Automated Design of an ASIP for Image Processing Applications

Henjo Schot and Henk Corporaal

Delft University of Technology
Department of Electrical Engineering
Section Computer Architecture and Digital Technique
P.O. Box 5031, 2600 GA Delft, The Netherlands
H.J.M.Schot@FEL.tno.nl, H.Corporaal@et.tudelft.nl

Abstract. This paper presents the design of highly optimized TTA architectures for image processing applications. An automatic processor design framework as described in [2] is used. Specialized hardware is used to improve the performance-cost ratio of the processors. An explorer searches the design space for solutions that are good in terms of cost and performance. We show that architectures can be found that efficiently execute very different algorithms at low cost. A hardware feasible architecture is presented that efficiently executes a set of image processing algorithms and performs almost equally or better than alternative, commercial-available solutions do.

1 Introduction

In this paper, we show the design of an application specific instruction set processor (ASIP) for a set of image processing algorithms. Processors and code are generated, trying to exploit the instruction level parallelism of image processing algorithms. We show that processors can be generated that efficiently execute very different algorithms at low cost. We add application specific hardware and functionality to improve the performance cost of the processors.

The architecture of the ASIP we develop is a Transport Triggered Architecture (TTA) [2]. An automated design framework, called the MOVE framework [], is used for the development of the VLIW like processor. It tries to find an architecture with an optimal cost/performance ratio. The designer can use Special Function Units (SFUs) to improve the cost-performance ratio of an architecture. These SFUs can be incorporated in the MOVE framework.

Our work on the design of a highly optimized processor architecture differs from others [3] in that we use TTAs and application specific hardware (SFUs) in order to find architectures with higher performance-cost ratios.

The next section describes the image processing algorithms we used. Section 3 shows the mapping of the algorithms to TTAs. Section 4 presents the results and conclusions.

A. Bode et al. (Eds.): Euro-Par 2000, LNCS 1900, pp. 1105-1109, 2000.

2 Image Processing Algorithms

Four image processing algorithms were mapped to TTAs. A color conversion algorithm, and three gray-scale neighborhood algorithms: convolution and two edge detection algorithms on a 3x3 area. Here we concentrate on the color conversion algorithm.

Color conversion is an operation from the area of color image processing. It is used to convert between color representations e.g. a color in RGB has to be converted to represent the same color using CMYK color components. There are several methods to perform color conversion. In our case we uses lookup tables (LUTs) and tri-interpolation.

Using this method, the conversion of a color P starts with searching the lookup table for the eight nearest points. Seven linear interpolations are performed on these points. Figure 1 shows point P and its eight nearest points that correspond with the corners of a cube. The interpolations are shown as bold lines. The final interpolation is performed on V and W. The distance *a* of point V to point P and the distance *1 - a* of point W to point P are used as interpolation coefficients.

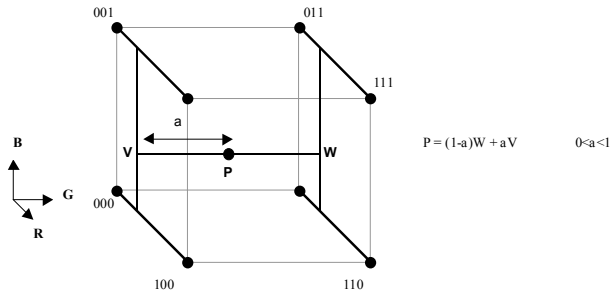

Fig. 1. Tri-linear interpolation of point P in the RGB space.

Since the calculation of each pixel is independent of other pixels, in principle, all pixels can be processed in parallel. The amount of parallelism that can be attained by TTAs is determined by the maximum number of available resources and the ability of the compiler to exploit the parallelism.

In our implementation, we use a LUT of 4 K entries. This size results in reasonable interpolation accuracy at low cost. Higher accuracy requires larger lookup tables. The LUT is addressed using the 4 most significant bits of each color component. The 4 least significant bits are used for the interpolation distance giving interpolation coefficients of 0, 1/16, ... 16/16. We aim to achieve a performance that is comparable or better than that of available solutions (12.5 Mpixels/s [7]), at lower cost.

3 Mapping the Algorithms to TTAs

The main components of the MOVE framework are a retargetable C/C++ compiler, a processor generator and hardware modeller and a design explorer. The explore tool searches the design space for architecture solutions using hardware cost and performance as its main design criteria. The explore tool drives both the compiler and the hardware modeler in order to find architecture solutions with a good performance/cost ratio. A pareto-curve with the resulting architecture solutions is produced, from which the designer chooses an architecture configuration.

The mapping of the algorithm starts with analyzing the solutions that are found in case we use basic operations only. The curve 'w/o SFUs' in figure 2 shows that the latency for high cost architectures remains quite long. E.g.. an architecture of cost 300 (in integer units) does color conversion of a single pixel in 11 cycles.

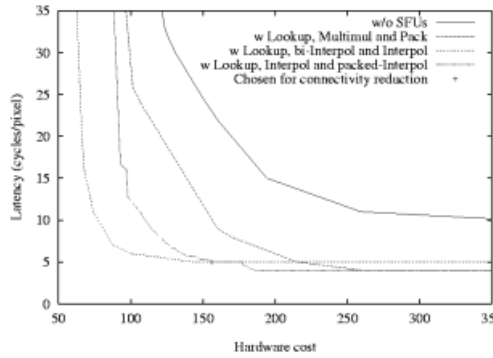

Fig. 2. The TTA design space for the color conversion application.

An enormous improvement in performance is achieved when SFUs are used. Parts of the algorithm that are implemented in hardware are the lookup operation and a linear interpolation. Figure 3 shows the dataflow symbols of different implementations of the interpolation. They are implemented by extending a multiplier FU with these specific functions. The impact of the SFUs on architectures cost/performance is also shown in figure 2. Solutions with 2 to 60 times better performance at equal cost are found.

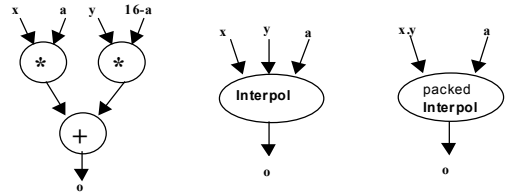

Fig. 3. Dataflow symbols of the interpolation functions

Architectures that execute a set of algorithms are found by combining multiple algorithms in an application. Resulting architectures showed a small loss in

performance for each algorithm, but overall they performed very well, as can be seen in figure 4.

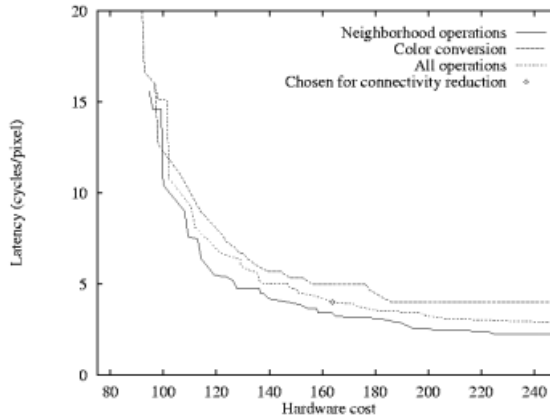

Fig. 4. The TTA design space for both color conversion and neighborhood operations.

4 Results

A feasible processor configuration which efficiently executes the whole set of algorithms is selected from the curve in figure 4. In this configuration, marked with '+', the in- and outputs of each FU and register file are connected to all buses, which is impractical. We therefore remove as much connections as possible without performance losses. The resulting processor is shown in figure5. It can do color conversion at 5.3 cycles/pixel. It contains 8 buses and 11 functional units (FUs). The register file, as shown, contains many read and write ports. However, the tools allow to split up this file into multiple small register files [4][5].

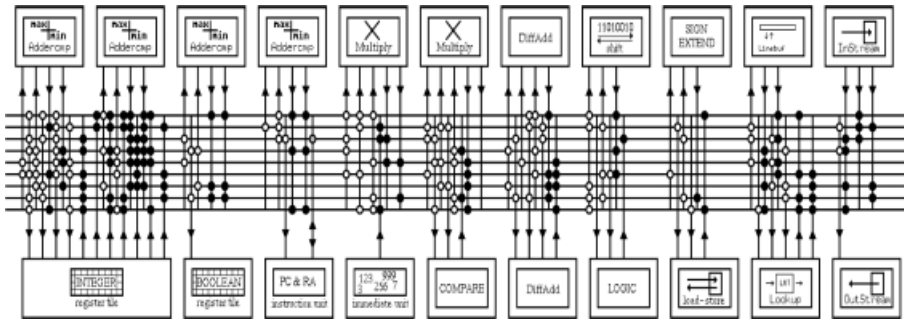

Fig. 5. A processor configuration that efficiently executes the color conversion algorithm and the whole repertoire of neighborhood operations.

Table 1 gives an overview of the performance of commercial available solutions [6]-[11] and our solution. It is seen that the Imagine and the C6x perform better for the convolution application than our solution does. For the other algorithms our solution performs significantly better.

Table 1. Performances of the applications for other available solutions and our solution.

Algorithm	Performance (Mpixels/s)				
	Imagine @ 66 MHz	PixelMagic 44 @ 75 MHz	TI C6201 @ 200 MHz	Chameleon	MOVE @ 100 MHz
convolution	37	22.8	40	n.a.	27
Min-max operation	< 15	11.8		n.a.	27
Edge detection	< 15	< 11.8		n.a.	23
Color conversion	3.0	-	-	12.5	19

5 Conclusion

In this paper, we showed that the MOVE framework can be used to find solutions for digital image processing algorithms. Solutions can be found for applications containing several algorithms, including very different ones. Furthermore, hardware feasible solutions are found that perform almost equally or better than alternative, commercially available solutions.

A large part of the processor design is done automatically. However, a lot of manual interaction is required in the identification and application of Special Function Units. Automation of this part of the design trajectory is currently being researched [1].

References

1. Marnix Arnold and Henk Corporaal, Automatic Detection of Recurring Operation Patterns, Codes '99, May 1999
2. Henk Corporaal, *Microprocessor Architectures; from VLIW to TTA,* John Wiley, 1998, ISBN 0-471-97157-X
3. Joseph A. Fisher, Paolo Faraboschi and Guiseppe Desoli, *Custom-Fit Processors: Letting Applications Define Architectures*,
4. Jan Hoogerbrugge, *Code Generation for Transport Triggered Architectures*, Delft University of Technology, 1996
5. Johan Janssen and Henk Corporaal, *Partitioned Register File for TTAs*, Delft University of Technology
6. Redford, J., Iler, J. and Berger, E., *The PM44: A Single-Chip SIMD GigaOp DSP for Imaging*, Pixel Magic Inc, Andover MA
7. *The Barco Chameleon ASIC; A very high speed, very high accuracy, color correction utility*, Barco Graphics, 1993
8. Redford, J., Iler, J. and Berger, E., *The PM44: A Single-Chip SIMD GigaOp DSP for Imaging*, Pixel Magic Inc, Andover MA
9. *Evaluation of the Performance of the C6201 Processor & Compiler*, Loughborough Sound Images plc., 1996
10. *TMX320C6201 DIGITAL SIGNAL PROCESSOR*, Texas Instruments Inc., 1997
11. *The Imagine engine; Documentation & User Manual*, Arcobel Graphics B.V., March 1994