OPTIMIZATION AND BENCHMARK OF VISION ALGORITHMS ON A DSP

BAUMGARTNER, Daniel; KUBINGER, Wilfried & ROESSLER, Peter

Abstract: *This paper shows our work on performance optimized implementations of low-level vision algorithms on a Digital Signal Processor (DSP). The platform is a TI TMS320C6414 DSP running with 1 GHz and is therefore a cutting edge DSP. Performance optimization steps for DSP implementations are shown. A final benchmark compares the DSP with Field Programmable Gate Array (FPGA) implementations.*
Key words: *DSP, FPGA, Performance, Computer Vision, Embedded System.*

1. INTRODUCTION

Computer vision algorithms are getting more and more important. The realization of such computer vision algorithms needs enormous computing power of the hardware. Cutting edge DSPs provide enough performance for computer vision algorithms if optimized algorithms are used. Highly optimized code with the use of intrinsics saves runtime and helps them to meet real time constraints. Porting the algorithm on an embedded system brings up questions like which hardware fits the needs of the application best, how fast can the algorithm be processed and how many resources are required. Selecting an embedded hardware has big influence on the performance. The scope of this paper is to present guidelines for the realization of fast DSP implementations. Furthermore, the performance of these implementations is compared with FPGA realizations.

2. LOW-LEVEL VISION ALGORITHM

2.1 Gaussian Pyramid

Two dimensional low-pass filters, such as the Gaussian low-pass filter, (Gonzalez & Woods 2002), work with a filter kernel of size (e.g.) 5x5 pixels, which calculates an average value for a destination pixel using a number of neighboring source pixels. The two dimensional Gaussian filter is defined by Equation 1.

$$g(x,y) = \frac{1}{\sqrt{2\pi\sigma}} \cdot e^{\frac{-(x^2+y^2)}{2\sigma^2}} \qquad (1)$$

2.2 Bayer filter demosaicing

The Bayer filter demosaicing algorithm, (Kimmel 1999), converts the raw image pixels from a Bayer color filter array into RGB values for every image pixel, as shown in Figure 1.

2.3 Sobel edge detector

The Sobel edge detector, (Gonzalez & Woods 2002), performs a gradient measurement over the x- and y- coordinates with separate filter kernels for each dimension, see Equation 2.

$$C_V = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} C_H = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix} \qquad (2)$$

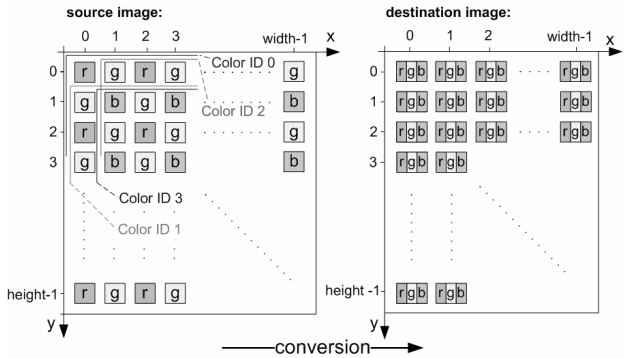

Fig. 1. Bayer filter demosaicing

2.4 RGB to HSL Converter

The HSL (Hue-Saturation-Lightness) color space model, (Foley & Van Dam 1996), defines colors more naturally, where hue specifies the base color and the other two values specify the saturation of that color and how bright the color should be. This color space model has the advantage that the conversion, see Equ. 3-5, from the RGB-model works without the use of trigonometrical functions, which is an important factor for the computational conversion.

$$L = \frac{Max(R,G,B) + Min(R,G,B)}{2} \qquad (3)$$

$$S = \begin{cases} \text{undefined} & \text{if } Max(R,G,B) = Min(R,G,B) \\ & \text{if } L = 0 \\ \frac{Max(R,G,B) - Min(R,G,B)}{Max(R,G,B) + Min(R,G,B)}, & \text{if } L < 0.5 \\ \frac{Max(R,G,B) - Min(R,G,B)}{2.0 - Max(R,G,B) - Min(R,G,B)}, & \text{if } L \geq 0.5 \end{cases} \quad (4)$$

$$H = \begin{cases} \text{undefined} & \text{if } Max(R,G,B) = Min(R,G,B) \\ & \text{if } S = 0 \\ & \text{if } S = undefined \\ \frac{1}{6} \cdot \frac{G-B}{Max(R,G,B) - Min(R,G,B)}, & \text{if } Max(R,G,B) = R \\ \frac{1}{6} \cdot \frac{B-R}{Max(R,G,B) - Min(R,G,B)} + 2, & \text{if } Max(R,G,B) = G \\ \frac{1}{6} \cdot \frac{R-G}{Max(R,G,B) - Min(R,G,B)} + 4, & \text{if } Max(R,G,B) = B \\ H + 1 & \text{if } H < 0 \end{cases} \quad (5)$$

3. DSP IMPLEMENTATION

The selected DSP TMS320C6414T-1000 from Texas Instruments is one of today's cutting edge DSPs. It is equipped with 1 MByte internal RAM and has a Very Long Instruction Word (VLIW) architecture. There are numerous constraints which should be considered for making a fast implementation (Texas Instruments 2006). Unnecessary inner loops for filter kernel handling should be removed first. All lines which are necessary for the filter kernel should be loaded together in the innermost loop, which is often the loop to handle the x-coordinate. To enable the compiler making a pipelined loop implementation, the innermost loop should not have function calls, divisions and branches. The use of "const" and the keyword "restrict" decreases the loop carried dependency. Software functions on DSPs typically can have their performance improved by using specific intrinsics. Intrinsics are special built-in functions like _dotpu4 or _mem4_const, which the compiler can directly translate into machine code. The intrinsic _dotpu4 (op1[31:0], op2[31:0]) allows to perform the operation op1[31:24] * op2[31:24] + op1[23:16] * _op2[23:16] + op1[15:8] * _op2[15:8] + op1[7:0] * _op2[7:0] in one calculation cycle. "Operand 1" is loaded into a 32 bit local variable, which stores the pixel values, and also "Operand 2", which stores the multiplication factors for the Gaussian pyramid. Figure 2 shows an optimization summary of the Gaussian pyramid with a performance gain by a factor of 12.66.

The strategy of improving performance is also applied to the image algorithm Bayer filter demosaicing and to the Sobel edge detector. This is possible because these low-level algorithms have nearly the same structure.

This optimization strategy works not exactly for the RGB to HSL color space converter, because several branch instructions, see Equ. 3-5, and divisions prevent a fast implementation. Instead of a division a multiplication with the reciprocal value is performed. For the multiplication the fast intrinsic mpylir(int src1, int src2) is used, which performs a signed 16 times 32 bit multiply with a following shift right by 15 bits. The multiplication factor is derived from a look up table, where the reciprocal values for 8 bit divisors are stored. It has the advantage to be fast, see Figure 3, and accurate at the same time. The if-conditions are converted in a way, that the conditions are precalculated and stored as const-statements. This includes that both branches of an if-condition must be calculated and then multiplied with the const-statement, but enables on the other side a pipelined loop implementation.

4. RESULTS

Figure 4 shows the performance for both the DSP and the FPGA implementations (derived from Baumgartner et al., 2007) of the low-level image processing algorithms. For the DSP implementations two implementation variants (using IRAM, ERAM + ROSDMA) are shown. As shown in Figure 4, the IRAM-based DSP implementation of the Sobel edge detector is only slightly faster than the implementation using ROSDMA, which shows that the ROSDMA technique (Zinner et al. 2007) is able to reduce the performance gap between having the image data contained in the IRAM or the ERAM. On the first sight all DSP implementations outperform the FPGA ones.

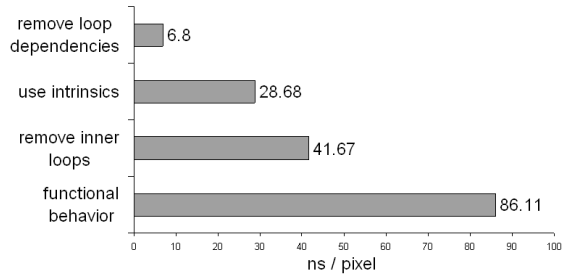

Fig. 2. Optimization summary of the Gaussian pyramid

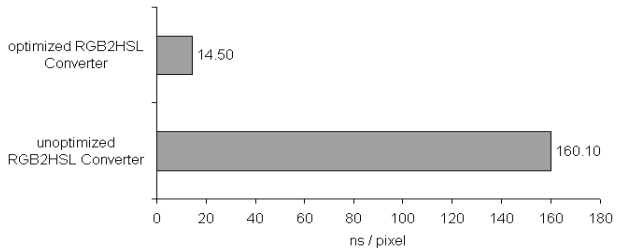

Fig. 3. Optimization potential of color space converter

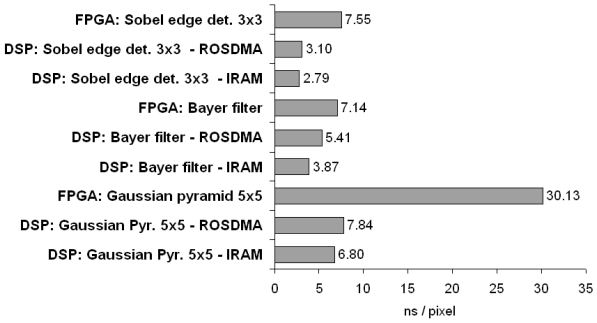

Fig. 4. Performance Benchmark for DSP and FPGA

For a complex vision system a lot of operations have to be processed by the DSP, the processing power which is available for each single operation decreases substantially. FPGA image processing blocks are able to process data concurrently. For example, if a Bayer filer and three Sobel edge detectors horizontally and vertically are processed on an FPGA for an image size of 800 x 400 pixels, the computation time per pixel is calculated with Equation 6. This results in a calculation time per image of 2.418 ms.

$$t_{PIX} = \left(\frac{1}{rows \cdot column} \cdot \sum_{i=1}^{7} T_{LAT} \right) + \frac{1}{f_{CLK}} = 7.555ns \qquad (6)$$

The same combination of low-level image processing algorithms with the use of ROSDMA+ERAM on the DSP needs 4.707 ms per image (800 x 400 pixels), calculated with Equation 7.

$$t_{PIX} = \sum_{i=1}^{4} T_{ExecPix} = 14{,}710ns \qquad (7)$$

It is shown that the DSP algorithms perform faster than their FPGA counterparts on a sub-function basis. However, from the system view the FPGA has advantages due to exploiting parallelism. This advantage increases the more algorithms are processed in parallel, where only the latency time increases for the FPGA implementation.

5. ACKNOWLEDGEMENTS

This research has been supported by the European Union project Robots@Home under grant FP6-2006-IST-6-045350.

6. REFERENCES

Baumgartner D.; Roessler P. & Kubinger W. (2007) Performance Benchmark of DSP and FPGA Implementation of Low-Level Vision Algorithms, *Proceedings of Computer Vision and Pattern Recognition –ECVW 2007,* ISBN: 1-4244-1180-7

Foley J. D. and Van Dam A. (1996) *Color space conversions,* pp.1–34, http://www.wmin.ac.uk/ITRG/docs/coloreq

Gonzalez R. C. and Woods R. E. (2002). *Digital Image Processing*, Pearson Education International

Kimmel R. (1999). Demosaicing: Image reconstruction from color ccd samples. *IEEE Transactions on Image Processing* pp. 1221-1228

Texas Instruments Inc. TMS320C6000 Programmer's Guide, 2006. Literature Number: SPRU198i

Zinner C.; Kubinger W. and Isaacs R. PfeLib – a performance primitives library for embedded vision. *EURASIP Journal on Embedded Systems*, 2007, doi:10.1155/2007/49051