Developing Parallel Architectures for Range and Image Sensors

Shaori Guo[+], Wayne Luk[++] and Penelope Probert[+]

[+] Department of Engineering Science, Oxford University, 19 Parks Road, Oxford OX1 3PJ, UK

[++] Computing Laboratory, Oxford University, 11 Keble Road, Oxford OX1 3QD, UK

Abstract

We describe a cost-effective method for developing parallel architectures which increase the performance of range and image sensors. A parametrised edge detector and its systolic implementation using Field-Programmable Gate Arrays (FPGAs) are presented. Experiments and analyses indicate that our circuits can satisfy the performance requirements, and some of the designs out-perform the software equivalent on a 486-based PC by nearly two orders of magnitude.

1 Introduction

This paper describes an approach for developing architectures for range and image sensors, which have applications in industrial inspection and in autonomous vehicles and robots. Our work has been inspired by three developments: the need to include powerful processing in sensing and control systems, the availability of programmable hardware like Field-Programmable Gate Arrays (FPGAs), and the advance in languages and tools for hardware synthesis.

Custom hardware is often used in real-time sensing; for example, Graefe [5] describes a vision system based on custom devices for car navigation. While commercial systems, such as the Datacube vision system, are available for realising more general algorithms, they are usually expensive, difficult to interface and may be too bulky for some applications.

Our work differs from the above in exploiting reconfigurable, off-the-shelf logic circuits as accelerators for low-level processing, with low power consumption and small physical size. In particular, we are interested in low-cost, flexible and efficient implementation of general-purpose image processing algorithms such as median filtering, convolution and edge detection. The rest of this paper shows that, for a number of sensing applications, FPGA-based processors provide a cost-effective implementation with increased flexibility and performance. Moreover, such implementations can be produced rapidly and reliably from parametrised descriptions in an appropriate language.

2 Range Sensors and Feature Extraction

The data used in the experiments described later are obtained from an optoelectronic range sensor [11] designed for vehicle navigation which triangulates a laser spot onto a lateral effect photodiode. The sensor scans in one dimension over ranges between 0.3m and 2.5m, and has been designed to operate at low signal-to-noise ratios. An earlier design consists of a pipeline of three transputers: one to control the hardware and for calibration, one for edge detection and one for tracking features between images. The cycle time of the transputer pipeline is about 400 μs, limiting the point sampling rate of the sensor. We would like to achieve one to two orders of magnitude increase in speed to handle faster changing environments, or to allow us to use a two dimensional scan.

Another sensor that we have used in vehicle guidance is an AMCW optoelectronic sensor [1], which employs a modulation frequency of about 5MHz on a light emitting diode source and detects range through measuring the phase difference between the transmitted and received waves. Adams and Probert [12] show how very dense spatial sampling results in improved reliability in sensing, but this requires faster hardware to achieve sufficient bandwidth even for a one dimensional scan.

Many manufacturing tasks also require high bandwidth processing. A recent application that we have addressed in industrial inspection aims for a resolution of about one square mm over a width of 1m for a production line travelling at 0.5m/s. This is equivalent to a sampling rate of around 1MHz, which cannot be achieved by many conventional low-cost processors.

Although the applications and technologies for sensing differ, the low-level processing usually requires two stages: filtering to reduce noise and feature detection. Common filters for range sensing and for image processing include linear filters, such as Gaussian convolvers, and non-linear filters, such as median filters. Feature detection frequently involves operations like edge detection and segmentation. Since these pixel-based operations are highly parallel, they are promising candidates for FPGA implementation.

3 Sensor Processing Architectures

Many robotics sensing algorithms cannot be implemented in real time on a single microprocessor. While multiple DSP devices or transputer-based parallel architectures can significantly increase the speed of signal processing, they are only effective in exploiting coarse-grain parallelism, when two or more of such processors run in parallel.

Array-based architectures, and systolic array architectures in particular, provide a means of exploiting fine-grain parallelism. There is little control overhead since all data movements are explicitly factored into the design, and often only nearest-neighbour communication is required.

However, the systolic approach has its drawbacks. One major disadvantage is its inflexibility. In many cases, a systolic architecture is restricted to only one kind of computational task. Thus custom systolic hardware may not be cost effective for experimental systems or for small-scale productions.

RAM-based FPGA technology offers an attractive solution: an array of FPGAs, consisted of a large matrix of uncommitted logic gates and registers, can be deployed as a special-purpose processor for many applications. The main advantages of FPGA-based systems can be summarised as follows:

- they enable rapid development since there is no fabrication delay and no need to test for fabrication errors;
- the reconfigurability of FPGAs results in flexibility;
- the fine-grain parallel architecture of FPGAs allows high performance;
- they can be developed at relatively low cost since FPGAs are mass produced, off-the-shelf parts.

FPGA-based prototypes are a fast and cheap alternative to a custom or semi-custom hardware prototype for exploring and verifying designs. A number of hardware accelerators based on FPGAs have been reported recently [4],[9]. Moreover, prototype compilers are available for converting imperative and declarative programs into a format for configuring FPGAs [7]; experimental software for designing array-based circuits, such as numerical and symbolic simulators and floorplanners, have also been developed.

4 A Hardware Edge Detector

4.1 Algorithm

This section illustrates our approach by developing in hardware the edge detector proposed by Marr and Hildreth [10]. The one-dimensional range image $I(x)$ is first convolved with the Laplacian of Gaussian function,

$$\bar{I}(x) = I(x) * (\nabla^2 G(x)) \tag{4.1}$$

where

$$\nabla^2 G(x) = \frac{1}{\pi\sigma^2}(\frac{x^2}{2\sigma^2} - 1)exp(-\frac{x^2}{2\sigma^2}) \tag{4.2}$$

For the discrete case, by choosing an appropriate σ, (4.2) can be transformed to an operator mask. The mask we choose is

$$G = [1 \quad 3 \quad -8 \quad 3 \quad 1] \tag{4.3}$$

which corresponds to computing

$$I_x + 3I_{x-1} - 8I_{x-2} + 3I_{x-3} + I_{x-4} \tag{4.4}$$

Notice that the coefficients of the mask have been simplified to obtain an efficient implementation.

Edges are then extracted by detecting the zero-crossings of $\bar{I}(x)$. A scheme is adopted to reduce the false detection of zero-crossings due to noise: if the magnitude of the convolution at a point is less than a threshold value, we assign zero as the convolution result of this point.

4.2 A Gaussian convolver

Since an FPGA has limited resources, choosing an efficient architecture is important. The symmetry of the coefficients in (4.4) allows us to add I_{x-1} and I_{x-3} before multiplication (see Figure 1), eliminating a multiplier. Moreover the coefficients of the mask, 1, 3 and -8, are chosen so that multiplications can be implemented by shifters and adders. This again reduces cell resources compared with using general-purpose multipliers.

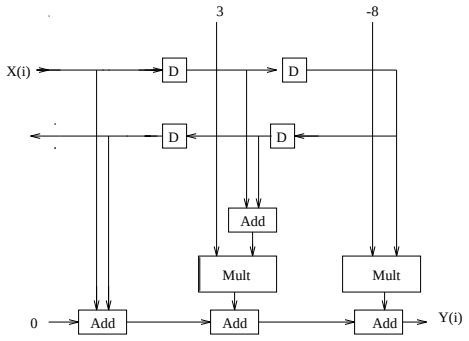

Figure 1 A Gaussian convolver with mask $[1 \quad 3 \quad -8 \quad 3 \quad 1]$. Here D denotes a D latch, Add denotes an adder, Mult denotes a multiplier, X(i) is input and Y(i) is output

4.3 Parametrised Gaussian convolvers

We have chosen very simple coefficients for our mask. For more general cases, suppose the size of the mask is $2n + 1$. Because of the symmetry of the Laplacian of Gaussian convolution, the mask can be described as

$$[w_0 \quad w_1 \quad \ldots \quad w_n \quad \ldots \quad w_1 \quad w_0] \tag{4.5}$$

One possible architecture is shown in Figure 2. It consists of a repeating unit A which, once designed, can be replicated as many times as desired.

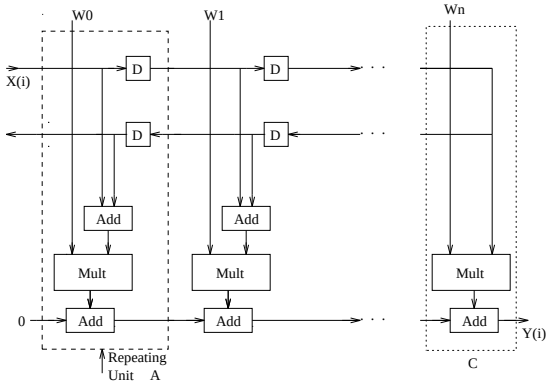

Figure 2 A Gaussian convolver with mask $[w_0 \quad w_1 \quad \ldots \quad w_n \quad \ldots \quad w_1 \quad w_0]$

More importantly, we can exploit this regular structure for parametrising and transforming designs [6],[7],[8], so that optimised designs satisfying specific requirements can be built rapidly and correctly. For example, we can introduce a high degree of pipelining to obtain a design with a short critical path (Figure 3) at the expense of having an extra latency of n clock cycles and n extra registers. Alternatively, we can introduce a lower degree of pipelining which results in the design shown in Figure 4, whose size, critical path and latency are between those of the design in Figure 2 and the one in Figure 3. Indeed, by varying the size of the repeating unit in Figure 4, one can produce designs with a variety of space-time trade-offs.

The above examples can in fact be generated from a single description in the Ruby language [6],[7]. Ruby has

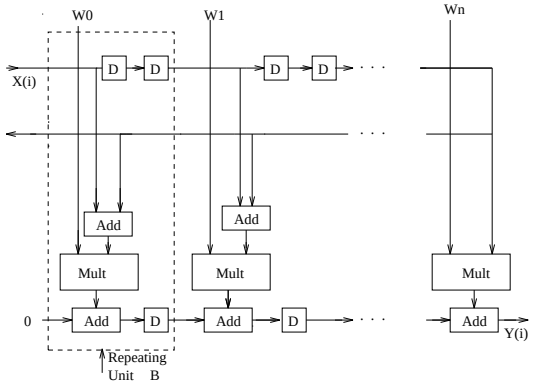

Figure 3 Fully pipelined version of Gaussian convolver

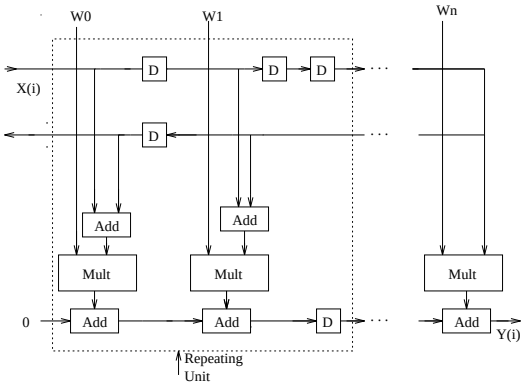

Figure 4 An alternative pipelined Gaussian convolver

various operators for capturing common patterns of computations: for instance $Q \leftrightarrow R$ denotes the composite circuit with Q placed beside R, provided that Q and R have a compatible interface; similarly, $\mathsf{row}_n\, R$ denotes n copies of R lying beside one another. Given that A and B are respectively the repeating units in Figure 2 and Figure 3 and C is the rightmost cell, the Ruby expression

$$(\mathsf{row}_m\,(\mathsf{row}_a\ \mathrm{A} \leftrightarrow \mathsf{row}_b\ \mathrm{B})) \leftrightarrow \mathrm{C}$$

describes the design in Figure 2 when $m = n$, $a = 1$ and $b = 0$. The designs in Figure 3 and Figure 4 can be obtained respectively by having $m = n$, $a = 0$ and $b = 1$ and $m = n/2$, $a = 1$ and $b = 1$.

The characteristics of the designs in Figure 2, Figure 3 and Figure 4 are summarised in Table 1. Note that T_m and T_a are respectively the combinational delays of a multiplier and an adder, and we ignore wire delays in estimating the critical path delays.

4.4 Thresholding and zero-crossing detection

Figure 5 shows the hardware design for thresholding and zero-crossing detection. Zero-crossing detection is implemented by comparing the sign bits of successive convolution results; a zero-crossing is detected when they are different. To reduce the false detection of edges due to noise, we use a thresholding scheme rather than directly comparing the sign bits of the convolution outputs. A comparator is used to compare the absolute value of the convolution result with a threshold value. If the convolution result is less than the threshold, we take it as zero. Otherwise, we compare its sign bit with that of the previous sample (using an exclusive-or gate), and store the current sign bit in a D latch for the next comparison. Hence the output of the comparator controls the updating of the D latch containing the sign bit of the previous sample.

Table 1 Characteristics of the designs in Figure 2, Figure 3 and Figure 4

	Figure 2	Figure 3	Figure 4
number of adders	2n+1	2n+1	2n+1
number of multipliers	n+1	n+1	n+1
Number of latches	2n	3n	2.5n
Latency	2n	3n	2.5n
Critical path delay	(n+2)T_a +T_m	2T_a +T_m	3T_a +T_m

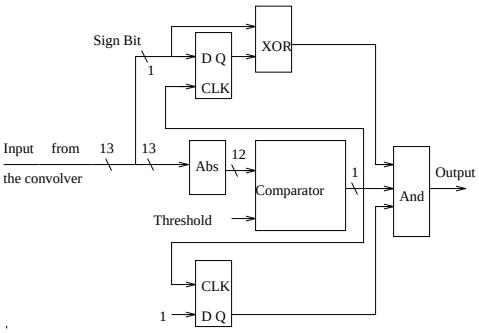

Figure 5 Hardware for thresholding and zero-crossing detection

Note that if the D latch is initialised to zero, then if the first convolution result is negative and its magnitude is larger than the threshold, the sign comparison will become one although no zero-crossing occurs. A similar situation will happen if the D latch is initialised to one. To avoid falsely identifying such cases as zero-crossings, we use another D latch to register the first occurence of the convolution whose magnitude is greater than one; both this and the other D latch will be initialised to zero. Consequently zero-crossing is detected only when the following conditions hold: (1) the absolute value of the convolution – except the first occurence – is greater than the threshold, and (2) the sign bit of the current convolution is different from that of the previous convolution whose magnitude is greater than the threshold.

4.5 FPGA implementations

Our current design and development environment is based on a commercially available system from Algotronix Limited known as CHS2x4 [3]. It is an add-on card for the PC/AT bus, and contains eight user programmable CAL1024 chips. At present, data transfer between the CAL1024 chips and the on-board memory is restricted to sequential input and output over a byte-wide channel, controlled by invoking C-library routines provided by the manufacturer.

CAL arrays are orthogonally connected structures obtained by replicating a basic cell which has an input port and an output port on each of its four sides. An input port of a CAL cell can be programmed to connect to one or more output ports, or to a function unit which can be programmed to behave either as a two-input combinational logic gate or as a latch. See [2] for further details of CAL devices.

The implementation of the edge detector in Figure 1 occupies two CAL1024 chips (Figure 6). This design is obtained from a parametrised description in the OAL language [7], a variant of Ruby specialised for the CAL architecture.

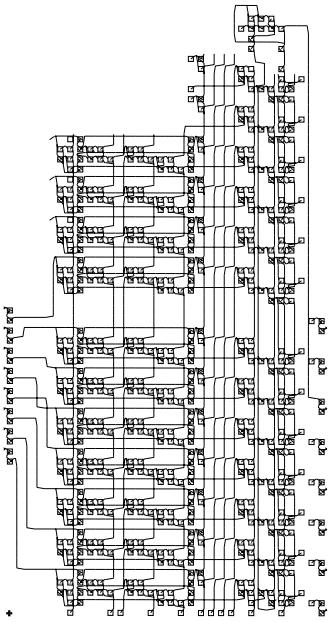

Figure 6 CAL implementation of an edge detector

The performance of the hardware implementations of the edge detector has been evaluated and compared with that implemented in software on a 486-based PC. The processing time required by software and hardware is summarised in Table 2; Figure 7 shows the processing results of our hardware edge detector. The results indicate that our designs detect edges precisely and robustly. Moreover, the CAL measured performance corresponds to a data rate of

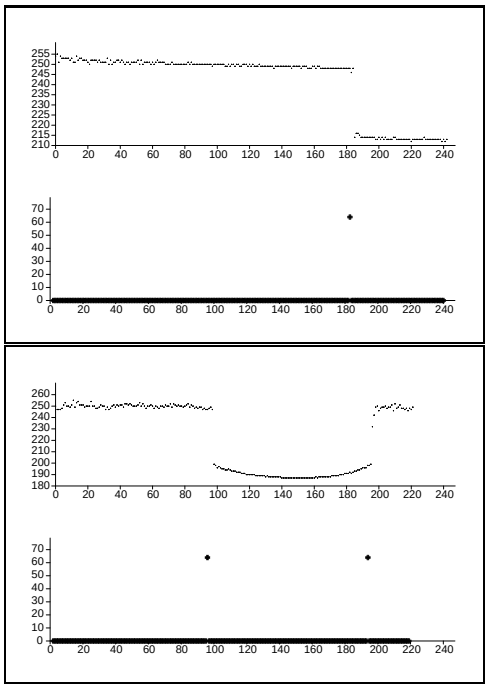

Figure 7 In each figure, the upper part is the raw range data while the lower part is the output of the hardware edge detector implemented using CAL chips

200KHz, which should be adequate for our optoelectronic range sensor mentioned earlier, while the peak performance of the pipelined design reaches 1MHz, which matches the speed requirement for industrial inspection. Better results have been obtained by another edge detector based on median filtering; the features of this design is summarised in Table 3.

Note that the hardware implementation is capable of performing one or two orders of magnitude faster than the software version, although in our present experimental setting the speed of data transfer is limited by the fixed software-controlled routines provided by Algotronix. Moreover, the speed of the hardware can be increased further by bit-level pipelining [6], although the resulting designs will have more registers and larger latencies.

5 Summary

We have described the architectural development and FPGA implementation of an edge detector. Although the architectures of the pipelined convolvers are well known, the use of high-level languages such as Ruby for rapid prototyping using FPGAs is novel. To summarise, the procedure for implementing a specific algorithm is as follows:

- develop a parallel architecture for the given algorithm;
- capture the architecture as a parametrised expression in a language such as Ruby;
- check the correctness of the design by numerical and symbolic simulation;
- apply correctness-preserving transformations to optimise the design to satisfy user constraints, like employing pipelining [6] to increase bandwidth, or serialisation [8] to reduce size;
- develop the most efficient implementation of the repeating units, using device-specific descriptions like OAL if necessary.

Our experience suggests that the reconfigurability of FPGAs offers increased flexibility, and increased performance can be obtained by exploiting fine-grain parallelism. From experimental results and analyses, some of our hardware implementations are capable of achieving a speed-up of one or two orders of magnitude over the equivalent software implementations on a 486-based PC.

We have shown that FPGA-based hardware can be exploited effectively in a systolic mode; some of our other work indicates that data-dependent and irregular operations can also be implemented using FPGAs. Other possibilities include using FPGA-based parallel processing units for data preprocessing of intelligent sensors, such as filtering and edge detection, while high-level processing can be implemented on general-purpose processors, transputers for example.

Although implementing algorithms in hardware is not a new idea, FPGA-based real-time architectures are still at an early stage of development. While others and our work have shown their promising prospect, much remains to be done. Examples include developing more efficient, high-level design tools for FPGA-based platforms, designing general-purpose interfaces between sensors, FPGAs and other devices, and extending our libraries of systolic implementations of signal and image processing algorithms.

Acknowledgements

S. Guo is supported by a Sino-British Friendship Scholarship. ACME Directorate of SERC supported the sensor development. The support of Oxford Parallel Applications Centre, Esprit OMI/HORN project, Scottish Enterprise and Algotronix Limited is gratefully acknowledged.

References

[1] M.D. Adams, *Optical Range Data Analysis for Stable Target Pursuit in Mobile Robotics*, D.Phil. thesis, Oxford University, 1992.

[2] Algotronix Limited, *CAL1024 Datasheet*, 1992.

Table 2 Comparing software and hardware implementations of edge detection based on Gaussian convolution

Number of pixels	Software version	CAL measured performance (unpipelined)	CAL peak performance (unpipelined)	CAL peak performance (pipelined)
10Kbytes	110ms	50ms	20ms	10ms
20Kbytes	220ms	110ms	40ms	20ms

Table 3 Comparing software and hardware implementations of edge detection based on median filtering

Number of pixels	Software version (quicksort)	Software version (Huang)	CAL measured performance (unpipelined)	CAL peak performance (unpipelined)	CAL peak performance (pipelined)
10Kbytes	770ms	440ms	50ms	42ms	6ms
20Kbytes	1540ms	930ms	110ms	84ms	12ms

[3] Algotronix Limited, *CHS2x4 Custom Computer User Manual*, 1992.

[4] F. Furtek, A Field-Programmable Gate Array for systolic computing, in *Research on Integrated Systems: Proc. 1993 Symp.*, G. Borriello and C. Ebeling (editors), MIT Press, 1993, pp. 183–200.

[5] V. Graefe, Dynamic vision systems for autonomous mobile robots, in *Proc. International Conference on Robotics and Systems*, 1989, pp. 12–23.

[6] W. Luk and G. Jones, Parametrised retiming of regular computational arrays, in *Parallel Processing for Computer Vision and Display*, P.M. Dew, R.A. Earnshaw and T.R. Heywood (editors), Addison-Wesley, 1989, pp. 50–63.

[7] W. Luk and I. Page, Parametrising designs for FPGAs, in *FPGAs*, W. Moore and W. Luk (editors), Abingdon EE&CS Books, 1991, pp. 284–295.

[8] W. Luk, Systematic serialisation of array-based architectures, *Integration, the VLSI Journal*, Vol. 14, No. 3, February 1993, pp. 333-360.

[9] W. Luk, V. Lok and I. Page, Hardware acceleration of divide-and-conquer paradigms: a case study, in *Proc. IEEE Workshop on FPGAs for custom computing machines*, D.A. Buell and K.L. Pocek (editors), IEEE Computer Society Press, 1993, pp. 192–201.

[10] D. Marr and E. Hildreth, Theory of edge detection, *Proc. Royal Society of London*, Series B, Vol. 27, 1980, pp. 187–217.

[11] N.E. Pears and P.J. Probert, Active triangulation rangefinder design for mobile robots, in *Proc. IEEE Workshop on Intelligent Robots and Systems*, 1992, pp. 2047–2052.

[12] P.J. Probert and M.D. Adams, Logical sensor development for planning and navigation for a mobile robot, submitted to *IJRR*, 1993.