High Volume Colour Image Processing with Massively Parallel Embedded Processors

Jan Jacobs[a], Winston Bond[b] Roel Pouls[a], Gerard J.M. Smit[c]

[a]Océ Technologies BV

[b]Aspex Semiconductor Ltd

[c]University Twente

Currently Océ uses FPGA technology for implementing colour image processing for their high volume colour printers. Although FPGA technology provides enough performance it, however, has a rather tedious development process. This paper describes the research conducted on an alternative implementation technology: software defined massively parallel processing. It is shown that this technology not only leads to a reduction in development time but also adds flexibility to the design.

1. Introduction

Océ Technologies B.V. develops products and services for the professional printing market, from small format office printers (up to A3) to wide format design department printers (up to A0+).

Océ is investigating ways to reduce the rather long development trajectory for the printer's colour image processing subsystem, which is currently implemented with FPGA technology. The FPGA technology, consumes more development time than a software defined system, such as DSPs, general purpose processors and associative processors. This research is inspired by the potential advantages of a massively parallel software defined system, namely flexibility and shorter design cycles, while attaining equivalent or better performance (by scalable design).

Many of the image processing tasks in a printer show simple massively parallel processing. Therefore, Océ has performed research into applicability of massively parallel embedded processors with similar characteristics. This research has been conducted in co-operation with Aspex, a fabless semiconductor company specialising in high performance, software programmable, parallel processors based on associative technology [5].

The problem addressed in this paper is: "can we use associative processing for high performance colour image processing?"

In chapter 2 the reader is introduced to some important concepts like: colour image processing, associative processing and FPGAs. The next chapter deals with the specification of the problem, which is followed by the mapping onto the associative technology. Finally the results, among which a comparison with an FPGA implementation, are given.

2. Related Work

Colour Image Processing

Colour image processing deals with the transformation of an input colour image into an output colour image with suitable properties (e.g. for the printing process). Three important issues that will be discussed are: image representation, image transformation and desired properties with respect to printing.

- Images are represented as matrices in which each element contains a single integer for monochrome or multiple integers for colour images. The resolution of an image, expressed in for example pixels/inch, directly relates to the dimensions of this matrix.

- Image transformations or operations, which change a pixel's value, may be divided into two categories. In the first category (point operation), for example, the result of thresholding a grey level pixel only depends on the value of the pixel itself. The second category is called neighbourhood operation. In this category pixels are taken from a certain environment in order to compute the resulting value of the pixel, e.g. edge detection.

- Printing high quality colour documents involves a carefully selected set of image transformations which are combined in a colour image pipeline. Examples are given in section 3.

Associative Processing

Traditional computers, rely upon a memory that stores and retrieves data by its address rather than by its content. In such an organisation (von Neumann architecture), every accessed data word must travel individually between the processing unit and the memory. The simplicity of this retrieval-by-address approach has ensured its success, but has also produced some inherent disadvantages. One is the von Neumann bottleneck, where the memory-access path becomes the limiting factor for system performance. A related disadvantage is the inability to linearly increase the performance of a unit transfer between the memory and the processor as the size of the memory scales up [1]. Associative memory, in contrast, provides a naturally parallel and scalable form of data retrieval for both structured data (e.g. sets, arrays, tables, trees and graphs) and unstructured data (raw text and digitized signals). An associative memory can be easily extended to process the retrieved data in place, thus becoming an associative processor. This extension is merely the capability of writing a value in parallel into selected cells [6]. Applications range from handheld gaming, multimedia, base stations, on-line transaction processing, image processing, pattern recognition and data mining [5].

Aspex's Linedancer is an implementation of a parallel associative processor. The approach taken by Aspex Semiconductor is to use many simple associative processors in a SIMD arrangement. Each of the 4096 processing elements on the Linedancer device has about 200 bits of memory (of which 64 are full associative) and a single bit ALU, which can perform a 1 bit operation in 1 clock cycle. Operations on larger data types take multiple clock cycles.

The aggregate processing power of Linedancer depends entirely on parallel processing. A 32-bit add will take many times the number of clock cycles taken by a high-end scalar processor, but due to the parallelism 4096 additions can be performed in parallel. Multiple Linedancer devices can be easily connected together to create an even wider SIMD array.

The Linedancer device (shown in Figure 1) includes an intelligent DMA controller, to ensure that data is moved in and out of the ASProCore concurrently with data processing, and a RISC processor, to issue high level commands to the ASProCore and to setup the DMA controller. All parts of the device run at the same clock frequency, which can be up to 400 MHz. A Linedancer is programmed in an extended version of C, with additional syntax for controlling the ASProCore.

FPGA

Currently FPGAs are used for the colour image processing. FPGA stands for *Field Programmable Gate Array* and denotes an integrated circuit which is programmed in the field as opposed to an *Application Specific IC* (ASIC). ASICs are typical used in high volume quantities because of the high development efforts, costs and manpower involved. As we are not targetting the consumer market, ASICs are not considered in this paper.

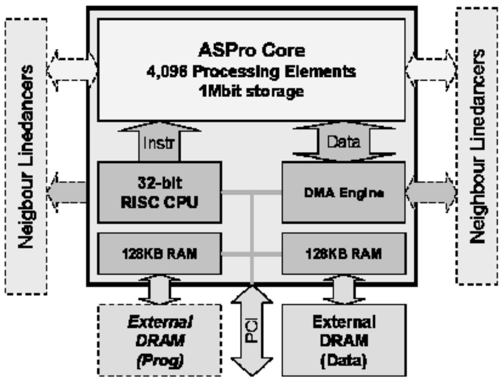

Figure 1. Aspex Semiconductor's Linedancer

From a computation viewpoint the FPGA offers a two dimensional array of configurable logic blocks which are capable of processing a two dimensional image. As the majority of image processing algorithms can be broken down into highly repetitive tasks, FPGAs present a very interesting alternative. An important property of an FPGA is that its throughput is better than the throughput of a von Neumann processor. This can be achieved because the individual logic cells of FPGAs map well to the individual mathematical steps involved in image processing.

FPGAs such as the Xilinx Virtex series [7] provide a large two-dimensional array of logic blocks where each block contains several flip-flops and look-up tables capable of implementing many logic functions. In addition, there are also dedicated resources for multiplication and memories that can be used to further improve the performance.

3. Specification of the Application

Functional process graph

For simplicity, it was decided to restrict the initial research to the most challenging components of the colour pipeline for a 30 pages per minute, 600 dots per inch, full colour printer. A block diagram of this simplified pipeline is shown in Figure 2. Also shown is the amount of communication between the blocks, indicated as bits per pixel.

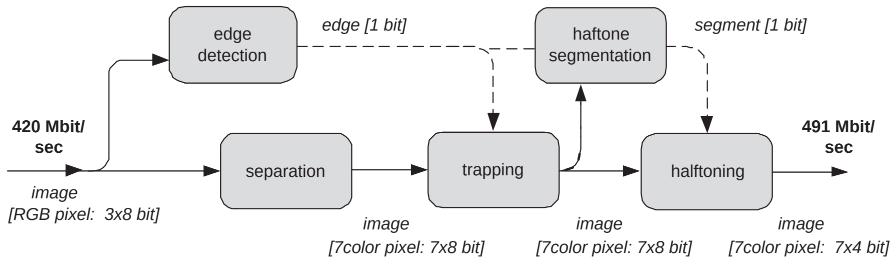

Figure 2. Simplified image processing pipeline

Separation

The Separation stage of the image processing pipeline, also known as colour space conversion, translates the RGB image data into the 7 toner colours that are available in the printer: black (K), blue (B), red (R), green (G), cyan (C), magenta (M) and yellow (Y).

Various algorithms exist for this task, but high quality colour space conversion is a highly non-linear operation [3]. The best results are obtained with a look-up table (LUT) based approach. The look-up table is a large LUT of $2^{24=3\times8}$ entries of 7×8 bit = 940 Mbit in total.

Edge Detection

Edge detection is a neighbourhood operation which determines whether a pixel is an edge pixel or not. The Edge Detection module assists the other modules in making the right choices how to process a particular pixel [2]. The functionality is based on absolute differences in the RGB colour values between each pixel and its immediate neighbours (in a 3×3 kernel). In general such a neighbourhood operation can be specified by the absolute value of a convolution $\text{edge} = |\text{kernel} \otimes \text{image}|$[1]. For a small and symmetrical 3×3 kernel this convolution may be described by

$$\forall_{i,j \in N\times M} \text{Pixel}\,(i,j) = \sum_{k=-1}^{1} \sum_{l=-1}^{1} \text{kernel}(k,l) \cdot \text{Pixel}\,(i+k, j+l)$$

Figure 3 shows an example of a 3×3 kernel.

-1	-1	-1
-1	8	-1
-1	-1	-1

Figure 3. Sample edge detection kernel

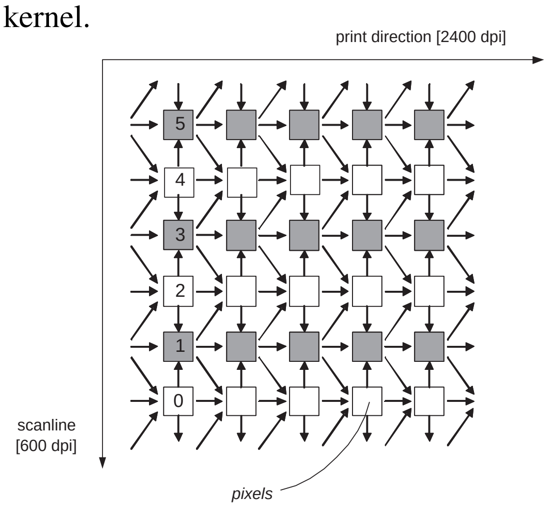

Figure 4. Half-toning error propagation, scanlines arranged horizontally

Trapping

The purpose of trapping is to enhance the print quality by reducing the visibility of small misalignments between the different mechanical components used to print each colour [4].

Trapping decreases the visibility of misalignments by creating an overlap between areas of the different toner colours.

One consequence of the trapping stage is that the imaging pipeline cannot process the 7 colour planes independently. Changes in one colour plane can cause changes in other colour planes.

Half-tone Segmentation

Half-tone segmentation controls how each colour channel has to be half-toned in order to select the best technique for half-toning further down the image pipeline. Half-tone segmentation is another edge detection operation, with many similarities to the Edge Detection stage. The main difference is that it must operate on the 7 colour data output by the trapping stage.

Half-toning

The purpose of half-toning is to render continuous tone information for a print engine, which has a lower tonal resolution than the input bitmaps. 8-bit image data can have 256 different values, but ink is binary – it is either printed or not [3].

[1] $\otimes$ stands for the convolution operator.

Printers overcome this problem by printing at a higher spatial resolution than the input bitmap. In our case, for example, the printer will print in one dimension 4 ink dots (sub-pixels) for each input pixel. A 600×600 dpi image will be printed using 600×2400 dots of each of the 7 colours, per square inch.

Half-toning has to translate 8-bit colour values into 4 ink dots per pixel and does this depending on the pixel being an edge or not. Pixels in a smoothly varying neighbourhood are treated by dithering, a technique which optimises grey level quality at the expense of some spatial resolution. Edges, however, are treated specially in order to retain the sharpness or spatial resolution: they spread the error between the desired colour and the realised colour around the pixels in the very close neighbourhood. See Figure 4 for this required spread of errors: the scanlines are arranged vertically and aggregate errors have to be propagated in horizontal direction.

Performance requirements

The processing of each pixel on the printed page is relatively simple, being mainly based on 3×3 convolution kernels. However, the total image processing pipeline is a challenging task because of the volume of data involved.

All images in the pipeline represent a 7K$\times$5K A4 bitmap page. Every stage must be performed on 35 million pixels and multiple colours. In the two edge detection stages alone there are 350 million (edge detection) operations for every printed A4 page and trapping adds even more.

Many of the tasks in the imaging pipeline can be implemented for many pixels in parallel. The next chapter describes how the image processing pipeline was implemented on the Linedancer parallel processing device.

Table 1 contains the processing requirements for a sequential implementation.

module	number of sequential operations / pixel
separation	3
edge detection	68
trapping	255
half-tone segmentation	30
half-toning	1162

Table 1
Processing requirements

4. Design

Because the size of the whole bitmap is much larger than the available storage space in the Linedancer we have to use a kind of bitmap tiling or patching. For a 4K processor array in which a single Processor Element (PE) deals with a single pixel, a 64×64 tiling scheme is used. For simple point operations each pixel has sufficient data to compute the result. However, for neighbourhood operations as in case of edge detection, the directly involved 8 neighbouring pixels have to be copied to each PE. This can be done for all PEs in parallel. A consequence of this approach is that pixels at the tile's boundary do miss pixel values so can not determine its value. For trapping, for example, only the 62×62 inner pixels can be computed effectively per tile. However, due to the subsequent neighbourhood like computation of half-tone segmentation an extra overlap band is needed, which yields an effective payload of 60×60 inner pixels.

For edge detection, trapping and half-tone segmentation we use the same square patching structure. However, the structure of half-toning breaks up the pipeline and forces us to use a second pass where all pixels are revisited. The intermediate results after half-tone segmentation are dumped into memory and are reloaded again but now using very slim, 1 pixel wide, scanline patches. The reason for this is that the error inputs (see Figure 4) must be added to each pixel in a line before it can be half-toned and its error output can be calculated. Every pixel propagates an error output to 3 neighbours. First the even pixels send an error component to the odd pixels in the current line and the even pixels on the next line. Then the accumulated error of the odd pixels in the current line send errors to the 3 neighbour pixels on the next line.

As mentioned we must process the error propagation one scan line at a time. First doing the calculations for the even pixels, then the odd pixels, while working from left to right.

We can fit an A4 page height (7K pixels) into one Linedancer by packing an odd/even pair of pixels into each processing element; 12% of the PEs remain unused.

This tiling strategy described above, implies a two pass process. All modules except for half-toning are processed in a 64×64 square pixel tile or patch with overlap to accommodate convolution operations like in edge detection. The initial time budget for processing a single patch can be derived by dividing the total number of clocks in the 2 sec/page (2×400 MHz) by the number of patches needed (35 Mpixels / 60×60)[2] yielding a budget of 82K clocks per patch for a single Linedancer.

All modules except for half-toning are run subsequently with a square patch and the intermediate result of pass 1 is dumped to the RAM, see Figure 5.

Then the second pass is started in which complete scanlines are processed in parallel.

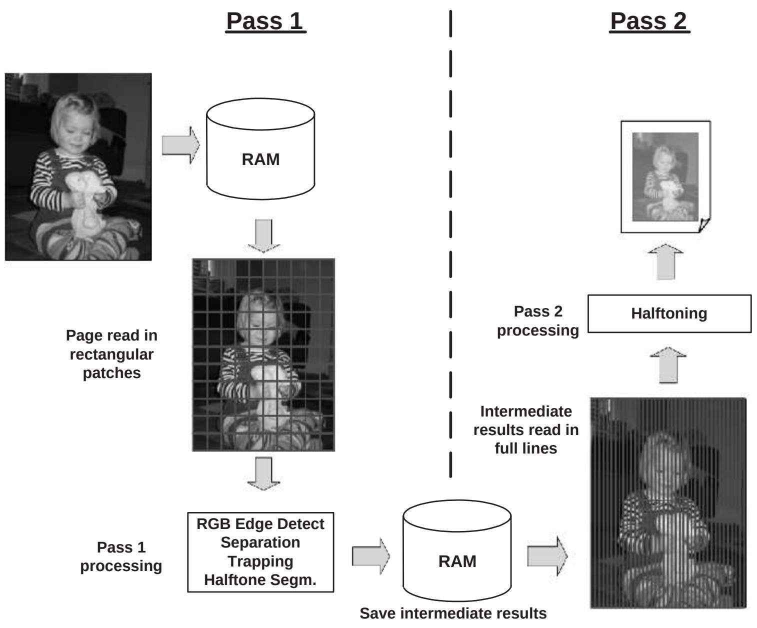

Figure 5. Overview of the 2 pass pipeline

[2]Effective area of a patch is $(64-4) \times (64-4)$ due to the 2×1 pixel overlap at each side.

5. Results and Comparison with FPGAs

In this section the results of the previously elaborated modules are combined in order to formulate a conclusion on the feasibility of functionality and timing of the Linedancer implementation. It serves furthermore as a basis for comparison with FPGA technology.

Pass 1 consisting of loading RGB, edge detection, separation, trapping and half-tone segmentation takes 16K5 cycles per patch. From these modules Separation runs on the DMA controller in parallel with the other modules. Half-toning in pass 2 takes up 2K5 clocks per line per colour (see Table 2).

ASProCore		DMA			
module	number of cycles / patch	module	number of cycles / patch	number of patches / page	number of cycles / page
PASS I					
		load tile	2K		
edge detection	2K2	separation	6K5		
trapping	7K3				
half-tone segmentation	5K				
	16K5 cycles per patch			9K7	161M
PASS II					
half-tone	17K5 cycles per line			5K	87M
TOTAL					248M

Table 2
Performance estimate

The overall processing time is 248M cycles per A4 page, which is equivalent to 0.62 seconds per page, with a single Linedancer at 400 MHz. This is well below the required 2 seconds per page.

Illustrative for the power of massively parallel computing is the speedup compared to the sequential implementation as shown in Table 1. Although the processing capacity of each PE is much lower than a von Neumann processor, the number of processors working in parallel yield large speedups. Parallelism accounts for the speedup although the clock-speed of the Linedancer is lower, see Table 3.

module	number of seq cycles[3]/ pixel	number of cycles / pixel	speedup
separation	3	6K5 / 3600 = 1.81	1.66
edge detection	68	2K2 / 3600 = 0.61	111
trapping	255	7K3 / 3600 = 2.03	126
half-tone segmentation	30	5K / 3600 = 1.39	21.6
half-tone	1162	17K5 / 7K = 2.5	465

Table 3
Measured speedup

The productivity benefits of using an associative processing approach to this problem rather than using FPGA technology are shown in Table 4. The ratio of development effort for an FPGA versus

[3]Based on single cycle operations

Linedancer is estimated at 2:1 (including coding, testing, PCB design etc.), assuming a person with domain and target hardware experience.

technology	development effort [man days]	execution speed [ppm]
2 way SMP Intel Pentium Xeon[4]	10-20	2-30
FPGA Spartan2E[5]	100	30
Linedancer	50	90

Table 4
Comparison

6. Conclusions

An associative processor combines the speed of FPGAs with high-level software programmability and flexibility.

A single Linedancer device is capable of implementing a colour image processing pipeline at a rate of 90 pages per minute, well above the required 30 pages per minute [ppm]. Large speedups can be realised compared to the sequential case: the speedup in operations/sec can go up as far as 400 (up to 80 in execution time).

A key issue in the design is how to partition the 35M pixels of a page into 4K chunks for processing. This apparently simple problem is complicated by the conflicting requirements of the various 3×3 kernel operations and the error propagation in half-toning.

Software defined systems enable fast developments. The development of code for a PC based solution is faster. But when real time performance is critical, and the choice is between FPGA or Linedancer, than the use of the latter reduces the design cycle by a factor of 2.

Due to the inherent scalable architecture the performance can scale with the number of processors without changing the code (e.g. delivering more productivity, more resolution, more colours).

References

[1] Deszo Sima, Terence Fountain, Peter Karsuk: Advanced Computer Architectures: A Design Space Approach. Addison Wesley. ISBN 0201422913. 1997.

[2] Gonzales, Woods: Digital Image Processing (2nd Edition). Prentice Hall. ISBN 0201180758. 2002.

[3] Kang: Color Technology for Electronic Imaging Devices. SPIE-International Society for Optical Engine. ISBN 0819421081. 1997.

[4] Alex Vakulenko: `http://www.oberonplace.com/draw/trapping/`. 1999.

[5] Aspex Semiconductor Ltd: `http://www.aspex-semi.com/products/downloads/aspex_technology_background.pdf`. 2004.

[6] Anargyros Krikelis, Charles C. Weems: Associative processing and processors. IEEE Computer, Volume 27 , Issue 11 (November 1994), Pages: 12 – 17.

[7] Xilinx: `http://www.xilinx.com/products/virtex4/overview.htm`.

[4]Numbers represent normal as well as the optimised case

[5]The system could be build using $\pm$ 5 Spartan XC2S400E devices