

Binomial Filters

MATTHEW AUBURY

u91mpa@ecs.ox.ac.uk
Programming Research Group, Oxford University Computing
Laboratory, Wolfson Building, Parks Road, Oxford, England OX1 3QD

WAYNE LUK

w.luk@doc.ic.ac.uk
Department of Computing, Imperial College
180 Queen's Gate, London, England SW7 2BZ

Received November 15, 1994. Revised July 24, 1995.

Abstract. Binomial filters are simple and efficient structures based on the binomial coefficients for implementing Gaussian filtering. They do not require multipliers and can therefore be implemented efficiently in programmable hardware. There are many possible variations of the basic binomial filter structure, and they provide a wide range of space-time trade-offs; a number of these designs have been captured in a parametrised form and their features are compared. This technique can be used for multi-dimensional filtering, provided that the filter is separable. The numerical performance of binomial filters, and their implementation using field-programmable devices for an image processing application, are also discussed.

Keywords: Gaussian filters, binomial filters, parametrised design, field-programmable devices.

1. Introduction

Gaussian filtering is probably the most common form of linear filtering. To overcome the problem of choosing filter coefficients against a set of conflicting constraints, we review an approximation of the Gaussian based on the binomial coefficients which results in a simple, accurate and flexible architecture. This design, called a binomial filter, does not require multiplications, thus allowing large filters to be easily implemented in current programmable hardware technologies, such as Field Programmable Gate Arrays (FPGAs). Moreover, its regular structure facilitates implementation in custom VLSI, and transformations [1],[2],[3] can be applied to produce systolic versions with high performance. It is feasible that such filters can be implemented alongside other hardware, such as programmable DSP chips, at little additional cost.

It is well known that, where applicable, separating a multi-dimensional filter into a cascade of one-dimensional filters significantly reduces the number of computations required to implement the filter [4],[5],[6]. In our case, given that m is the size of the filter and n is the number of dimensions of convolu-

tion, the use of separated binomial methods results in a reduction from m^n multiplications and $m^n - 1$ additions per data point required by conventional methods to $n(m - 1)$ additions and no multiplications. This advantage enables both software and hardware implementations to gain up to two orders of magnitude improvement in speed over conventional methods.

Some of the basic ideas behind the binomial filter are well known and have been discussed by other researchers. For instance, Canny ([6], p. 77) mentioned the binomial approximation of a Gaussian but did not consider the possibility of a parallel implementation, while David et. al. [7] described techniques similar to ours for implementing a given transfer function, rather than studying Gaussian filtering in particular. Chehikian and Crowley [8] and Nishihara [4] discussed binomial approximations of Gaussians and developed various instances of a binomial filter. Wells [5] presented a method of cascading simple filters for computing Gaussian filters and pyramids. The purpose of our work is to provide a coherent account of the numerical properties and architectural variations for the binomial filter, as well as its implementations

in FPGA technology and its application in X-ray image processing.

An overview of this paper is as follows. Section 2 outlines the requirements for a good filter implementation in hardware, while Section 3 introduces the basic binomial filter structure and explores its numerical properties. Section 4 describes various word-level and bit-level architectures for the binomial filter, and outlines how some of these designs can be captured in a parametrised form to facilitate their synthesis. Section 5 covers an example application involving field-programmable devices in an X-ray based defect detection system. Finally, concluding remarks are presented in Section 6.

2. Requirements

One of the most useful sets of linear convolutions is the family of functions based on the Gaussian function. From this function it is possible to derive low-pass, high-pass and band-pass filtering, and many forms of edge detection. The one-dimensional Gaussian function (centred on $r = 0$) is given by:

$$g(r) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{r^2}{2\sigma^2}\right) \quad (1)$$

where σ is the standard deviation. This function possesses several useful qualities. For instance, it is a good low-pass filter since its smooth shape does not lead to ‘ringing’ in the output signal, and because the filter may have a cut-off point determined simply by choice of the standard deviation, σ . The sampled function is then truncated to a fixed number of coefficients, $(2k + 1)$, about a central point. One can rearrange the discrete convolution to give:

$$f(t) * g(t) = \sum_{\tau=-k}^k f(t - \tau)g(\tau)$$

For hardware implementation one often needs to convert g into a function of integers or fixed-point values; this is particularly necessary when field-programmable devices are used, as their silicon resources are limited and floating-point operations are usually too large for them. These conversions naturally introduce deviations from ideal filtering. The factors for accurate filtering can be summarised as follows:

- A sufficient number of coefficients should be used to suppress stop-band ripples. For an 8-bit system, our numerical experiments showed that functions should be truncated at greater than 27σ to obtain a low cut-off frequency.
- A large number of bits should be used to represent the coefficients to maintain a good degree of numerical accuracy.

However, the requirements for an efficient hardware implementation are:

- The number of arithmetic operations, particularly multipliers, should be minimised because they require a large number of gates; this is particularly important for implementations based on field-programmable devices. Hence we should use as few coefficients as possible.
- As few bits as possible should be used, as the number of gates required in data-paths and arithmetic operations is proportional to the number of bits. This implies that we should use as small integer coefficients as possible.
- It would be nice to have the sum of the coefficients to be a power of two, so that bit shifts can be used to normalise the result of a convolution, rather than large division circuits.
- The architecture should be simple and regular to simplify implementation and to achieve high performance.

The result of all of these conflicting requirements is that no perfect solution can be found, and that in general coefficients are hand chosen. However, an interesting method for deriving suitable coefficients can be obtained by considering the properties of a Gaussian.

3. Binomial Filter: Analysis

Let us motivate the development of the basic binomial filter. The central-limit theorem [10] states that, if a function is ‘humped’ at the origin, then repeated convolution with itself causes its shape to tend to that of the Gaussian; in other words, given that

$$f(x) \rightarrow c - dx^2, \quad \text{as } |x| \rightarrow 0,$$

then

$$f_1 * f_2 * \dots * f_n = f^n \rightarrow a \exp\left(-\frac{x^2}{b}\right), \quad \text{as } n \rightarrow \infty$$

Reversing this principle, reasonable approximations to the Gaussian can be obtained by convolving a simple function with itself the desired number of times. If we take a discrete pulse (1,1) which satisfies the requirements of the central-limit theorem and convolve it with itself, we obtain a sequence of coefficient sets:

$$\begin{array}{rcl}
 f^0 & = & 1 \\
 f^1 & = & 1 \quad 1 \\
 f^2 & = & 1 \quad 2 \quad 1 \\
 f^3 & = & 1 \quad 3 \quad 3 \quad 1 \\
 f^4 & = & 1 \quad 4 \quad 6 \quad 4 \quad 1 \\
 f^5 & = & 1 \quad 5 \quad 10 \quad 10 \quad 5 \quad 1 \\
 f^6 & = & 1 \quad 6 \quad 15 \quad 20 \quad 15 \quad 6 \quad 1 \\
 & & \vdots
 \end{array}$$

It is clear that the table above is Pascal's triangle, a tabulation of the binomial coefficients. These form an increasingly good approximation to the Gaussian as more terms are added. Since they are integers, there is no need to worry about rounding errors, and the sum of the coefficients in each set is a power of two (it doubles at each iteration, starting from one). The basic binomial filter structure can be derived from the polynomial

$$(1 + z^{-1})^n$$

which produces the binomial coefficients. This formula can also be regarded as a description of a cascade of hardware units, each containing an adder and a latch – the latter for implementing the z^{-1} term. Figure 1 shows an example of a binomial filter for $n=4$, with coefficients (1,4,6,4,1). In this diagram, as in all others in this paper, latches are shown as triangles with lines through them. Note that this structure is very simple and regular; if it provides a close approximation of the Gaussian, then it would satisfy admirably our requirements outlined earlier.

The quality of the approximation is shown in Figure 2, a plot of the RMS error for each set of coefficients compared against a Gaussian of the same variance; it is clear that the error is reduced to a very small value for large filters. The first zero of such a filter lies at the maximum signal frequency, and as a result there are no pass-band ripples present in the Fourier transform. The absence of truncation effects is shown in Figure 3, which shows the frequency response of f^0 to f^{10} . We can also derive first and second differential coefficients

from the same source, by simply taking the discrete differential of the coefficients already calculated:

$$\begin{array}{rcl}
 \partial(f^0) & = & -1 \quad 1 \\
 \partial(f^1) & = & -1 \quad 0 \quad 1 \\
 \partial(f^2) & = & -1 \quad -1 \quad 1 \quad 1 \\
 \partial(f^3) & = & -1 \quad -2 \quad 0 \quad 2 \quad 1 \\
 \partial(f^4) & = & -1 \quad -3 \quad -2 \quad 2 \quad 3 \quad 1 \\
 \partial(f^5) & = & -1 \quad -4 \quad -5 \quad 0 \quad 5 \quad 4 \quad 1 \\
 & & \vdots
 \end{array}$$

or the second differentials for use in the Laplacian:

$$\begin{array}{rcl}
 \nabla^2(f^0) & = & 1 \quad -2 \quad 1 \\
 \nabla^2(f^1) & = & 1 \quad -1 \quad -1 \quad 1 \\
 \nabla^2(f^2) & = & 1 \quad 0 \quad -2 \quad 0 \quad 1 \\
 \nabla^2(f^3) & = & 1 \quad 1 \quad -2 \quad -2 \quad 1 \quad 1 \\
 \nabla^2(f^4) & = & 1 \quad 2 \quad -1 \quad -4 \quad -1 \quad 2 \quad 1 \\
 & & \vdots
 \end{array}$$

Similarly it is possible to form an approximation starting from a four-point set of coefficients (1, 1, 1, 1). This too satisfies the central-limit theorem, but has two zeroes in the frequency range of the system which leads to ripples in the stop-band. However, as can be seen from Figure 4, repeated convolution with (1, 1) for more than eight times suppresses the stop-band ripples to below the -46 dB dynamic range of an 8-bit system. The advantage of such four-point filters can be seen in Figure 5, which shows the frequency response of two-point and four-point filters for $n = 8$. The cut-off frequency for the four-point filter is considerably lower, without any ripples penetrating the 8-bit dynamic range in the stop-band. Such four-point filters are also normalisable by a power of two, and hence do not lose the benefits of the two-point filters. If very large filter sizes are required, it would become practical to use 8-point filters possessing a low cut-off point. These filters would have to be convolved with (1,1) many times to suppress the stop-band ripples.

4. Architectures for Binomial Filters

The efficiency for a given hardware architecture depends on many factors, including:

- *Size* — the number of gates and latches required to implement a given filter.
- *Speed* — the maximum clock rate for a given circuit, normally as a function of latency and its critical path.

- *Flexibility* — the breadth of choice the architecture gives in terms of choice of coefficients.
- *Routability* — a measure of the number and length of connections between gates and latches required to implement a given filter.

We find that, in general, these parameters can be traded off against one another to obtain a range of designs that achieve a given result.

In the following we shall first describe various word-level and bit-level binomial filter implementations. A comparison of the design trade-offs of different binomial filter architectures and other methods will then be provided. Next, we shall present a concise way of representing binomial filter designs, which also facilitates their automatic implementation in hardware.

4.1. *Binomial Convolver Designs*

Pipelines may be built from the components shown in Figure 6 to perform all of the binomial convolutions discussed so far. Buffered versions of the blocks require more latches but reduce the critical path to an adder or a subtractor. These circuit blocks are ideal for implementing binomial approximation to Gaussian filtering in a very small amount of circuitry.

There are, of course, many further variants on such designs. These variants can be produced by trading off speed against hardware size, or by trading off parallelism in designs using retiming and serialisation techniques discussed in [1], [2] and [3]. These techniques can be applied at both the word-level and the bit-level.

For instance, one can retime the word-level design in Figure 1 such that each repeating unit becomes the one shown in Figure 6(b). Clearly the retimed design can be operated at a higher clock frequency than the original, since the critical path includes one adder delay instead of four. However one has to wait for four clock cycles to obtain the first output, and the additional registers increase the size and power consumption of the design. In reality the degree of pipelining will be controlled carefully in order to obtain the optimal balance between speed, area and power consumption [2].

It is also possible to retime bit-level components. As an example, one stage of a fully-pipelined binomial filter is shown in Figure 7. Another version, pipelined by every two by two bit-level cells, is shown in Figure 8.

If there is insufficient hardware to implement a filter of a given size, serialisation techniques such as LPGS (Locally Parallel Globally Sequential) and LSGP (Locally Sequential Globally Parallel) transformations can be applied to trade off speed against size [3]. Figure 9 shows an LPGS-serial version of the design in Figure 1 that requires only two adders and some multiplexing hardware to control the feedback process. However, the serial implementation adopts a less efficient input/output scheme: an extra cycle must be inserted between successive input samples, and results are produced every other cycle. Again one needs to control the degree of serialisation (probably in conjunction with other techniques such as retiming) to achieve the optimal design for a given application. Other approaches for constructing large Gaussian filters can be found, for instance, in [4].

Nothing has been said so far about normalisation, which is necessary if the output data is to be the same number of bits wide as the input. Each binomial stage increases the size of the data by one bit from input to output. The input data may be normalised by dropping a number of least significant bits, which is likely to introduce very large errors into the output, and in particular may lead to a large steady-state error. The results may be normalised to give the greatest accuracy, but this requires an additional bit in the calculation for every stage of the filter, and introduces some redundant circuitry at the input of the pipeline. Alternatively, each stage may be made normalising by dropping the least significant bit from the result of the addition. This makes each stage only the width of the data, with no redundancy. This leads to slight inaccuracy, as dropping a bit at each stage inherently rounds down the result, leading to a slight loss of amplitude. One solution is to make some stages round up and others round down, which appears to largely compensate for this effect.

We can often separate a multi-dimensional filter into a cascade of one-dimensional filters. This method offers a very simple and cheap way of building multi-dimensional filters. For instance, a two-dimensional convolution can be implemented by convolving the data in one direction, followed by convolving the result in the other direction. It may be necessary to include additional line delays between successive filter stages in one of the directions – an example can be found in [4]. If data are stored in SRAM, one can build address generators to read the image column by column for the vertical convolution.

4.2. Comparison of Convolver Architectures

An estimation of hardware reduction in implementing multi-dimensional convolution using separated binomial methods has been provided in Section 1; Section 5 covers examples of using two-dimensional binomial filters (implemented as cascades of one-dimensional filters) in image processing. Here we compares different methods of implementing multi-dimensional convolution.

As one can observe from the coefficient sets developed earlier, and also because linear-phase finite-impulse response filters are desirable, the coefficients are normally symmetric or anti-symmetric. To this end, an M coefficient convolution requires $(M + 1)/2$ multiplications (Figure 10) [1], [11]. Other architectures exist to improve on any one of the values, but always at the cost of another.

Table 1 summarises the architectures that we discussed. The values listed are for general forms of each convolver. The top four are un-pipelined, using the simplest architectures. For instance, Figure 10 is an example of a symmetric filter which corresponds to the description with $d=1$ and $M=9$ in the third row from the top of Table 1; Figure 1 is an example of a binomial filter described by the fourth row in Table 1 with $d=1$ and $M=5$.

The binomial architecture is clearly superior because it requires no multiplications. As discussed in Section 4.1, when completely unpipelined it can have a longer critical path than some of the other architectures, but this can be controlled by the use of pipelining. When fully pipelined down to the bit-level, it has a very short critical path of a single fulladder, at the expense of an increase in the number of latches and latency cycles. Overall, if the limitation of using only binomial coefficients is acceptable, a multi-dimensional binomial convolver requires significantly less hardware to achieve a particular speed at the expense of multiple memory access. This makes implementations of useful large filters practical in programmable hardware.

4.3. Parametrised Description

Implementation of the architectures introduced earlier has been achieved using hardware compilation techniques to generate circuit netlist information in a device-specific format, such as Xilinx XNF [12], or in

a device-independent form, such as VHDL. The result can then be optimised and transferred to one or more FPGAs.

One way of capturing designs is to use the Rebecca compiler [13] to produce hardware from Ruby, a language of functions and relations. This language is particularly powerful at expressing circuit blocks, and at composing them in various patterns appropriate for the design of systolic arrays. Details about Ruby can be found elsewhere [2],[3],[14]; here we shall introduce the use of Ruby for describing binomial filters.

A design is represented in Ruby by a binary relation of the form $x R y$ where x and y belong respectively to the domain and range of R . For instance, an adder can be described by the relation

$$\langle x, y \rangle \text{ add } (x + y)$$

and a component for replicating its domain datum is given by

$$x \text{ fork } \langle x, x \rangle$$

Ruby has various operators for capturing common patterns of computations: for instance $Q ; R$ denotes the composite circuit with Q connected to R (Figure 11a) provided that Q and R have a compatible interface. Hence

$$x \text{ (fork ; add) } (x + x)$$

Parallel composition, on the other hand, describes a composite design with independent components (Figure 11b). For instance, given $R = \text{fork ; add}$, then

$$\langle x, y \rangle [R, (R ; R)] \langle 2x, 4y \rangle.$$

Similarly, there are constructs in Ruby for iterative structures formed by repeating the binary compositions. Examples include R^n , which denotes n copies of R assembled together to form a cascade (Figure 12a), and $\triangle_n R$, which represents a triangular-shaped array of components (Figure 12c).

To deal with sequential designs, a relation in Ruby can be considered to relate an infinite sequence of data in its domain to another infinite sequence in its range. Hence given that

$$R = (\text{fork ; add}),$$

we have

$$\langle \dots, x_{t-1}, x_t, \dots \rangle R \langle \dots, (2x_{t-1}), (2x_t), \dots \rangle$$

A latch can be described by the relation \mathcal{D} , given by

$$\langle \dots, x_{t-1}, x_t, \dots \rangle \mathcal{D} \langle \dots, x_{t-2}, x_{t-1}, \dots \rangle$$

Formal definitions of these and other components, as well as their algebraic properties, can be found in [2], [3] and [14].

We have used Ruby in representing binomial filter blocks. Given that ι is the identity relation such that $x \iota x$, a parametrised $2k$ -point smoothing filter block, which contains a *fork*, a cascade of k latches and an adder, is given by

$$fda_k = \text{fork} ; [\iota, \mathcal{D}^k] ; \text{add}$$

The simplest convolver block, shown in Figure 6(a), can then be described by fda_1 , while the 4-point filter block shown in Figure 6(e) can be described by fda_2 . The buffered version of fda_k is given by

$$fdab_k = fda_k ; \mathcal{D}$$

so that the designs in Figure 6(b) and (f) are represented by $fdab_1$ and $fdab_2$. Differential blocks can be obtained by replacing the adder *add* with a subtractor *sub*, so

$$fdsb_k = \text{fork} ; [\iota, \mathcal{D}^k] ; \text{sub} ; \mathcal{D}$$

A complete filter can be built using these blocks. For instance, given $k = 1$ and $n = 4$,

$$bf_{n,k} = (fdab_k)^n ; fdsb_k$$

produces a pipelined first-differential binomial filter with coefficients

$$(-1, -3, -2, 2, 3, 1).$$

Ruby descriptions shown above can be simulated, analysed and compiled into hardware using the Rebecca system [13].

We have also used the OAL language, a version of Ruby specialised for the Algotronix (now Xilinx) FPGA, to develop bit-level designs [15]. Each CAL cell has an input and an output at each of its four sides, and a function block in the centre which can implement a designated two-input logic function or operate as a latch. An input can be programmed to connect to one or more output ports or to the function unit. A CAL cell can hence be used to perform processing and routing simultaneously. Figure 13 shows a CAL cell with its northerly output connected to its easterly input, and its easterly output, emerging from the centre,

is the Boolean conjunction of its westerly and northerly inputs.

Using this architecture, we generate a fulladder in a 3×2 array of cells, and the *fork* and delay required by the binomial structure can be included using a further 1×2 array. Input and carry pipelining requires 1×2 and 3×1 arrays of cells respectively. An example of a filter constructed using such blocks, is shown in Figure 14. This is a partially-pipelined, four-stage, 24-bit filter, with the data flowing from right to left. It is implemented in an array of 20×49 CAL cells, with the bottom row providing zeroes at the input of the carry circuits.

5. Performance Assessment

A potential industrial application for large size, fast Gaussian filtering is in X-ray based defect detection for food and other production line items. At present the system, used by one of our industrial sponsors, deals with X-ray images delivered to a dedicated image processing board, Sharp GPB-1, by an image intensifier / CCD camera or a solid state detector system.

There are two types of defects which are difficult to detect: small high-density defects such as metal, and large low-density defects such as plastic. These defects show up respectively as small dark points and regions of low intensity, on a background of non-uniform product variation, packaging and noise. The best processing technique was found empirically to be a form of band-pass filtering, followed by thresholding and some binary morphology. The band-pass filter isolates low-density defects in the manner of a second differential edge detector, and high-density defects as sharp variations from the local mean.

The band-pass filtering is done by differencing a slightly smoothed image with a heavily smoothed image, the heavily smoothed image being formed by seven convolutions of a 3×3 smoothing filter. The resulting filter size is 15×15 . The impulse response of the original Gaussian-based filter, and that of the separated binomial filter designed using MATLAB, are shown in Figure 15. The result of applying the new filter to a typical X-Ray image is shown in Figure 16.

A prototype version has been implemented on the HARP system [16], which integrates a T805 transputer, 4Mbyte DRAM, a Xilinx 3195 FPGA chip with two local banks of 32K by 16-bit SRAM, and a 100MHz variable clock frequency synthesizer on an industry-standard TRAM module. The speed of the FPGA

depends on the critical path of the logic that it implements, and it can be varied using the frequency synthesizer. Our current experimental setup deals with real-time video rather than X-ray images: the HARP board takes its input from a CCD camera and sends its result to a video display. Using this system, we have been able to demonstrate that our binomial filter implementation is capable of running at video rate.

Several binomial filters have also been implemented using the CHS2×4 FPGA-based system [17]. This system contains eight CAL1024 FPGA chips arranged in a two by four array, giving 8192 user-programmable cells. A 13×13 pixel Gaussian smoothing filter has been implemented in a PC containing a CHS2×4 system as unseparated software, separated software, binomial software and binomial hardware to compare performance and check the results of the different implementations. The software is written in Microsoft C and timed on an Intel 486DX33 based computer.

The speed of the different implementations is shown in Figure 17, excluding the downloading and uploading times for the hardware implementation. It is notable that the 13×13 filter used by no means represents the limit of the CHS board; it is perfectly capable of implementing filters upto 50×50 pixels. This design is partially pipelined, having a critical path of around 200 nanoseconds, giving a maximum running speed of 5 MHz. This is far above the actual clocking speed of approximately 180 KHz, which is due to the way the board is clocked in software by the host PC. An external clocking scheme would thus allow a speed-up factor of around thirty over the timings shown.

The filter given can also be quite easily fully pipelined, giving a maximum speed of 30 to 50(+) MHz, though at the cost of increasing latency. This would give a factor of around 150 increase in speed over the times shown. The graph of Figure 18 shows these projected speed increases against the Sharp VLSI image processing system. If in future a CAL FPGA is capable of accommodating an 128×128 array of logic blocks, one can then implement on a single chip a fully pipelined, 42-bit 26×26 filter at a pixel rate of the order of 30 to 100 Mpixels/s, with enhanced accuracy by adopting fixed-point number representation. This is one to two orders of magnitude improvement on the performance of current VLSI image processing systems such as the Sharp GPB-1, and it comes with a reduction in size and an increase in flexibility. For instance, for the X-ray application it may be pos-

sible to dynamically reconfigure the FPGAs to carry out band-pass filtering, thresholding and morphological operations on the same hardware.

6. Concluding Remarks

To summarise, there are two main achievements of this work. First, we review a binomial approximation to the Gaussian, and show how this architecture offers a simple, accurate and readily extensible solution satisfying the requirements described in Section 2. The use of binomial filters provides a convolution method which has been shown to require considerably fewer arithmetic operations, by virtue of not requiring multiplications. The resulting architecture is compact and efficient, and is readily parametrisable for a range of trade-offs in time, space and numerical accuracy.

Second, these filters have been implemented both in software and in several FPGA-based platforms. For instance, several Ruby descriptions have been implemented in Algotronix CAL FPGAs, and the results of the implementations have been verified against the same algorithm in software. Timing of the hardware shows that, even when being clocked at around 180kHz, a speed far below its maximum capability, it still out-performs software running on a reasonably powerful desktop computer. Our experiments confirm that the increase in speed achieved by using separated filters, binomial coefficients and the efficient binomial architecture can be several orders of magnitude over conventional techniques, combined with a dramatic reduction in hardware size.

There are several areas for further research, some of which are in progress. One of our current programmable hardware platforms, the CHS2×4 [17], operates in a stand-alone format, with long delays in the downloading of images for processing and in retrieving the results. These delays often offset the advantage gained from the efficient filtering technique. In order for the filters to be of practical use, the programmable hardware should be integrated into the system where it is to be used, for example as a programmable coprocessor for existing image processing hardware. Although this has been achieved to some extent in our HARP system [16] for video-rate image processing, further work is required to provide a solution for systems involving stored data.

At present generating these filters for the targets shown in this paper requires an in-depth understanding of a wide variety of tools. We are exploring the

development of design aids which, given a user specification of a filter, will automate the parametrisation, code generation and hardware compilation steps. It is also desirable for such tools to support data partitioning [18], hardware/software codesign [19] and, where appropriate, exploitation of dynamic reconfiguration of reprogrammable FPGAs. This would make the design system very flexible and appealing to non-specialists and experts alike, particularly if an integrated system, as described in the preceding paragraph, is available.

The main application for the filters described here has been presented as image processing, but they are likely to have considerable uses elsewhere. High speed linear noise removal for real-time sensors [11] and short-term memory elements for temporal neural networks [20], are two other examples of areas in which these filters can offer a significant advantage over conventional methods.

Acknowledgements

Thanks to Jim Crowley, Keith Nishihara, Dick Shoup and members of the Oxford Hardware Compilation Research Group for discussions and suggestions, and to the two reviewers who provided useful comments. The support of Imperial College Department of Computing, the ESPRIT OMI/HORN (P7249) project, Oxford Parallel Applications Centre, Scottish Enterprise and Xilinx Development Corporation is gratefully acknowledged.

References

1. S.Y. Kung, *VLSI Array Processors*, Prentice Hall, 1988.
2. W. Luk, "Systematic pipelining of processor arrays," in G.M. Megson, ed., *Transformational Approaches to Systolic Design*, Chapman and Hall Parallel and Distributed Computing Series, 1994, pp. 77–98.
3. W. Luk, "Systematic serialisation of array-based architectures," *Integration*, vol. 14, no. 3, 1993, pp. 333–360.
4. H.K. Nishihara, *Real-Time Implementation of a Sign-Correlation Algorithm for Image-Matching*, Technical Report No. TR-90-02, Teleos Research, 1990.
5. W.M. Wells, "Efficient synthesis of Gaussian filters by cascaded uniform filters," *IEEE Trans. PAMI*, vol. 8, no. 2, March 1992, pp. 234–239.
6. J. Canny, *Finding Edges and Lines in Images*, M.Sc. Thesis, Massachusetts Institute of Technology, 1983.
7. D. David, T. Court, J.L. Jacquot and A. Pirson, "INP 20: an image neighborhood processor for large kernels," *Proc. IAPR Workshop on Computer Vision: Special Hardware and Industrial Applications*, University of Tokyo, 1988, pp. 241–244.
8. A. Chehikian and J.L. Crowley, "Fast computation of optimal semi-octave pyramids," 7th S.C.I.A., Aalborg, August 1991.
9. H.K. Nishihara, *System for Expedited Computation of Laplacian and Gaussian Filters and Correlation of Their Outputs for Image Processing*, United States Patent 5119444, June 1992.
10. R.N. Bracewell, *The Fourier Transform and its Applications*, McGraw-Hill International, 1986.
11. S. Guo, W. Luk and P. Probert, "Developing parallel architectures for range and image sensors," *Proc. IEEE International Conference on Robotics and Automation*, 1994, pp. 2205–2210.
12. Xilinx Inc., *The Programmable Logic Data Book*, 1994.
13. S. Guo and W. Luk, "Compiling Ruby into FPGAs," to appear in W. Luk and W.R. Moore, eds., *Proc. FPL95*, Lecture Notes in Computer Science, Springer Verlag, 1995.
14. G. Jones and M. Sheeran, "Circuit design in Ruby," in J. Staunstrup, ed., *Formal Methods for VLSI Design*, North-Holland, 1990, pp. 13–70.
15. W. Luk and I. Page, "Parameterising designs for FPGAs," in W.R. Moore and W. Luk, eds., *FPGAs*, Abingdon EE&CS Books, 1991, pp. 284–295.
16. A. Lawrence, A. Kay, W. Luk, T. Nomura and I. Page, "Using reconfigurable hardware to speed up product development and performance," to appear in W. Luk and W.R. Moore, eds., *Proc. FPL95*, Lecture Notes in Computer Science, Springer Verlag, 1995.
17. Algotronix Ltd, *CHS2x4 Custom Computer User Manual*, 1992.
18. W. Luk, T. Wu and I. Page, "Hardware-software codesign of multidimensional algorithms," in D.A. Buell and K.L. Pocek, eds., *Proc. IEEE Workshop on FPGAs for Custom Computing Machines*, IEEE Computer Society Press, 1994, pp. 82–90.
19. W. Luk and T. Wu, "Towards a declarative framework for hardware-software codesign," in *Proc. Third International Workshop on Hardware/Software Codesign*, IEEE Computer Society Press, 1994, pp. 181–188.
20. M.C. Mozer, *Neural Network Architectures for Temporal Sequence Processing, Predicting the Future and Understanding the Past*, Addison Wesley, 1993.

Matthew Aubury is a fourth year undergraduate at Oxford University studying Engineering and Computing Science. He is currently working within the Hardware Compilation Group of the Computing Laboratory on the tools used in software to hardware translation.

Wayne Luk studied Engineering and Computing Science at Oxford University and has been a faculty member there. He is currently Governors' Lecturer at the Department of Computing, Imperial College, University of London. His research interests include theory and practice of hardware compilation and hardware/software codesign.

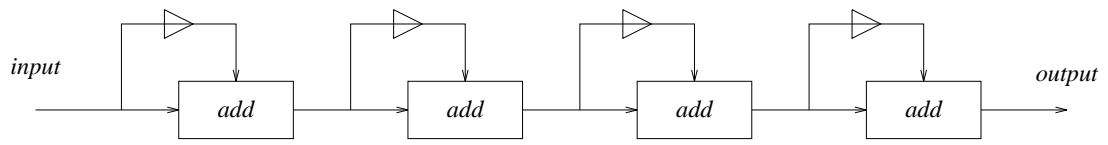


Fig. 1. Basic binomial filter structure. The triangular blocks represent latches.

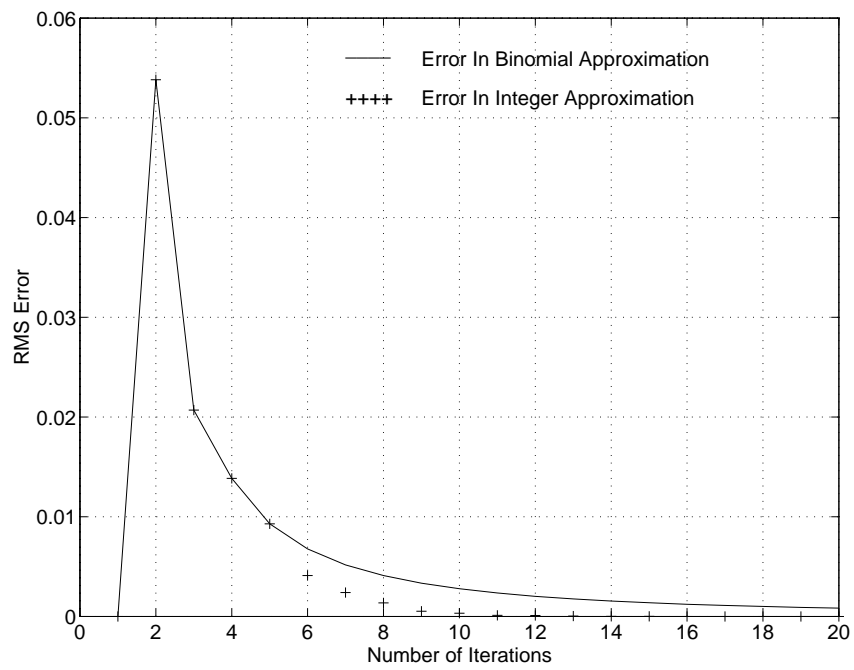


Fig. 2. RMS error for binomial approximation to Gaussian.

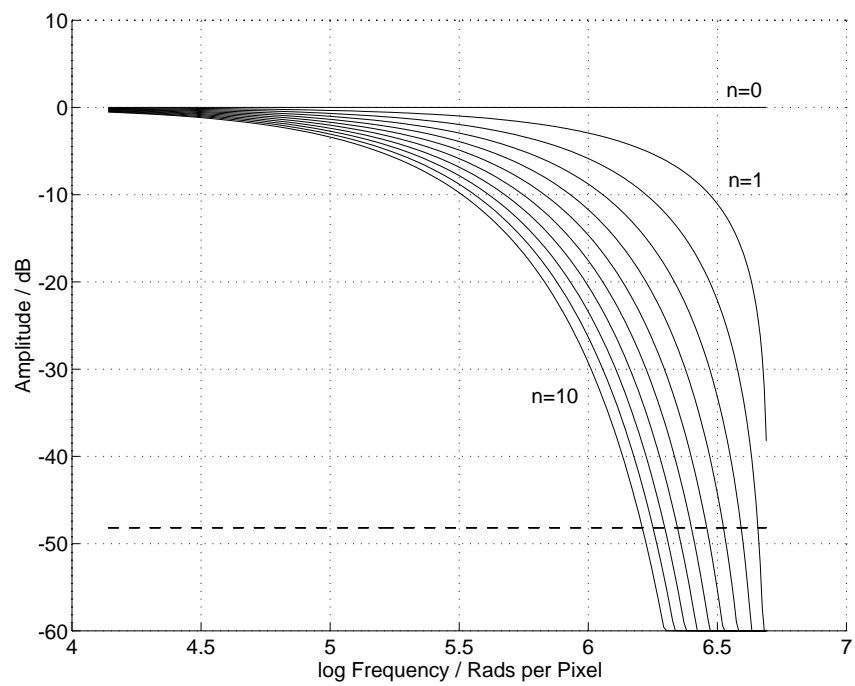


Fig. 3. Frequency response of $(1, 1)^n$ filter for $0 \leq n \leq 10$. The dotted line indicates the -46dB dynamic range of an 8-bit system.

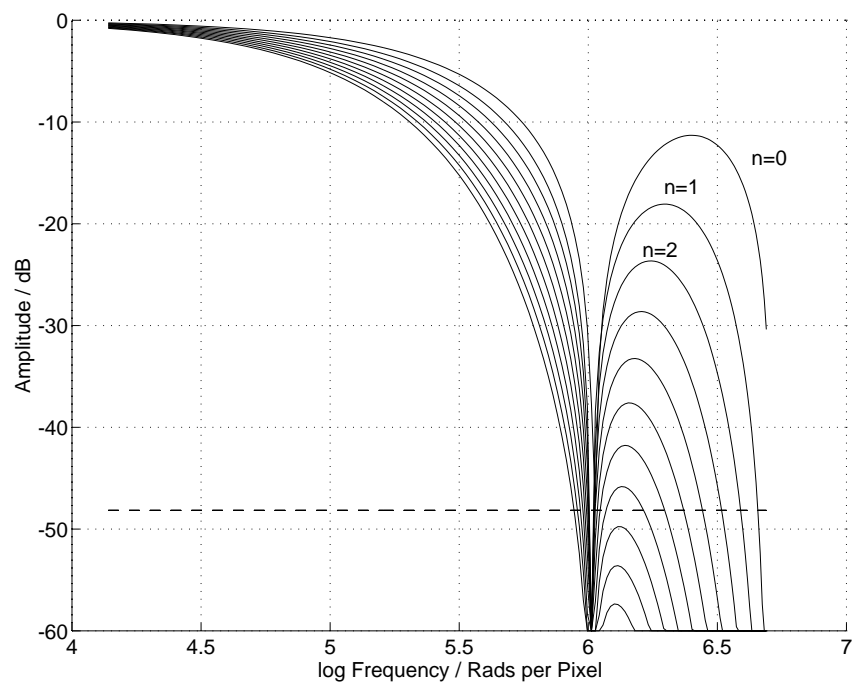


Fig. 4. Frequency response of $(1, 1, 1, 1)(1, 1)^n$ filter for $0 \leq n \leq 10$.

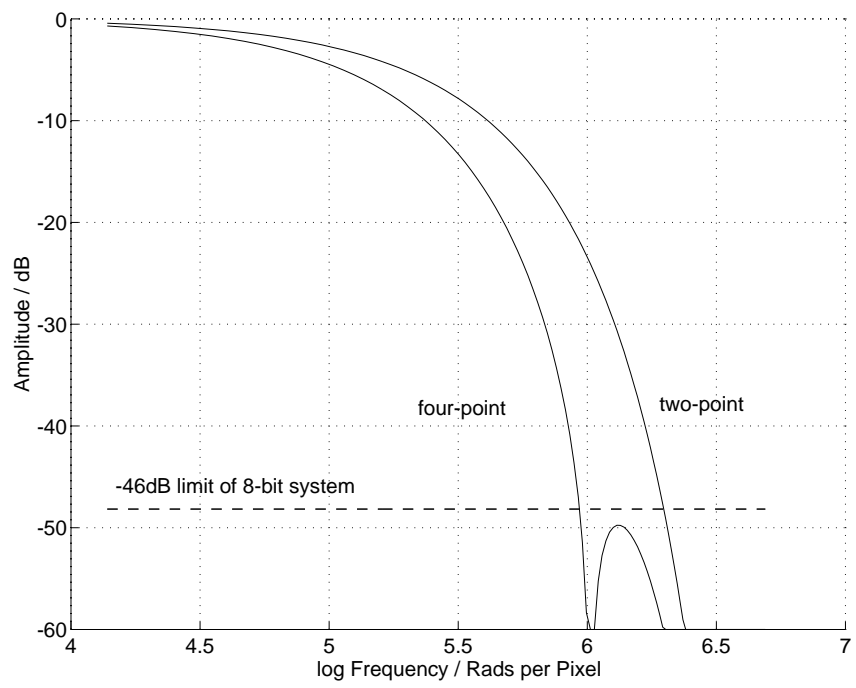
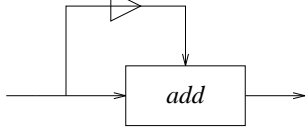
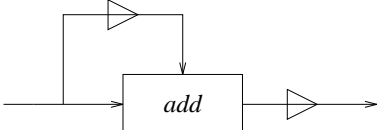
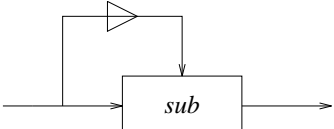
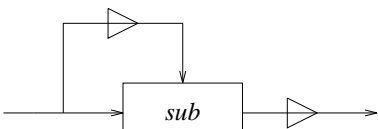
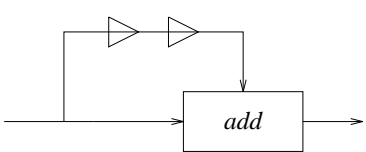
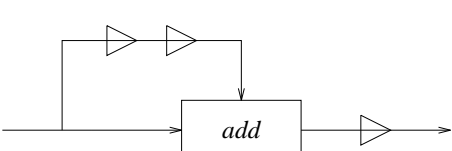


Fig. 5. Comparison of two-point and four-point filters for $n = 8$.

Schematic	Hardware required	Description
(a) 	1 adder 1 latch	2-point smoothing
(b) 	1 adder 2 latches	2-point buffered smoothing
(c) 	1 subtractor 1 latch	2-point differential
(d) 	1 subtractor 2 latches	2-point buffered differential
(e) 	1 adder 2 latches	Separated 4-point smoothing*
(f) 	1 adder 3 latches	Separated 4-point buffered smoothing

* Gives (1, 1, 1, 1) coefficients when combined with a single two-point smoothing filter

Fig. 6. Circuit blocks for constructing binomial filters. As in Figure 1, the triangular blocks represent latches.

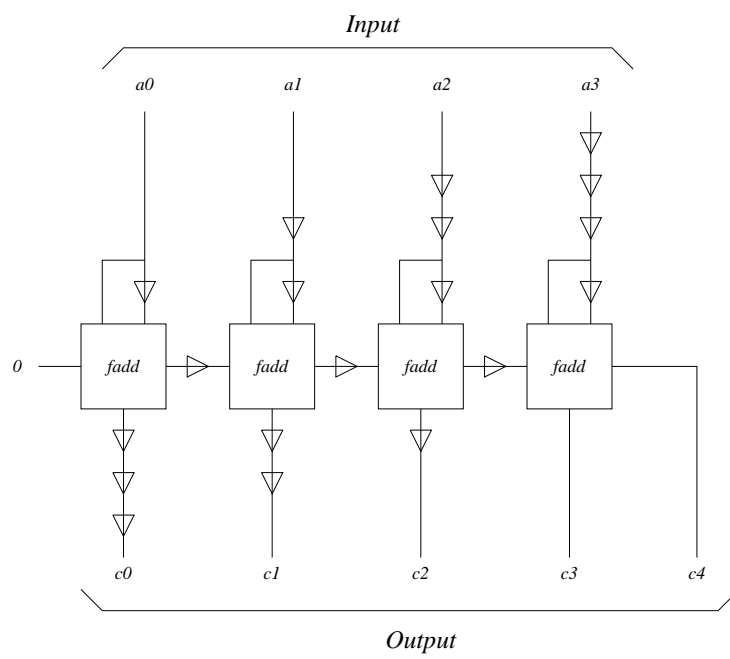


Fig. 7. A fully-pipelined bit-level binomial filter stage. *fadd* denotes a fulladder.

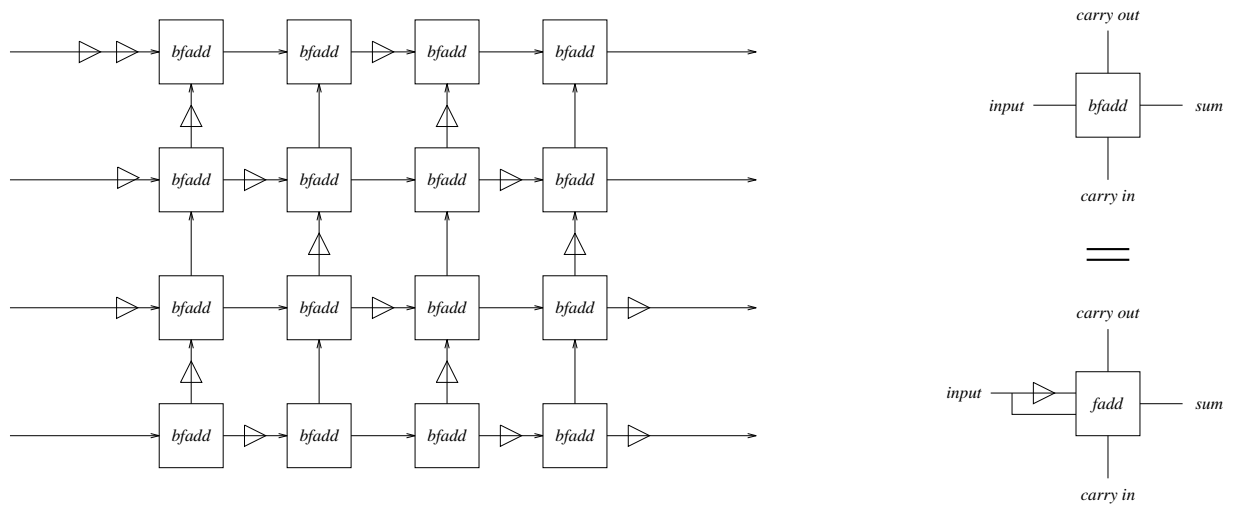


Fig. 8. A binomial filter pipelined by every 2×2 *bfadd* cells, each of which denotes a fulladder and a latch (see legend on the right of the diagram), with its horizontal inputs derived from the sum output of the *bfadd* cell on its left.

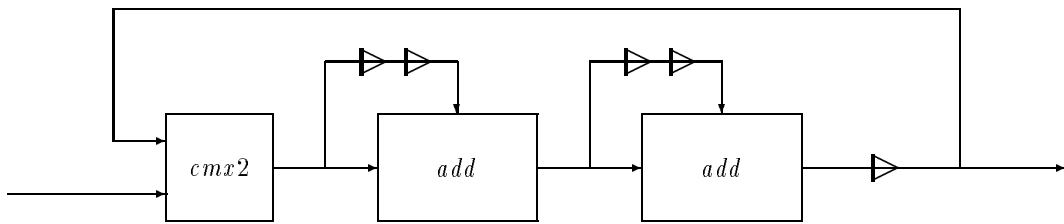


Fig. 9. A serial binomial filter for the coefficients (1,4,6,4,1). The component *cmx2* is a multiplexer that alternately connects its output to its bottom input and to its top input at successive clock cycles.

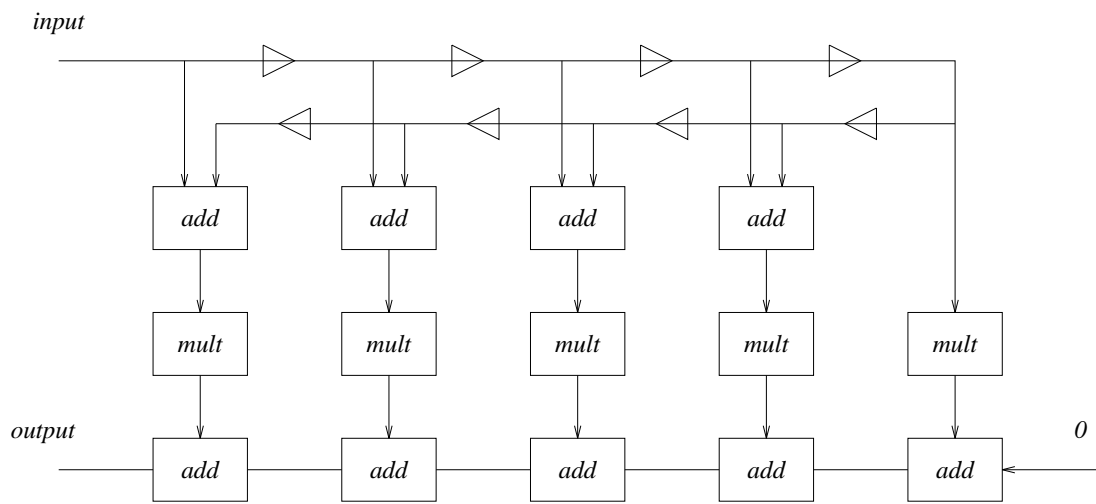
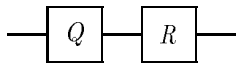
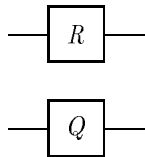


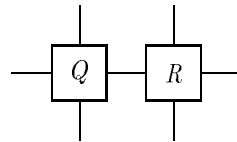
Fig. 10. Linear-phase filter (multiplication coefficients are not shown) – not a binomial filter.



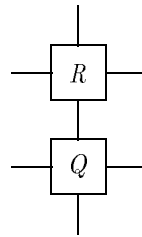
a. $Q ; R$



b. $[Q, R]$

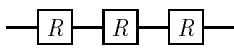


c. $Q \leftrightarrow R$

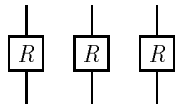


d. $Q \updownarrow R$

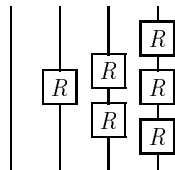
Fig. 11. Binary compositions.



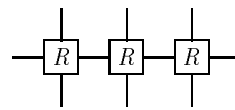
a. R^3



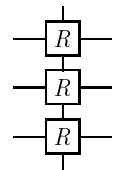
b. $\text{map}_3 R$



c. $\Delta_3 R$



d. $\text{row}_3 R$



e. $\text{col}_3 R$

Fig. 12. Repeated compositions.

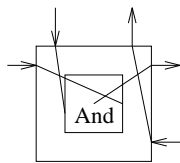


Fig. 13. An example CAL cell.

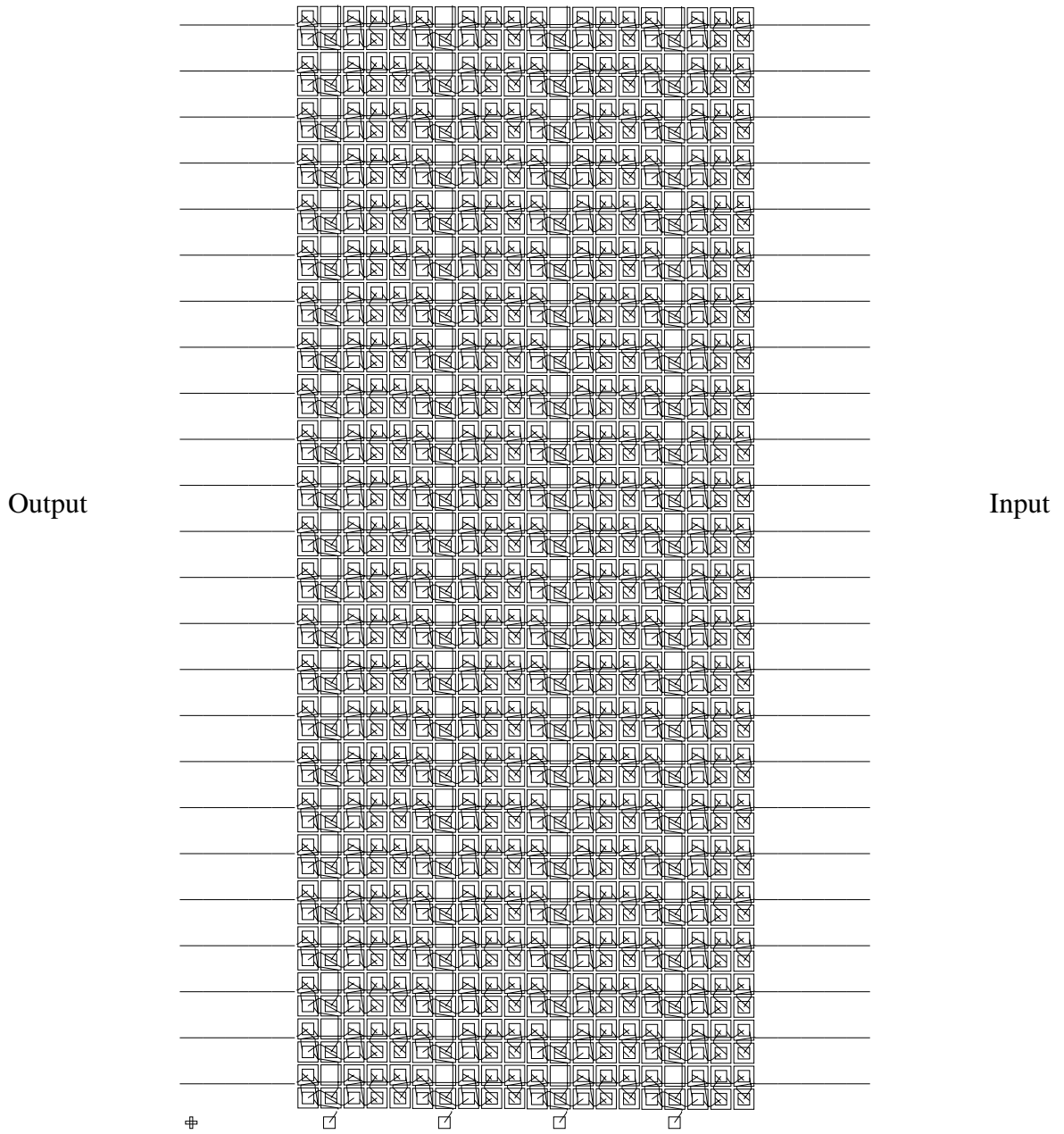


Fig. 14. Binomial filter implemented in CAL cells.

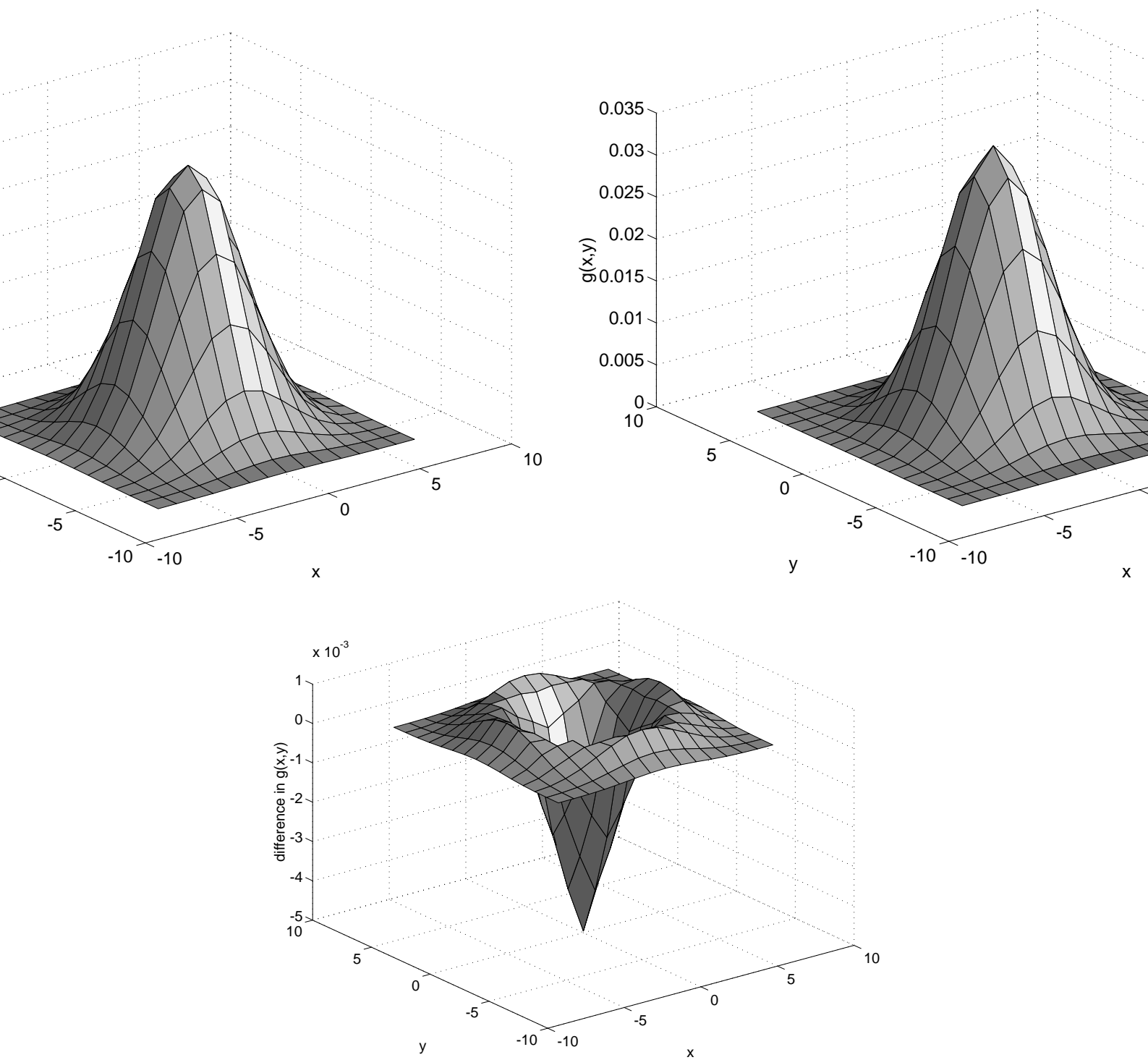
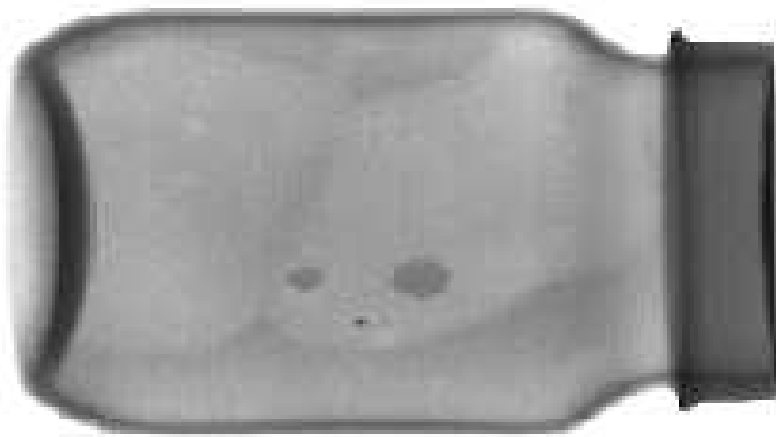
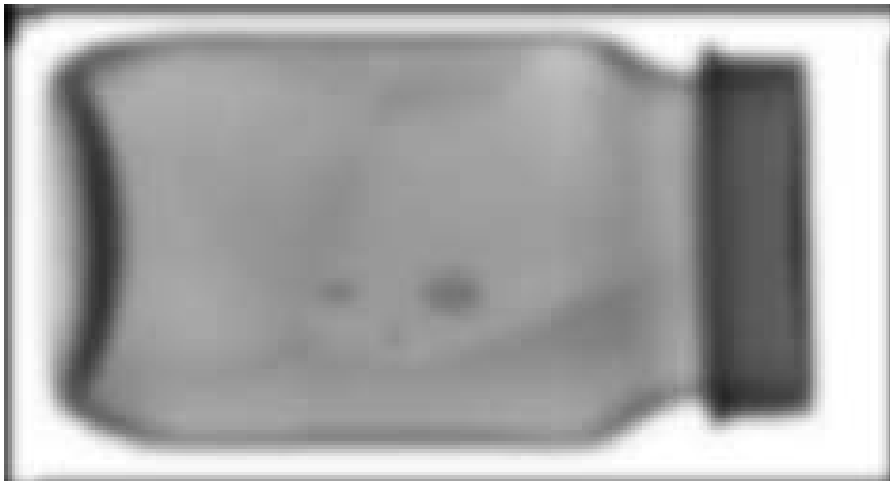


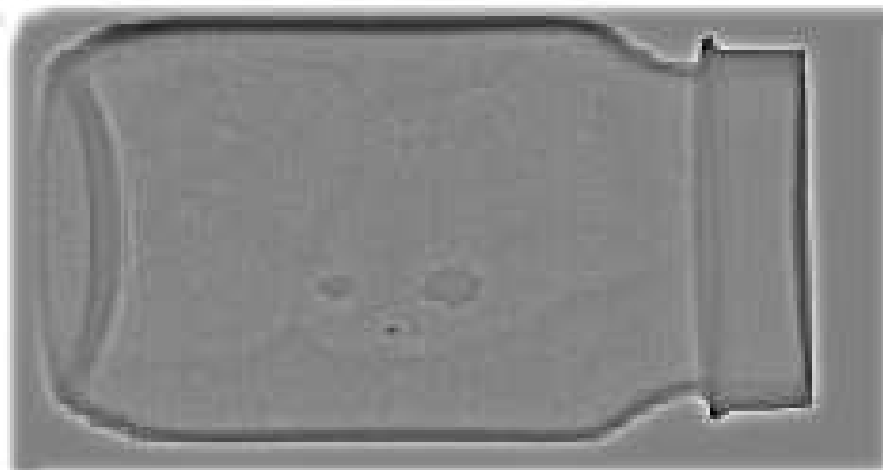
Fig. 15. Left: original filter impulse response; right: response of separated binomial filter; Below: error between filters



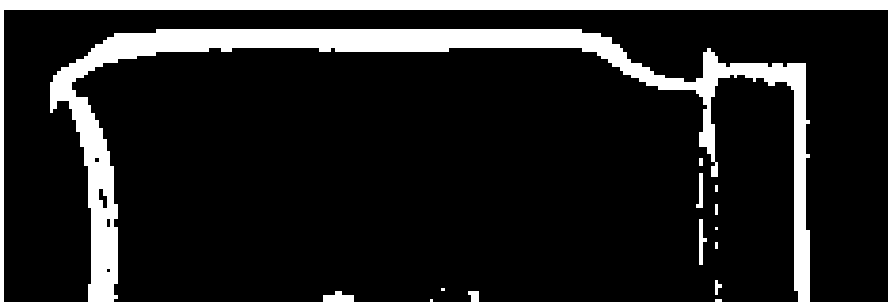
Original Image



Smoothed Image



Difference Image



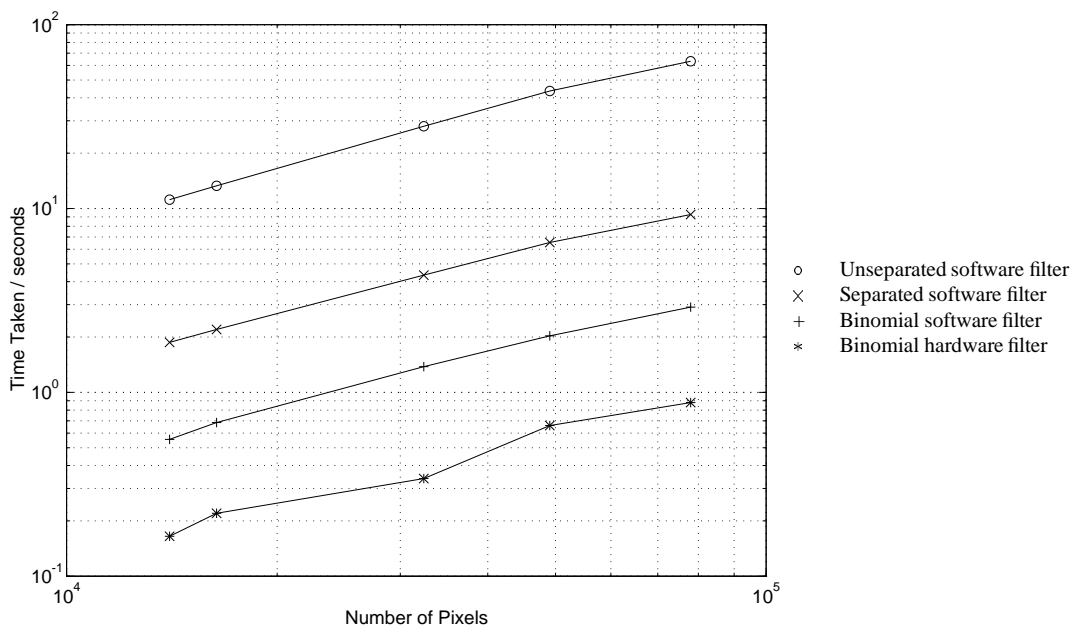


Fig. 17. Performance comparison of various hardware and software filters.

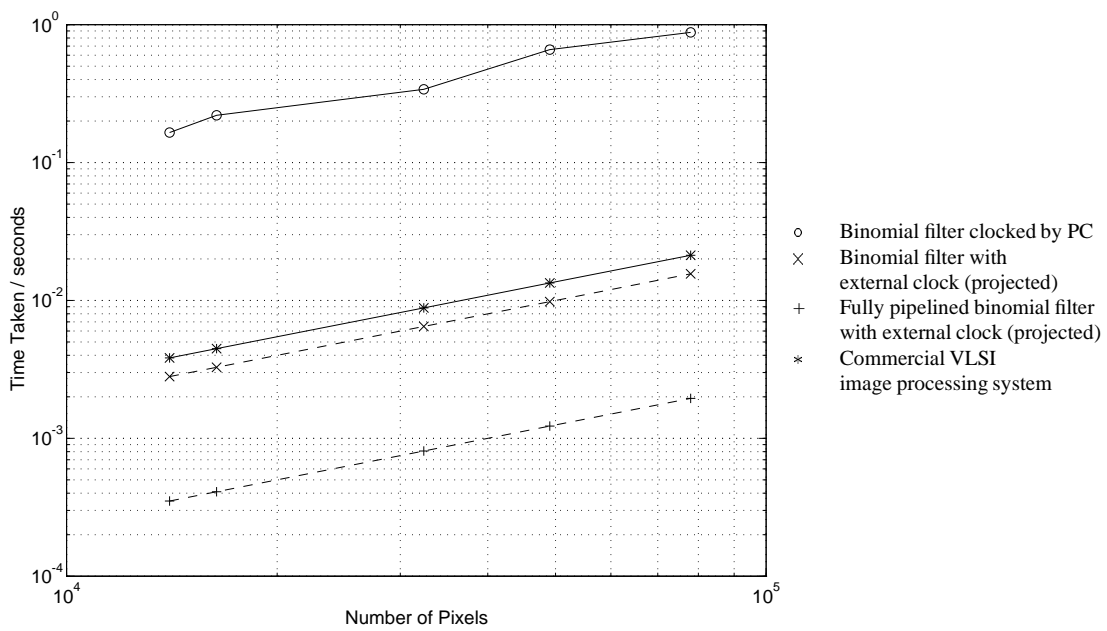


Fig. 18. Performance comparison of various filter implementations.

Architecture	Memory accesses	N -bit multipliers	N -bit adders	Latches	Latency	Critical path
Unseparated	1	M^d	$M^d - 1$	—	—	—
General separated	d	M	$M - 1$	MN	$\frac{M-1}{2}$	$\tau_{mult} + (M-1)\tau_{add}$
Symmetric separated	d	$\frac{M+1}{2}$	M	$(M-1)N$	$\frac{M-1}{2}$	$\tau_{mult} + (M+1)\tau_{add}/2$
Binomial separated	d	0	$M - 1$	$(M-1)N$	$\frac{M-1}{2}$	$(M-1)\tau_{add}$
Word-level pipelined binomial	d	0	$M - 1$	$2(M-1)N$	$\frac{3(M-1)}{2}$	τ_{add}
Fully pipelined binomial	d	0	$M - 1$	$\approx 2MN + N^2$	$\frac{3M+2N-7}{2}$	τ_{fadd}

Notes:

- M is the number of coefficients used in the convolution.
- N is the number of bits used to compute the convolution, and is assumed constant (that is, each computational element normalises its output).
- d is the number of dimensions of convolution.
- τ_{add} , τ_{mult} and τ_{fadd} are respectively the combinational delay of an adder, a multiplier and a fulladder. Wire delays are ignored in the estimation of critical path.
- Latency is defined to be the number of cycles between a number being input and its reaching the centre of a set of coefficients.
- For the unseparated filter, the number of latches and the latency is dependent upon the dimensions of the input data.
- The exact number of latches for a fully pipelined binomial architecture is $N^2 + 2MN - M - 4N + 1$.

Table 1. Comparison of architectures.