An FPGA Sliding Window-Based Architecture Harris Corner Detector

Alexandru Amaricai             Constantina-Elena Gavriliu             Oana Boncalo
University Politehnica Timisoara
Computer Engineering Department
Timisoara, Romania
{alexandru.amaricai, oana.boncalo}@cs.upt.ro

Abstract— This paper proposes a FPGA implementation based on sliding processing window for Harris corner algorithm. It represents one of the most frequently used pre-processing method, for a wide variety of image processing algorithms, such as feature detection, motion tracking, image registration, etc. It relies on a series of sequential steps, each processing an image outputted by the previous step. The purpose of the sliding window is to avoid storing intermediate results of processing stages in the external FPGA memory or to avoid utilize large line buffers typically implemented with BRAM blocks. Therefore, the entire processing pipeline benefits from data locality. Implementation results for Virtex5 and Spartan-6 devices show that the proposed solution has very good performance (more than 130 fps for 1280x720 images in Xilinx Spartan-6) with significant less BRAM usage with respect to other approaches.

Keywords—Image processing, corner detection, Harris’s algorithm

I. INTRODUCTION

Corner detection represents one of the widely used methods in image processing. It is an important pre-processing component in many computer vision applications, such as feature detection, motion tracking, image registration, 3D modeling, object recognition [2][3][4]. Harris algorithm represents one of the most accurate corner detection methods, especially when dealing with noisy images [1]. The main drawback when using it is its high computational requirements, due to the sequential nature of executing the five sub-operations required by this algorithm: gradient computation, Gaussian smoothing, Harris measure computation, thresholding and non-maximum suppression [1]. Each of these sub-operation outputs an image, which is fed as input to the following sub-operation. In order to increase the performance of Harris edge detection, a possible solution would be to buffer several lines of each sub-operation’s outputs. This approach requires the usage of line buffers between the processing units corresponding to each sub-operation [5][7][8]. Thus, these implementations present a high number of BRAM blocks required for line buffer implementations.

This paper proposes two types of architectures for Harris corner detection on FPGA which try to overcome the limitations of having a large number of line buffers. The first architecture performs the entire computation required for determining a corner pixel locally without the usage of large line buffers. This is achieved using a sliding window of pixels and several parallel processing units for each sub-operation. This approach uses a small 3x3 neighborhood for non-maximum suppression. Increasing the size of the non-maximum suppression window for this approach will result in a significant increase of slices required by the accelerator.

In order to address the significant slice based resources increase required by a larger window for the last sub-operation, a second architecture is proposed, which performs non-maximum suppression on 7x7 neighborhood. This implementation represents a hybrid between the one which performs all the computations locally and the ones based on line buffers after each sub-operation: the first four sub-operations are performed locally, while the last - non-maximum suppression – uses a series of 7 line buffers. With respect to the full local architecture, the hybrid one presents better accuracy, and lower slice consumption; however, it has higher usage of BRAM blocks.

The paper is organized as follows: Section II presents Harris corner detection and a review of existing FPGA accelerators for this algorithm; Section III is dedicated to the local processing architecture; the hybrid implementation is described in Section IV; implementation results and comparisons are presented in Section V.

II. FPGA IMPLEMENTATIONS FOR HARRIS CORNER DETECTION

A. Harris Corner Detection Algorithm

Harris corner detection relies on the fact that a corner has strong gradients in all directions [1]. Gaussian filtering is included for noise elimination. The algorithm consists of the following steps [2][3][6]:

1. Gradient computation. Derivatives for both X and Y axes are computed for each pixel. The computation of the two gradients \( (D_x \text{ and } D_y) \) represents the convolution between the 3x3 pixels neighborhood and a gradient mask. Based on the vertical and horizontal gradients, the \( D_{xx} = D_x \times D_x \), \( D_{yy} = D_y \times D_y \) and \( D_x \times D_y \) values are computed.
2. **Gaussian smoothing.** Gaussian smoothing is applied on the three gradient images \((D_x^2, D_y^2, D_x D_y)\) obtained in the previous step. Processing consists of the convolution between a Gaussian matrix with the corresponding pixel neighborhood in the three gradient images. Three Gaussian values are obtained \((I_x^2, I_y^2, I_{xy})\).

3. **Harris measure.** This measure is a good indicator of whether a pixel may be a corner. In order to compute the Harris measure, the following matrix is used.

\[
S = \begin{pmatrix}
I_x^2 & I_{xy} \\
I_{xy} & I_y^2
\end{pmatrix}
\]

The eigenvalues of this matrix indicate whether a pixel is part of a flat region, an edge or a corner. Because eigenvalue computation is computationally intensive, a Harris measure is used instead [1].

4. **Thresholding.** A corner pixel is characterized by a large positive value of Harris measure. Thus, applying a corresponding threshold value reduces pixels associated to flat and edge regions.

5. **Non maximum suppression.** The corner pixel is characterized by the largest Harris measure within a corner region. This step eliminates all the other pixels within a corner region.

Steps 1, 2 and 5 require processing on the pixel neighborhood. For Gaussian smoothing, the neighborhoods are taken from the three images obtained after the gradient computation. The non-maximum suppression neighborhood is taken from the image obtained after thresholding. It is worth emphasizing that steps 3 and 4 apply to values associated to the corresponding pixel, and not to a neighborhood. Thus, the algorithm requires storing intermediary images after two steps: three images after gradient computation and one image obtained after thresholding.

**B. FPGA Implementations of Harris Corner Detection**

Several FPGA implementations for Harris corner detection have been presented in [5][6][7][8]. All these approaches rely on the usage of line buffers implemented using BRAM modules for caching the intermediate images obtained after each processing stages. The approach in [5] uses a computational pipeline composed of the four combinational processing units (gradient computation, Gaussian smoothing, Harris measure and thresholding, and non-maximum suppression) and four sets of BRAM based line buffers. The processed neighborhood is stored in register based memory elements, connected to the outputs of the line buffers. The approach in [7] is similar in many aspects with the one in [5]. The main difference is given by the fact that all the computations are performed on 8 bits. In this case, saturation is applied in case overflow occurs. Computations on 8 bits have been carried out in order to reduce the size of the line buffers. For this implementation, all the pixels associated to a corner region will have the same Harris measures. It uses another Gaussian filtering between the Harris measure and non-maximum suppression, which requires an additional set of line buffers.

The solution proposed in [8] performs corner detection for color values (RGB), and not for grayscale values as in [5][7]. The main characteristic of this approach is represented by the utilization a single memory module (implemented with several BRAM blocks). This module is shared by all the processing units; this is a significant difference with respect to [5][7], which used dedicated memory buffers for each processing unit.

The main contribution of our paper is represented by performing the entire computation pipeline associated to the Harris corner detection locally, without the use of line buffers.

**III. LOCAL PROCESSING HARRIS CORNER DETECTION**

Our approach tries to reduce the amount of large line buffers on the one hand, for cost optimization by reducing the number BRAM blocks and on the other hand for increased performance by reducing the time required to fill the line buffers. Thus, the proposed implementation relies on performing the entire computation process locally. Fig. 1 depicts the size of the processing window (neighborhood) in the original image used to compute the corner pixel, using a 3x3 neighborhood for non-maximum suppression and 3x3 Gaussian filter. In order to determine whether a pixel is a corner, a 3x3 Harris measures have to be computed; for the 9 Harris measures, 3x3 sets of Gaussian values are required; these require 5x5 set of values for gradients; the 25 sets of values for gradients are obtained using a 7x7 processing window from the original image. The proposed implementation for Harris corner detection is composed of the following:

1. **Pixel buffer** – A 7x3 buffer is used for storing the required pixels for the 5 gradient computations. It is implemented using conventional slice based registers

2. **Gradient computation modules** – 5 parallel computation modules are used for computing the 5 sets gradient values. These values are computed in two clock cycles: the first clock cycle is used for computing \(D_x\) and \(D_y\), while the multiplications are performed in the second clock cycle.

3. **Gradient buffer** – It is used for storing 5x3 sets of gradient values. It provides the inputs for the Gaussian smoothing units.
4. **Gaussian smoothing units** – 3 parallel processing units are used to compute the three sets of Gaussian smoothed values. A 3x3 Gaussian filter is used for smoothing operation. The latency for this operation is of 2 clock cycles.

5. **Harris measure and threshold unit** – 3 parallel units are used for determining the Harris measure and for applying the threshold value. Each of these units has as inputs a set of gradient values and outputs one Harris measure. The latency of this unit is of 2 clock cycles.

6. **Harris measure buffer** – It is used for storing 3x3 thresholded Harris measures. It provides the inputs for the non-maximum suppression unit.

7. **Non-maximum suppression unit** – Non-maximum suppression is applied on a 3x3 neighborhood. If the center pixel is greater than the entire neighborhood, then it is considered a corner pixel.

Fig. 2 describes the proposed local processing architecture for Harris corner detection. The input pixels are provided through set of input line buffers.

### IV. HYBRID PROCESSING ARCHITECTURE

Using a small neighborhood (3x3) for non-maximum suppression may affect the corner detection process. Using a larger neighborhood leads to a larger slice-based cost for the accelerator. In this sub-section we present a hybrid architecture which uses a 7x7 neighborhood for non-maximum suppression. The inputs for the last sub-operation in the Harris corner detection are provided by a set of 7 line buffers, as in [5][7]. The first four sub-operations are performed locally, as in the architecture described in the previous section.

Fig. 3 depicts the hybrid processing pipeline for Harris corner detection. The main differences with respect to the full local processing architecture are:

1. **Reduced size of the pixel and gradient buffer** – 5x3 pixel and 3x3 gradient buffers are used instead of the 7x3 pixel buffer and 5x3 gradient buffers used in the full local architecture.

2. **Reduced number of processing units** – The hybrid architecture requires 5 gradient computation units, 3 Gaussian smoothers and only 1 Harris measure and threshold unit.

3. **Increased size of the non-maximum suppression unit** – For a 7x7 non-maximum suppression window, 48 comparisons are performed instead of 8 required for a 3x3 window.

4. **Increased usage of line buffers** – A series of 7 line buffers are placed between the threshold and non-maximum suppression units.

The hybrid processing architecture presents a reduced slice usage, at the cost of an increased number of BRAM blocks.

### V. RESULTS AND COMPARISON

The two architectures for Harris corner detection have been implemented on Xilinx Spartan6 FPGA device (Digilent Atlys board) and Xilinx Virtex-5 FPGA device (Digilent Genesys) using Verilog HDL. Synthesis and implementation have been performed using Xilinx ISE 14.4 software. Table I presents cost and performance results for different image resolutions. The proposed solution has better overall performance (for both Spartan-6 and Virtex-5) with respect to the approach in [5], and similar with respect to [7] Virtex-5 implementation. Regarding maximum frequency, the Altera Aria V implementation in [7] has similar working frequency with respect to Xilinx Virtex-5 proposed implementation.

Cost comparisons are difficult to make; the approaches in [5][7] report the cost for the entire image acquisition and processing system. This also includes the cost for modules such as camera and display controllers, external memory controller, etc. However, we may conclude that the proposed approach has significant less BRAM usage, which is due to the reduced number of line buffers. Furthermore, we also observe for the proposed approach a reduce in number of DSP blocks.
used (24 for full local vs 76 in [7] and 29 in [5]); because these types of modules are generally used only for data processing, we may say that the vast majority of these are utilized within the Harris corner detection accelerators.

VI. CONCLUSIONS

This paper proposes two types of FPGA based architectures for Harris corner detection algorithm. The first architecture performs the entire computation pipeline in a local manner, using only conventional FPGA logic resources. The proposed solution does not require large line buffers for storing intermediate results. This leads to reduced BRAM utilization. Several sub-operations within Harris algorithm, such as gradient computation, Gaussian smoothing or Harris measure computation, are performed using parallel processing units. Small register based buffers are used to store intermediate results between the computational units. Increasing the size of the non-maximum suppression window or the size of the Gaussian filter will result in the increase of the number of parallel computational units and the size of the intermediate registers buffers; thus, the number of required slices will increase.

A second architecture which tackles the problem of larger non-maximum suppression window is proposed. It represents a hybrid between the full local solution and the approaches which rely on storing intermediate results between different sub-operations. A set of line buffers is placed between the Harris measure and threshold unit and the non-maximum suppression unit. This way, it is possible to increase the size of the non-maximum suppression window without increasing the number of logic slices, at the expense of larger utilization of BRAM blocks.

The proposed solutions present reduce BRAM and DSP blocks usage with respect to approaches in [5] [7]. Regarding performance, it presents significant increase with respect to the one in [5], while having comparable throughput and working frequency to the one in [7].

ACKNOWLEDGEMENTS

This work was supported by the Romanian National Authority for Scientific Research (UEFISCDI) project number PN-II-RU-TE-2011-3-0186.

REFERENCES


TABLE I – IMPLEMENTATION RESULTS

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Platform</th>
<th>Resolution</th>
<th>LUT-FF pairs/Logic elements</th>
<th>Memory blocks</th>
<th>DSB Blocks</th>
<th>Frequency (MHz)</th>
<th>Preformance (fps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>[7] Altera Arria V</td>
<td>1024 x 1024</td>
<td>8624</td>
<td>43</td>
<td>76</td>
<td></td>
<td>232</td>
<td>93</td>
</tr>
<tr>
<td></td>
<td>1476 x 1680</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>166</td>
<td></td>
</tr>
<tr>
<td>Full Local Processing</td>
<td>Xilinx Spartan6</td>
<td>1024 x 1024</td>
<td>4296</td>
<td>4</td>
<td>24</td>
<td>153</td>
<td>146</td>
</tr>
<tr>
<td>Harris Corner Detection</td>
<td>1280 x 720</td>
<td>4351</td>
<td>7</td>
<td>4</td>
<td>24</td>
<td>225</td>
<td>214</td>
</tr>
<tr>
<td></td>
<td>Xilinx Virtex-5</td>
<td>1024 x 1024</td>
<td>4099</td>
<td>4</td>
<td>24</td>
<td>225</td>
<td>244</td>
</tr>
<tr>
<td></td>
<td>1280 x 720</td>
<td>4125</td>
<td>4</td>
<td>12</td>
<td></td>
<td>138</td>
<td>130</td>
</tr>
<tr>
<td>Hybrid Harris Corner Detection</td>
<td>Xilinx Spartan6</td>
<td>1024 x 1024</td>
<td>4067</td>
<td>13</td>
<td>23</td>
<td>125</td>
<td>134</td>
</tr>
<tr>
<td></td>
<td>1280 x 720</td>
<td>4149</td>
<td>13</td>
<td></td>
<td></td>
<td>125</td>
<td>134</td>
</tr>
<tr>
<td>Hybrid Harris Corner Detection</td>
<td>Xilinx Virtex-5</td>
<td>1024 x 1024</td>
<td>4367</td>
<td>10</td>
<td>12</td>
<td>225</td>
<td>213</td>
</tr>
<tr>
<td></td>
<td>1280 x 720</td>
<td>4497</td>
<td>13</td>
<td></td>
<td></td>
<td>225</td>
<td>241</td>
</tr>
</tbody>
</table>