2013 International Conference on Electronic Engineering and Computer Science

Videoclip Denoising Algorithm Implementation Using the Blackfin Microcomputer Family

Sorin Zoican*

POLITEHNICA University of Bucharest, Bucharest, Romania

Abstract

This article presents a general framework for scheduling videoclips denoising processes ensuring the quality of service (QoS). In general, a denoising algorithm has two phases which are run sequentially: the first one determines the noisy pixels in the videoclip frames and the second applies a median filtering over the each frame considering the only good pixels. In all such denoising algorithms, the first phase is run for multiple times depend on the noise power. The second phase also may be executed more than one time but this depends on the specific algorithm. The issue in such applications is the denoising process may not terminate within its deadline. The proposed solution adapts the execution time in such way so the deadline to be respected by determining the remaining time to the deadline before running each phase and reducing the number of runs in each phase in order to not exceed the deadline. The goals of the article are the following: presents the QoS scheduling algorithm and proposes an implementation solution of based on a Blackfin microcomputer with support of Visual DSP kernel (VDK). The article is organized in several sections: a brief introduction to set up the general context of quality of service in videoclips denoising applications, the modified video processing algorithm, the proposed solution, its VDK implementation and performance evaluation, and the conclusions.

© 2013 The Authors. Published by Elsevier B.V. Open access under CC BY-NC-ND license.
Selection and peer review under responsibility of Information Engineering Research Institute

Keywords: Impulsive noise removal; noisy pixel detector; median filtering; quality of service; digital signal processor;

* Corresponding author. Tel.: +40214024803
E-mail address: sorin@elcom.pub.ro.
1. Introduction

Impulsive noise, called “salt and pepper”, is caused by camera sensors, faulty hardware memory locations, or because errors occurred during communication channels transmitting images, affecting randomly a fraction of the total number of pixels. It may be eliminated using a denoising algorithm based on median filtering. However, the median filter may cause blurred in the reconstructed image. To overcome this phenomena and preserved the edges, a noised pixel detector is applied before median filtering. The noised pixels detector is repeatedly applied over the image [1], [2]. Unfortunately, this additional phase added to the classical median filter increases the computation time and it is possible that the application not run in real time. Moreover, many image denoising algorithms have a second phase that computes median value adaptively using the results from the first phase. The Figure 1 illustrates an image denoising algorithm with two phases. For videoclips, the algorithm is applied each frame.

2. The QoS scheduling algorithm

In real time systems, there are specific deadlines to be met. In particular, for a video processing application, deadlines are determined by the number of frames per second. There are methods that may be used to maintain system functionality: reserving of additional resources, tasks skip, adaptation of the task activation period and an adaptation of task execution time. In embedded systems, with limited resources, additional resources reserving can not be a viable option. More, the activation period is fixed and it not be modified without a severe degradation in performance. In such systems only the task skipping and execution time adaptation may lead to deadline meeting while preserving the functionality. The videoclip processing is divided into two task category: mandatory tasks and additional tasks [6]. The mandatory task ensures a basic quality of the videoclip and the additional tasks improves the quality. In case that the deadline may be exceeded, some of additional tasks (or all of them) will be skipped. The execution time adaptation is as similar methods. In this situation, the videoclip processing may be divided into several tasks consists of iterative sequences. The image processing performance increases with the number of iterations of each task. In this paper the execution time adaptation is used. The scheduling algorithm should be aware about the remaining execution time of each task before it reaches its deadline.

Fig. 1. The denoising algorithm
If the remaining time is less than a proper chosen threshold the scheduling algorithm notify the task to modify the execution parameters in phase 1 and in phase 2 so that the execution time to be reduced and the deadline to be not exceeded. The scheduling algorithm is illustrated in the Figure 2. After each iteration in phase 1 and phase 2, the remaining time to the deadline is evaluated and the computations for the current phase may terminated. Using this scheduling algorithm, each frame will be processed as well is possible and therefore the quality of service (QoS) is ensured. The detailed algorithm is the following:

**Initial data**
- Corrupted image: $I = \{I(i, j) \mid i = 0,..N-1, j = 0,...,M-1\}$
- Noise matrix: $N = \{N(i, j) \mid i = 0,..N-1, j = 0,...,M-1\}$
- Current value good pixel count: $C = 0$
- Previous value of good pixel count: $C_1 = 0$
- Noisy pixels counter in the current window: $S = 0$
- Scanning window length: $L = 3$
- Maximum scanning window length: $L_{max} = 5$
- Reconstructed image: $J = \{J(i, j) \mid i = 0,..N-1, j = 0,...,M-1\}$

**Phase 1**

*For* $i = 0..N-1$ and $j = 0..M-1$

1. Update the current window, centered of the current pixel
   
   $p = I(i, j)$, $W = \{I(i+k_1, j+k_2) \mid k_1, k_2 \in \{-L, -L+1,\ldots, L\}\}$

2. Compute minimum and maximum element in the current window:
   
   $w_{min} = \min(W \setminus \{p\})$, $w_{max} = \max(W \setminus \{p\})$

3. Compute the noise matrix elements and good pixel counter:
   
   If $p \in [w_{min}, w_{max}]$ then $(N(i, j) = 0$ and $C = C + 1)$ else $N(i, j) = 1$

*End For*

*If* (remaining time < threshold1) *then*

*If* $(C < C_1$ and $L \leq L_{max})$ *then* $(C = C_1, L = L + 1$, Repeat Phase1) *else* goto Phase2

**Phase 2**
For $i = 0..N - 1$ and $j = 0..M - 1$
4. $L = 1$, $S = 0$
5. Update the current window, centred of the current pixel
   \[ p = I(i, j), W = \{I(i + k_1, j + k_2)\} , k_1, k_2 \in \{-L, -L + 1, \ldots, 0, \ldots, L - 1, L\}\]
6. Count the noisy pixels in the current window:
   \[ \text{If } (N(i + k_1, j + k_2) = 1, k_1, k_2 \in \{-L, -L + 1, \ldots, 0, \ldots, L - 1, L\}) \text{ then } (S = S + 1)\]
7. Test the noisy pixel count:
   \[ \text{If remaining time < threshold2 then} \]
   \[ \text{If } (S < S_1 \text{ or } L < L_{\text{max}}) \text{ then goto 5 else goto 8}\]
   \[ \text{else goto End Phase2} \]
8. Compute median value and set the current pixel in the reconstructed image:
   \[ J(i, j) = \text{median}(\{W^*\}) , W^* = W \mid N(i + k_1, j + k_2) = 0, k_1, k_2 \in \{-L, -L + 1, \ldots, 0, \ldots, L - 1, L\}\]
End For
End Phase 2

3. The implementation and performance evaluation

In order to meet the time constraint, two thresholds are tested after each iteration in Phase 1 and Phase 2. Without testing the deadlines the quality of videoclip will be dramatically degraded due the fact that some very noisy frames will be not processed. The above presented algorithm was implemented on Blackfin digital signal processors family [3], [5] with VDK [4] support that provides critical kernel features: preemptive scheduler (time slicing and cooperative scheduling), thread creation, semaphores, interrupt management, inter thread messaging, events, and memory management. The image processing algorithm may be implemented easily using VDK functionality: each phase in algorithm will be separately coded in a dedicated task and a time measurement mechanism will be defined using a periodic semaphore. The following VDK primitives will be involved to measure the current execution time: MakePeriodic() and GetSemaphoreValue(). A semaphore is defined and it is declared as periodic semaphore by calling MakePeriodic primitive (the semaphore is posted every tick). In the processing image task, the GetSemaphoreValue is called and the value returned is used as a local execution time. If this value is greater than thresholds discussed above, the proper actions will be taken (the processing parameters for Phase 1 and Phase 2 will be modified in order to deadline to be not exceeded). The following threads have been defined: Main, P1 and P2. The Main thread creates P1 thread and set the periodic semaphore then it is destroyed. The P1 thread creates P2 thread, read the periodic semaphore value and executes an iteration of Phase 1 of the algorithm. Before continuing with other iteration in this phase, P1 thread read again the periodic semaphore value and calculates the time interval to deadline. If this interval is greater enough, a new iteration of Phase 1 is running, else the P1 task exits and P2 task will be executed. The P2 task performs similar operations as P1 but it implements the Phase 2 of the algorithm, instead Phase 1 as P1 task. An identical mechanism to measure the time interval to the deadline is used as in P1 task. Figure 4 illustrates how periodic semaphore is involved in time measuring. The main issue here is to achieve a real time implementation. The proposed algorithm controls the execution of Phase 1 and Phase 2 to reduce the number of iterations of each of them with respect of the deadlines. If the frame processing is complex, (if the impulsive noise has higher power or the frame size is larger) then the QoS scheduling algorithm will reduce the execution time so the tasks can complete within their deadline. However, at least one iteration of each phase must be completed. In certain situation that may take a large execution time, therefore
special methods to optimize the tasks execution should be involved. These methods will be discussed below for achieving a real time implementation using the digital signal processing Blackfin microcomputer family. The Blackfin processor has an orthogonal instruction set and single instruction multiply data (SIMD) instructions. Issuing parallel instructions and using vector operations may be used to obtain a real time functioning. The Blackfin processor does permit up to three instructions to be issued in parallel: one 32-bit DSP instruction and two 16-bit instructions (load/store, DSP load). A powerful feature of Blackfin processors is the existence of instructions that manipulate video pixels. Such instructions perform 8-bit pack and unpack, quad 8-bit subtract operation that can be used to compute the minimum and maximum values in four windows simultaneously. Using assembly language implement the iterative instruction as hardware loops that save processor cycles. In the Figure 5 is illustrated how the minimum of four data sets is computed in parallel. The current values are subtracted from the current minimum values in parallel using a specific video pixel instruction. The result is stored in two registers. Using shift operations and bit-wise logical AND operations the sign extensions of the subtractions are aligned in a single 32 bits register that will contains four groups of 8 bits with all bits equal with 1 or 0 depend on the result of subtracting operation (that is, if all bits in the group are 1 the result is negative, otherwise the result is positive). The next operations perform a bit-wise logical AND between current minimum values and aligned sign extension and a bit-wise logical AND between current values and negated aligned sign extension. The results of these two operations are concatenated using a bit-wise logical OR obtaining the new minimum for all four values.

Table 1. The computational effort for computing median value

<table>
<thead>
<tr>
<th>Data set length</th>
<th>Quick sort method</th>
<th>Selection sort (using quad operations)</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>29</td>
<td>9</td>
</tr>
<tr>
<td>25</td>
<td>116</td>
<td>62</td>
</tr>
<tr>
<td>49</td>
<td>275</td>
<td>231</td>
</tr>
<tr>
<td>81</td>
<td>514</td>
<td>625</td>
</tr>
</tbody>
</table>

Table 2. The execution time for various grayscale image sizes

<table>
<thead>
<tr>
<th>Image size</th>
<th>FPS</th>
<th>Lmax = 1</th>
<th>Lmax = 2</th>
<th>Lmax = 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(176,144)</td>
<td>266</td>
<td>95</td>
<td>48</td>
<td></td>
</tr>
<tr>
<td>(320,240)</td>
<td>87</td>
<td>31</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>(480,320)</td>
<td>43</td>
<td>15</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>(640,480)</td>
<td>21</td>
<td>7</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

Table 3. The mean squared error

<table>
<thead>
<tr>
<th>MSE</th>
<th>Corrupted frame</th>
<th>Reconstructed frame</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Lmax = 1</td>
<td>Lmax = 2</td>
</tr>
<tr>
<td>35</td>
<td>17</td>
<td>19</td>
</tr>
<tr>
<td>39</td>
<td>20</td>
<td>21</td>
</tr>
<tr>
<td>42</td>
<td>35</td>
<td>33</td>
</tr>
</tbody>
</table>
The above presented schema is as well involved for maximum value computing and in noise matrix updating. If the median value in the second phase of the denoising algorithm is computing using a selection sorting algorithm, this schema can be used for median filtering too. The table 1 shows the computational effort for median value computing using the quick sorting method and the selection sort method that uses the above method to compute minimum value. One can observe that for a medium number of values in the data set the computational effort is smaller for the proposed method. This is the case of median filtering used in the denoising algorithm. Additionally, dual-core Blackfin processors, (BF561 and BF60x) may be involved. The frame processing task is designed as a dual-core application that allows for splitting the main code on the two cores. Two successive input frames may be processed in the two cores of the processor, as illustrated in the Figure 5. In Figure 5, $T$ is the frame rate and $T_c$ represents the computation time for the current frame. Each core in the Blackfin processor has its own interrupt system. The input frames are acquired from an input port that generates a common interrupt for both cores (indicated in the Figure 4 as IRQ_core_A and IRQ_core_B). Each core implements the frame processing algorithm in its own main program (denoted as Main_core_A and Main_core_B) if an appropriate flag (flag_A or flag_B) is set to 1. These flags are set in the interrupt service routines for core A or core B. A necessarily functioning condition is $T_c < 2T$ (the flowchart for core B is is similar to the flowchart of core A). Two counters, $n_A$ and $n_B$ were defined to determine the even and odd frames. Table 2 illustrates the average execution time, in frames per seconds, for various image sizes. In table 3 is shown the mean squared error (MSE), in decibels, for corrupted frames and reconstructed frames. One can observe that the new median filtering can be used for color image size about $320 \times 240$, if the admitted frame per second is minimum 25.
Fig. 4. Computing minimum value using Blackfin quad operations.
4. Conclusion

This work proposed a scheduling algorithm for image processing tasks which have two computations steps both of them depend of noise power. The scheduling algorithm trades off between the quality of the restored image and the constraint to meet the deadlines. The paper presents a framework for real time implementation of such image processing algorithms that ensure a reasonable quality of the videoclips in systems prone to impulsive noise but meeting deadlines.

Acknowledgements

This work has benefited the support of FP7 project ALICANTE—FP7—ICT—2009—4 no. 248652.

References
