an interesting compromise between implementation complexity and noise removing performances. Especially, the Bayer [5] bilateral filter proposed in [11] is well adapted for single sensor devices. In this paper, we simulate and optimize the Bayer bilateral filter execution on a common media-processor: the TM3270 [4] from the NXP Semiconductors TriMedia family. To do so we use the TriMedia Compilation System (TCS). We applied common optimization techniques (such as LUT, loop unrolling, convenient data type representation) as well as custom TriMedia operations. We finally propose a new Bayer bilateral filter formulation dedicated to the TM3270 architecture that yields an execution improvement of 99.6% compared to the naïve version. This improvement results in real-time video processing at VGA resolution at the 350MHz clock rate.

**Keywords:** data load, fetches, custom operations, data cache

## 1. INTRODUCTION

In photographic digital imaging, sensor noise continues to be a major problem for the output image quality. Embedded devices impose power processing constraints and thus algorithmic limitations. The bilateral filter is an interesting solution because of its simplicity and of its efficient noise removing performances. We focus on the Bayer bilateral filter proposed in [11], a sensor adapted version. The Bayer bilateral filter avoids the perturbation of the noise distribution introduced by the demosaicing step and allows an efficient noise removing. It also has a lower computational complexity due to a lower amount of data to process [11]. In this paper we present the execution simulation and the code optimization of the Bayer bilateral filter on the TM3270 media-processor from the NXP Semiconductors TriMedia family. Our goal is to perform real time-processing at VGA resolution using the maximum processor clock ratio (350MHz). We first present a common photographic imaging processing chain. We then recall the bilateral filter properties and its formulation for the Bayer pattern. In section 5 we present the TM3270 architecture and its programming environment. In section 6, we present the techniques used for the code optimization. It includes common methods (such as LUT, loop unrolling, convenient data type) and custom TriMedia operations. In section 6.6, we present the optimization performances, which we then discuss. In section 7, we propose a new Bayer bilateral filter formulation dedicated for the TM3270 architecture. In section 7.2, we present our new optimization performances results. Finally, in section 8, we present the general conclusion and the perspectives of this work.

## 2. PHOTOGRAPHIC IMAGING PROCESSING CHAIN

As it is presented in [11], a digital image is the result of three main steps, the optical image formation trough the lenses system, the conversion of the light into electrical signal and some image processing operations, including: demosaicing, white balancing, noise removing, gamma curve adjustment, etc. A complete documentation concerning computational photography is presented in [13]. Figure 1 shows a block diagram of a common imaging system. Such a system includes the optical lenses, the color filter array (CFA) and the image sensor (CCD, CMOS). It also includes the main control systems, as automatic gain control (AGC), analog to digital converter (ADC), auto focus and auto exposure circuitry. The digital signal path is completed with color and digital image processing, and is finally sent to the baseband for storage, or to the interface for visualization.

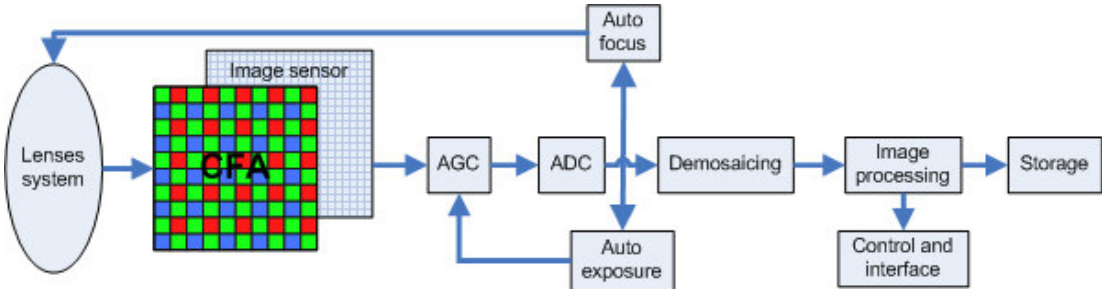



Fig. 1. General block diagram of a digital camera.

## 3. BILATERAL FILTER PROPERTIES

The bilateral filter is a non-linear filter well suited for denoising applications. It exhibits demonstrated effectiveness properties and its formulation simplicity contributes to its popularity. The bilateral filter replaces a pixel value in an image with a weighted mean of its neighbors considering both their geometric closeness and photometric similarities [6, 7, 8, 9]. The most popular version is the Gaussian bilateral filter, it is expressed as follows:

$$v(x) = \frac{1}{C(x)} \sum_{\beta} \exp \frac{-|x-y|}{\rho^2} \exp \frac{-|u(x)-u(y)|}{h^2} u(y), \qquad (1)$$

where $\beta$ represents the sliding window, $y$ is a set of 2-D pixel positions in the sliding window, and $x$ is the 2-D position of the centered pixel in the sliding window. $u(x)$ is the intensity of the pixel at the $x$ position in the original image, $v(x)$ is the estimated pixel at the $x$ position, $\rho$ and $h$ are respectively the standard deviation of the Gaussian distribution of the geometrical and the intensity weight. The filter behavior depends on the parameters setting. Figure 2 shows an example of the bilateral filtering behavior using a varying $h$.

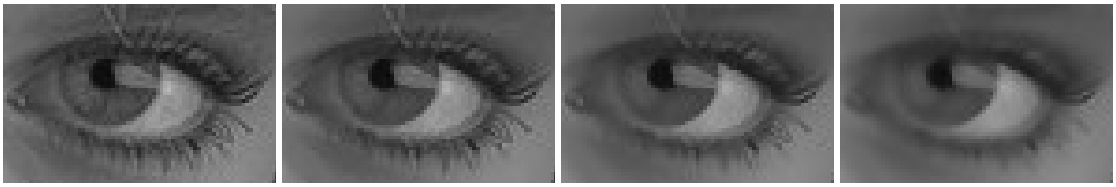


Fig. 2. Gaussian bilateral filtering using a 5x5 square window as $\beta$ and a variable $h$, from left to right $h = 0$, $h = 10$, $h = 20$, $h = 40$.

# 4. BAYER BILATERAL FILTER FORMULATION

In this section we made a rapid presentation of the Bayer bilateral filter proposed in [11]. We present first the Bayer pattern mainly used in single sensor devices. We present then the adaptation of the bilateral filter to the Bayer pattern.

## 4.1 Bayer color filter array

A major defect of single sensors is that they can get only one signal information per photosite. To introduce color information a color filter is superimposed on the sensor. There exist different color filter schemes, the most widely used being the Bayer arrangement [5], it is shown on figure 3.

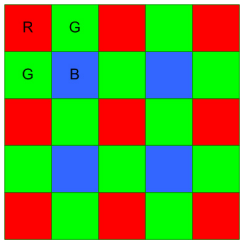


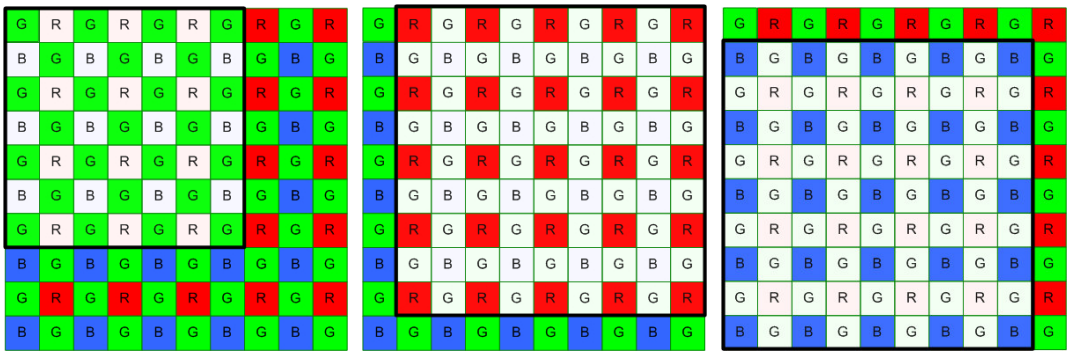

Fig. 3. Bayer color filter arrangement

## 4.2 Bilateral filter formulation for the Bayer pattern

In [11], the authors propose to apply the bilateral filter before the demosaicing step. Indeed, filtering the color filter array allows for better noise removing performance by avoiding the perturbation of the noise distribution induced by the demosaicing step. The Bayer bilateral filter consists of processing each color separately. Green, red and blue filtered values are respectively calculated using green, red and blue pixels. The Bayer bilateral filter is expressed as follows:

$$v(x_c) = \frac{1}{C(x_c)} \sum_{\beta_c} \delta(c - c') \exp \frac{-\left|x_C - y_{C'}\right|}{\rho^2} \exp \frac{-\left|u(x_C) - u(y_{C'})\right|}{h^2} u(y_{c'}) \qquad (2)$$

The symbols are the same as in equation (1), $\delta$ is a discrete Dirac function, $c$ is the color filter characteristic of the current pixel, $c'$ is the color characteristic of a pixel used for the weighted mean, and $\beta_c$ is the size of the square windows. Due to the Bayer pattern arrangement, the size of $\beta_c$ vary considering the processed color. This way we keep the same number of pixels for the weighted mean calculation irrespective of the different color arrangements. Figure 4 shows the kernel windows $\beta_c$ used in the case of a 25 weighted mean pixels. $\beta_c$ is a 7x7 square window for the green pixels convolution, which becomes a 9x9 square window for the red and blue pixels convolution.



(a) (b) (c)

Fig. 4. From (a) to (c), are presented the different convolution kernels used for the green, red and blue bilateral filtering

### 4.3 Computational complexity

The Bayer bilateral filter is a mean weighted filter, it has a linear computational complexity O(n), where n represents the number of processed pixels. Calculations performed for each pixel are listed on Table 1.

Table 1. Bayer bilateral filter algorithm computational complexity.

| Operations | Exponentials | Floating point operations | | | Absolute values | Square roots |
|---|---|---|---|---|---|---|
| | | Multiplications | Additions | Divisions | | |
| location weights | 25 | 75 | 25 | 25 | 0 | 25 |
| intensity weights | 25 | 25 | 25 | 25 | 25 | 0 |
| mean and normalization | 0 | 50 | 24 | 1 | 0 | 0 |
| **total** | **50** | **150** | **74** | **51** | **25** | **25** |

## 5. TM3270 ARCHITECTURE OVERVIEW

### 5.1 TriMedia TM3270 Processor

The TM3270 is a VLIW media-processor from the NXP Semiconductors TriMedia family. Complete characteristics concerning the TriMedia architecture can be found in [1, 2, 3, 4]. Table 2, gives an overview of the main TM3270 architectural features. It operates at up to 350MHz and can execute at up 5 instructions per clock cycle, tuned to match the performance demands of standard definition video processing. It supports SIMD multimedia and IEEE-754 floating-point operations. It uses 128 32-bits registers (unified register-file), a 128kB data cache (4-way set associative) and a 64kB instructions cache (8-way set associative), with 35 execution units and an extensive set of multimedia instructions for video and image processing.

Table 2. TM3270 architecture overview

| Architectural feature | Quantity |
|---|---|
| Architecture | 5-issue slot VLIW, guarded RISC-like operations |
| Pipeline depth | 7-13 stages |
| Address width | 32-bit |
| Data width | 32-bit |
| Register-file | Unified, 128 32-bit registers |
| Functional units | 35 |
| Floating point | IEEE-754 |
| SIMD | 1x32-bit; 2x16-bit; 4x8-bit |
| Instruction cache | 64 Kbytes, 8 way set-associative, 128 byte lines |
| Data cache | 128 Kbytes, 4 way set-associative, 128 byte lines |

### 5.2 TriMedia Programming Environment

The TriMedia processors are supported by a TriMedia Compilation and Simulation System (TCS) that speeds up creation of multimedia applications entirely in the C and C++ programming languages, with an ANSI-compliant C/C++ compilation system. The TCS provides a suite of system software tools to compile and debug multimedia applications, analyze and optimize performances, and simulate execution on a TriMedia processor by a cycle-close machine-level simulator.

## 6. CODE OPTIMIZATION ON TM3270 ARCHITECTURE

In this section, we seek to optimize the Bayer bilateral filter for the TM3270 architecture using the TriMedia Compilation System. Various optimization methods supported by the TCS as well as techniques for exploiting the fine-grain parallelism can be used. We use common code optimization techniques as Look Up Table, loop unrolling, and TriMedia custom operations to take advantage of the TriMedia architecture.

### 6.1 Naïve code version

The naïve code version is the version of the code without any optimizations. This version uses if and else conditions, complex mathematic operations (natural exponential) and none loop optimizations. The pseudo-code of the Bayer bilateral filter is shown on figure 5.

```
count_width=1
count_heigh=1

while count_width<image_width
        while count_heigh<image_heigh
                if pixel_color==red
                     new_pixel_value=red_bilateral_filtering
                else if pixel_color==green
                   new_pixel=green_ bilateral_filtering
                 else
                    new_pixel=blue_ bilateral_filtering
                 end if
         end while
end while
```

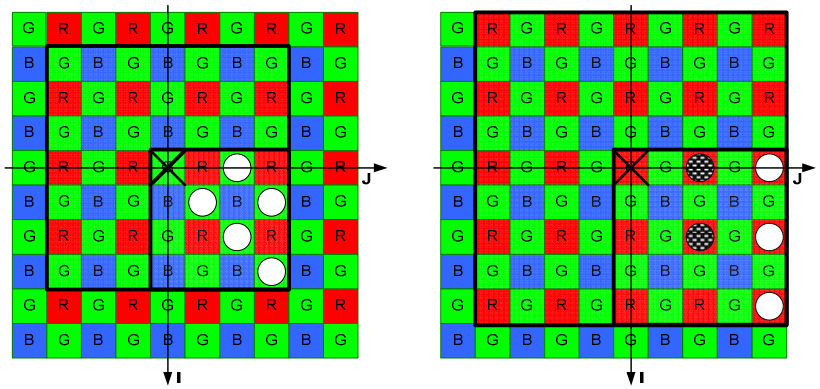
Fig. 5. Bayer bilateral filter naïve pseudo-code

### 6.2 Look up table

The Bayer bilateral filter in the Gaussian version (equation (1)) uses natural exponential computations depending on pixels intensities and locations. The computational complexity can be decreased by replacing these functions by simple memory fetches using look up tables (LUT).

#### 6.2.1 Location weights

Using a 9x9 square window for the red and blue pixels convolution and a 7x7 square window for the green pixels convolution, we deduce that there are 80 possible pixel positions, which lead to use a LUT of 80 float values for the position weights. To reduce the size of the LUT we can take advantage of the Bayer pattern symmetry properties. Indeed, the position of a pixel comparing to the centered pixel is calculated using the 2-d Euclidian distance, it induces no direction dependencies. If we consider an orthogonal axis (I,J) centered at the current pixel position of a 7x7 square window (figure 6), a green pixel can take the coordinates: 1, 2 or 3, on the I and J axis. We can deduce the number of 2-D coordinates permutations without repetition using the combinatory laws: $(3)_2 = 6$. Excluding the coordinate (0,0), we find 5 unique 2-d Euclidian distances. They are presented using white dots on figure 6(a).
Considering now the 9x9 window used for the red and blue pixels convolution, using the same reasoning we find also 5 unique 2-d Euclidian distances, they are represented using white dots on figure 6(b). It can be reduced to 3 possible cases, because 2 of them have already been counted in the green case. They are represented using black dots on figure 6(b). Finally, we concluded that all the location cases comparing to the centered pixel of the convolution kernel, can be stored using a LUT of only 8 float values.




(a) (b)

Fig. 6. (a) shows with white dots the elementary positions used for the green interpolation, (b) shows with white and black dots the elementary positions used for the red and blue interpolation. Black dots mean common position between green and red or blue elementary position.

#### 6.2.2 LUT fusion

Using two LUT induces two fetches and one multiplication for the calculation of one pixel weight. We can decrease the number of operations by making a fusion between the location LUT and the intensity LUT. Using this idea, we create 8 LUT of 255 float values containing pre-multiplied location and intensity weight values. As a consequence, we decrease our computation const from two fetches and one multiplication to a single fetch.

#### 6.2.3 Computational complexity

Calculations performed for each pixel are listed on Table 3. Compared to the naïve code computational complexity (table 1), the LUT utilization allows an important number of simplifications. We note that the 50 exponential calculations are

replaced by 25 fetches; we also note a gain of 75 multiplications, 50 additions, and 50 divisions that are directly due to the weights pre-calculation. Execution results of the code by using this method are presented on table 5.

Table 3. Bayer bilateral filter algorithm computational complexity – Look Up Table version

| Operations | Exponentials | Floating point operations | | | Absolute values | Fetches |
|---|---|---|---|---|---|---|
| | | Multiplications | Additions | Divisions | | |
| location weights | 0 | 0 | 0 | 0 | 0 | 0 |
| intensity weights | 0 | 0 | 0 | 0 | 25 | 25 |
| mean and normalization | 0 | 25 | 24 | 1 | 0 | 0 |
| **total** | **0** | **25** | **24** | **1** | **25** | **25** |
| *total-naïve version* | *50* | *150* | *74* | *51* | *25* | *25* |

## 6.3 Loop unrolling

Loop unrolling is done by replicated several iterations in a single sequence. In digital image processing it is done by computing several neighboring pixels in the same iteration. This technique increases significantly code execution by eliminating the iteration's jump instructions. It provides better processor pipeline utilization and operations scheduling.

By looking at equation (2), we remark that the Bayer bilateral filter computes different filtering treatments according to the pixel color. It induces control flow to select the best treatment (if conditions). We can apply loop unrolling to a Bayer matrix image by processing two or more consecutive pixels in an image line at each iteration. This can partially eliminate the conditional choices generated by the Bayer pattern because two different colors of a line are processed in the same loop iteration. Figure 7 illustrates a loop unrolling in the Bayer bilateral filter algorithm. We note that the control flow appears at each line in the unrolled version, while it appears at each pixel in the original naïve version. Experimental results of using this technique are given in section 6.6.

```
for (j=0; j<height-1; j++)
        {
        for (i=0; i<width-1; i++)
                {
                Switch(pixel_actual){
                Case(Red):
                   Img [j*width+i] = Red_pixel ();
                Case(GreenR):
                   Img [j*width+i] = GreenR_pixel ();
                Case(GreenB):
                   Img [j*width+i] = GreenB_pixel ();
                Case(Blue):
                   Img [j*width+i]  = Blue_pixel ();
                   }
                   Update(pixel_actual)
                }
        }
```

```
for (j=0; j<height-1; j++)
        {
        Switch(Pixel_Line_Start):
           Case(Red):
           for (i=0; i<width-1; i+=4)
                   {
                   Img [j*width+i]    = Red_pixel ();
                   Img [j*width+i+1] = GreenR_pixel ();
                   Img [j*width+i+2] = Red_pixel ();
                   Img [j*width+i+3] = GreenR_pixel ();
                   }
           }
           Case(GreenB):
           for (i=0; i<width-1; i+=4)
                   {
                   Img [j*width+i]    = GreenB_pixel ();
                   Img [j*width+i+1] = Blue_pixel ();
                   Img [j*width+i+2] = GreenB_pixel ();
                   Img [j*width+i+3] = Blue_pixel ();
                   }
        }Update(Pixel_Line_Start);
        }
```

Fig. 7. Versions without loop unrolling (left) and with loop unrolling technique (right) for bilateral filter algorithm.

## 6.4 Convenient data type representation and custom TriMedia operations

Data type representation has an important impact in code optimization. It has influences on the number of cycles per operation, cache memory usage, and storage. To increase the code execution performances it is important to use data types as small as possible. For example, working with char values also reduces the memory required for the Look Up Tables from 8192 bytes of memory (8 LUT of 255 floating point values) to 2048 bytes (8LUT of 255 char values). Optimization results obtained by working with unsigned char values are listed on table 5.

Furthermore, using the TM3270 multimedia processor, it is preferable to work with integer values to take advantage as much as possible of the TM3270 custom operations that use 4-bytes registers. Indeed, it is possible to represent 4, 2 or 1 pixel(s) in each 4-bytes registers depending of their data representation (8-, 16- or 32-bits). These operations allow the processor to compute up to four operations concurrently and reduce the total number of cycles of the code section. Among this set of custom operations the TriMedia library proposes absolute value calculation, maximum or minimum function that can be executed faster than usual C/C++ operations. Figure 8 shows an example of such an operation using the *dspuquadabssub* function used to calculate the rank in one of the 8 intensity weight LUT. This function takes in input two 4-bytes registers that contain four 1-bytes words (pixels). It returns the results of four absolute values differences in a 4-bytes register. We use the function *super_ld32* to retrieve consecutive 8-bytes data from memory in two 4-bytes destination registers. This doubles the memory-processor bandwidth and allows retrieving an image window's convolution line using one instruction. Table 4 illustrates the number of processor cycles necessaries to perform four pixel treatment. The experimental results of custom TM3270 functions use are listed on table 5.

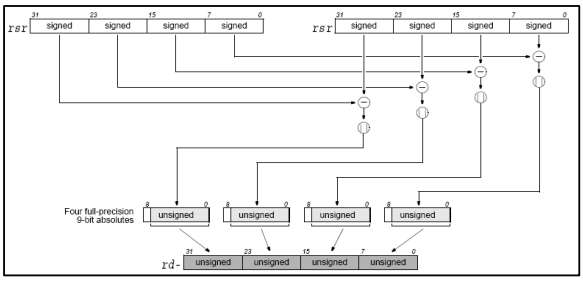



Fig. 8. Scheme of the *dspuquadabssub* function

Table 4. Processor cycles gain using custom TriMedia operations

| | | | Function latency | Latency per 4-pixel computation | Total Cycles |
|---|---|---|---|---|---|
| **Memory data load** | Classical load operation | | 5 | 20 | 20 |
| | **TriMedia *super_ld32()* function** | | **5** | **10** | **10** |
| **LUT index computation** | Classical C functions | Substraction | 1 | 4 | 12 |
| | | Absolute value | 2 | 8 | |
| | **TriMedia *dspquadabssub()* function** | | | **3** | **3** |

## 6.5 Experimental Results

In this section we present the performances of the optimization steps presented above. We use the TCS environment on the TM3270. As optimization measure we use the number of clock cycles, the % of improvement compared to the naïve code version, and the number of frames per second giving a VGA (640x480) image at the maximum clock ratio (350 MHz). VGA is the common embedded devices video resolution and real-time processing is possible from 25 frames/second. Our goal is to obtain a real-time processing at VGA resolution. Optimization results are listed on Table 4.

The first row of the table presents the results of the naïve version. We obtain 6026.1 cycles/pixel, which results in 0.19 frames/second. The second row presents the results obtained by using LUT, we can see the effectiveness of this code transformation. It allows having 92.6% of improvements comparing to the naïve version, giving 445.7 cycles/pixel and 2.56 frames/second. The third row presents the results of the loop unrolling optimization. It shows an important optimization gain with an improvement of 50.82% compared to the LUT version. It performs at 219.2 cycles/pixel and allows the CPU to process 5.2 frames/second. The passage to integer type values reduces the number of cycles/pixel to 155.5 or 7.33 frames/second, an improvement of 97.42% compared to the naïve version and 65.12% compared to the LUT version. Finally the TriMedia custom operations results in an efficient optimization, improving by 98.94% the naïve version and of 85.62% the LUT version. It makes it possible to process 17.78 frames/second and it reduces the number of cycles per pixel to 64.1.

Table 5. Bayer bilateral filter algorithm optimization results on TriMedia TM3270 processor.

| | Cycles per pixel | Improvement/ naive(%) | Improvement/ Look up table version (%) | Frame/second (640x480) @350MHz |
|---|---|---|---|---|
| **Naïve version** | 6026.1 | | | 0.19 |
| **Look up table** | 445.7 | 92.6% | | 2.56 |
| **Loop unrolling** | 219.2 | 96.36% | 50.82% | 5.2 |
| **Integer** | 155.5 | 97.42% | 65.12% | 7.33 |
| **Trimedia custom functions** | 64.1 | 98.94% | 85.62% | 17.78 |

Interpreting these results, we see that by using all these optimizations it is still not possible to achieve real-time processing at VGA resolution. We seek to find why the algorithm formulation is not adapted for the TM3270 processor:

- The first problem appears by looking at the kernel convolution size. Indeed loading lines of 7 and 9 pixels of 1-byte values is not adapted for an efficient use of the *super_ld32* function (that allows loading 8-bytes per call). Particularly, loading a line of 9 pixels of 1-byte values requires two *super_ld32* function call. We estimate that using a 8x8 square convolution window would increase the yield of the *super_ld32* function.

- The second problem is the utilization efficiency of the loaded values. Processing the pixels one by one reduce the utilization efficiency of the loaded values. Indeed because of the Bayer matrix arrangement 50% of the values loaded to process one pixel are not used. We conclude that we must take advantage of the Bayer arrangement and process red, green, and blue values simultaneously to increase the utilization efficiency of the loaded values.

In the following section, we seek to adapt the Bayer bilateral filter formulation considering the discussion above to increase the optimization performances.

# 7. NEW BAYER BILATERAL FILTER FORMULATION DEDICATED FOR TRIMEDIA 3270

## 7.1 New bayer bilateral filter formulation

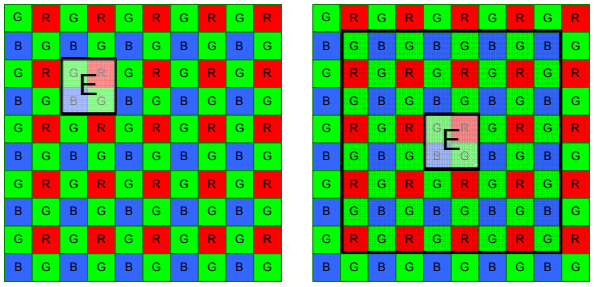
Considering the conclusion of section 6, we seek to propose a new Bayer bilateral filtering formulation:

- To optimize the load function calls, we impose an 8x8 square kernel windows. This configuration enables the loading of a complete kernel line per load instruction, so all the pixels values of kernel can be charged with 8 *super_ld32* instructions. Two 32-bits destination variables are used to store the pixels values loaded, in an

unsigned char codification (4 pixel values are stored in each destination variable), allowing the four-operand instructions use. The pixel values loaded are ready to use by TriMedia special functions, like *dspuquadabssub()*.

- To optimize the utilization efficiency of the loaded values we impose to process red, green and blue interpolation simultaneously. To do so, we can take advantage of the Bayer pattern arrangement. As we can see on figure 9(a), the Bayer pattern is the repetition of a 2x2 square element E that includes red, green, and blue pixel. We propose to process this kernel at each iteration. This way we process all the loaded pixels values from the window kernel by computing simultaneously the values of the element E. In addition, custom TriMedia instructions are used more effectively by avoiding data ranking.

The new convolution kernel can be represented as in figure 9(b), it as a 8x8 square size and a 2x2 centered element E.




(a) (b)

Fig. 9. Scheme of the new Bayer bilateral filter formulation

Figure 10 illustrates the absolute value of pixels difference calculation needed to calculate the LUT index. RgRgXX and GBGBXX represent a couple of 4-bytes loaded data. RgRg0 and GBGB0 represent the central convolution kernel pixels. In the following section we show the experimental results obtained with this new formulation.

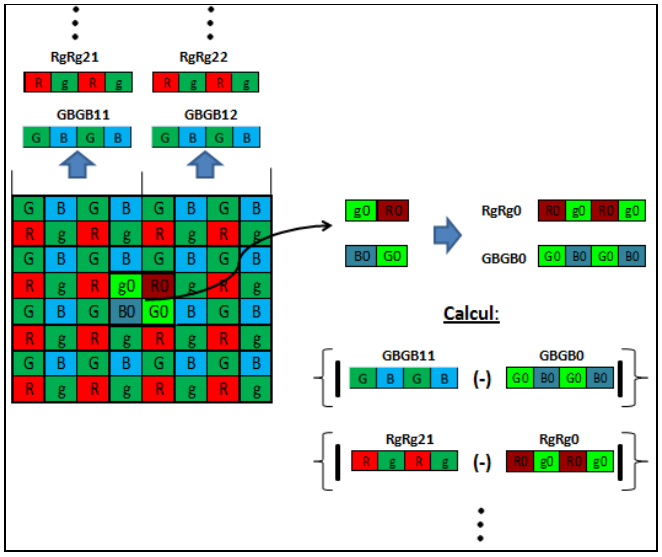



Fig. 10. Scheme of the new Bayer bilateral filter formulation

## 7.2 Experimental results

In this section we present the performances optimization of the new Bayer bilateral filter formulation. As in section 6, we use the TCS environment on the TM3270. As optimization measure we use the number of clock cycles, the % of improvement compared to the naïve code version, and the number of frames per second giving a VGA (640x480) image at the maximum clock ratio (350 MHz).

We used the new formulation approach with a convolution kernel line size up to 8 bytes (8x8 window), as discussed before. Loop unrolling was implicitly performed (we treat 4 pixels in a 2x2 central square at once) and explicitly by processing two central square element at one loop iteration. We use the LUTs system presented in section 6.3 and we work with unsigned char types values. We also experiment another version of the algorithm using a convolution kernel with 6-bytes of line size (6x6 square window). The results of these experiments are shown on table 6 and on figure 11.

We can see that using our new algorithm formulation allows to improve the algorithm execution efficiency by 49.61% compared to the full optimized original version (corresponding to the TriMedia custom functions row in the table). It allows us to process 35.27 frames/second, thus achieving real time processing.
Reducing the convolution kernel size to a 6x6 square improves the algorithm execution efficiency further to 61.93% compared to the full optimized original version. It allows us to process 46.9 frames/second, giving thus an important margin for real time processing.

Table 6. Bayer bilateral filter algorithm optimization results on TM3270 processor: new formulation

| | Cycles per pixel | Improvement/ naive(%) | Improvement/ Look up table version (%) | Improvement/ Naïve + TM custom fonctions version (%) | Frame/second (640x480) @350MHz |
|---|---|---|---|---|---|
| **Naïve version** | *6026.1* | | | | *0.19* |
| **Look up table** | *445.7* | *92.6%* | | | *2.56* |
| **Loop unrolling** | *219.2* | *96.36%* | *50.82%* | | *5.2* |
| **Integer** | *155.5* | *97.42%* | *65.12%* | | *7.33* |
| **Trimedia custom functions** | *64.1* | *98.94%* | *85.62%* | | *17.78* |
| **New formulation-8x8 windows** | 32.3 | 99.46% | 92.75% | 49.61% | 35.27 |
| **New formulation-6x6 windows** | 24.3 | 99.6% | 94.55% | 61.93% | 46.9 |

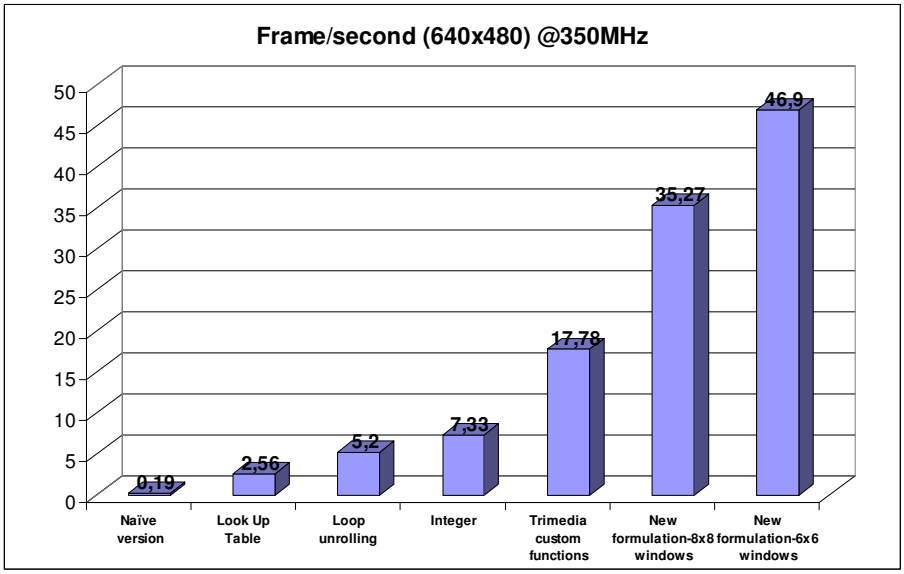



Fig. 11. Histogram of the Bayer bilateral filter algorithm optimization results on the TM3270 processor

## 8. CONCLUSION

In this paper, we have simulated and optimized the Bayer bilateral filter execution on the TM3270 using the TCS tool. Our goal was to reach real time processing using a VGA resolution image at the maximum processor clock ratio (350MHz). We have used common optimization method as LUT utilization, loop unrolling, convenient data type representation; we also used the custom TriMedia operations to take advantage of the TriMedia architecture. Despite using all this optimization tools and method it was still impossible to achieve real time video processing using the original formulation of the Bayer bilateral filter. The maximum speed was found to be 17.78 frames/sec. By analyzing the code and the custom TriMedia function call, we have found that the Bayer bilateral filter formulation does not allow an efficient utilization of the data load function (*super_ld32*). We have also found that is prevents an efficient utilization of the loaded data (we notice that 50% of the loaded data are not usef). To resolve these problems we have proposed a new Bayer bilateral filter formulation dedicated for the TM3270. It uses a 8x8 (or a 6x6, depending on the version) square convolution kernel , that results in an efficient use of the load function, and a 2x2 centered square element that results in a full utilization of the loaded data. Using this formulation it is finally possible to reach real-time processing with speeds of 35.27 frames/second for the 8x8 convolution kernel version, and 46.9 frames/second for the 6x6 convolution kernel. It improves the naïve version algorithm by 99.46% and 99.6% respectively. The real-time processing margin obtained with this two versions makes it possible to perform consecutive real-time processing.

## REFERENCES


[1] Rathnam, S and Slavenburg, G. An architectural overview of the programmable multimedia processor, tm-1, In Proceedings of the COMPCOM'96, pp. 319-326,1996.

[2] HAlfhill, T. Philips power up for video, IN Microprocessor Report, http:/www.mpronline.com/, November 2003.

[3] Hennessy, J.L. and Patterson, D.A. Computers Architecture: A Quantitative Approach, 3rd edition, Morgan Kauffmann, 2003.

[4] van de Waerdt, J.-W.; Vassiliadis, S.; Sanjeev Das; Mirolo, S.; Yen, C.; Zhong, B.; Basto, C.; van Itegem, J.-P.; Dinesh Amirtharaj; Kulbhushan Kalra; Rodriguez, P.; van Antwerpen, H. "The TM3270 media-processor", 38th Annual IEEE/ACM International Symposium on Microarchitecture Proceedings, pp .12, 12-16 Nov. 2005.

[5] B. E.Bayer, Color Imaging Array. United State Patent, 3,971,065, 1976.

[6] Carlo Tomasi and Roberto Manduchi, “Bilateral filtering for gray and color images,” pp. 839–846, 1998.
[7] V.Aurich and J.Weule, “Non-linear gaussian filters performing edge preserving diffusion,” Proceedings of the DAGM Symposium, 1995.
[8] Stephen M. Smith and J. Michael Brady, “Susan a new approach to low level image processing,” Int. J. Comput. Vision, vol. 23, no. 1, pp. 45–78, 1997.
[9] S.Paris and F.Durand, “A fast approximation of the bilateral filter using a signal processing approach,” European conference on computer vision, 2006.
[10] R. Costantini and S. Susstrunk, “Virtual sensor design,” Proc. IS&T/SPIE Electronic Imaging 2004: Sensors and Camera Systems for Scientific, Industrial, and Digital Photography Applications, vol. 5301, pp. 408–419, 2004.
[11] H.Phelippeau, H.Talbot, M.Akil and S.Bara, “Bayer bilateral denoising filter,” IGM2008-04.
[12] C. Liu, R. Szeliski, S. B. Kang, C. L. Zitnick, and W. T. Freeman, “Automatic estimation and removal of noise from a single image,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 30, no. 2, pp. 299–314, 2008.
[13] R. Raskar, J. Tumblin, A. Mohan, A. Agrawal, and Y. Li, “Computational photography,” State of the Art Report, 2006.