Edge-Based Segmentation on Maxeler

Christian Permann

Faculty of Computer Science, University of Vienna,
Währinger Straße 29, 1090 Vienna

Abstract This paper explains the basics of Edge-Based Segmentation and how to trivialy implement it on a Maxeler architecture. This should result in a high speedup, since the calculations can benefit a lot from the DataFlow paradigm.

Contents

1 Introduction and Basic Idea

Edge-Based Segmentation is an image processing method that aims to find the outlines and shapes of objects within a picture. For this the image is usualy analysed in grayscale and a filter may be applied. Filters look at a window of size $n \times n$ (here 3×3) around the targe pixel and have different logic for choosing a pixel. There are three main filter variants fo be considered for this task.

High-Pixel Filter The High-Pixel Filter looks for the highes pixel value within the window and selects it for the target pixel.

Low-Pixel Filter The Low-Pixel Filter looks for the lowest pixel value within the window and selects it for the target pixel.

Median-Pixel Filter The Median-Pixel Filter looks for the median pixel value within the window and selects it for the target pixel.

After this filter is aplied the value of each pixel in the image is compared to a threshold, which is usualy chosen as the overall average or median pixel value, and either rounded to the white or to the black value bepending on if the value is higher or lower.

This results in a purely black and white representation of the image which suffices for easier edge and shape recognition for humans. To further analyse this for shapes it is now easy to compare each pixel with its neighbours and if not all of them are the same, the target pixel is part of an edge. With this logic a new image can be created form this one, which contains a certain color for edge pixels and white for non-edge pixels. The resulting edges may also still be improved with different methods.

2 Implementation

For this university assignment the task was to extract one loop from the algorithm and implement it with Maxeller Java. For this the filtering and rounding was chosen, especialy because the median filter may increase the amount of work per iteration, such that using the accelerator has a higher payoff.

The main challenges of implementing this were firstly dealing with the boundary cases when looking at the 3 by 3 window arround pixels on the border of the image. For this a counter was used and each access of a window element is checked if it still is within the bound, otherwise the center pixel is used because we know for sure that it is in the image bounds.

It is also possible to accelerate the reduction to just edges, although it may not pay off as much because there is less work per loop iteration and the comunication with the accelerator may be more expensive then the performance gain.

2.1 Maxeler Execution Graph

As already mentioned, an execution graph is generated by the Maxeler compiler which visualizes how the FPGA will be reprogrammed. The execution graph for this example can be seen in Figure 1.

Figure 1: Execution graph for calculating r

3 Conclusion

The implementation of the calculation of the Pearson Correlation Coefficient shows nicely what Maxeler can be used for. This is a trivial first example that shows especialy well how IO works and the different variable types. The proposed solution and code written in Maxeler Java could still be improved to for example allow for varying array sizes durring runtime, aswell as returning intermediate results. Depending on the application this may not be needed though, omiting these features can save on space on the FPGA and may therefor allow for calculating larger problem sizes.

Acknowledgments

The author would like to thank Milos Kotlar and Prof. Veljko M. Milutinovic for their teachings on the subject of Maxeler and DataFlow computing.