J. Indian Inst. Sci., Mar.–Apr. 1991, **71**, 159–170.
©Indian Institute of Science.

A parallel fast convolver for computer vision

K. RAMANI, S. SRIKANTA, Y. V. VENKATESH AND J. W. RAYMOND
Computer Vision and AI Laboratory, Department of Electrical Engineering, Indian Institute of Science, Bangalore 560 012, India.

Received on August 13, 1990; Revised on March 12, 1991.

Abstract

We report here the design and fabrication of a fast convolver board, compatible with the I/0 slots of the PC/AT, for the basic mathematical operation of 2-dimensional convolution of images. Extension of the hardware design to perform convolution in parallel is described. The basic hardware is designed and fabricated using locally available components.

Key words: 2-D Signal analysis, convolution, image processing, low-level vision, parallel computation and table look-up.

1. Introduction

Convolution is one of the basic mathematical operations encountered in 1- and 2-D signal processing. Signal conditioning, filtering and feature extraction are some of the operations that involve convolution. In digital image processing, discrete convolution is performed, during early stages (low-level processing), to enhance images and to extract features such as edges and texture boundaries.

Convolution is computation intensive. But, digital convolution can be speeded up by making use of the finite nature of digital images. Further, it will be desirable to perform convolution using hardware, because considerable amount of time can be saved in performing the low-level operations. The fast convolution method is inherently amenable to parallel implementation. Even the serial version of the method gives an order of magnitude speed-up. The hardware is designed to perform fast convolution in parallel, using multiple add-on boards to a PC/AT. This can facilitate some of the computer vision applications, like the robotic vision system, to perform tasks in near real-time.

In this paper, we present an efficient approach to the realization of the fast convolution, and give the hardware details of its implementation. It is believed that the proposed design can be extended to realize a multiprocessor architecture in VLSI for the real-time processing of vision data.

The mathematics of convolution is given in Section 2, followed by a brief reference to the relevant literature in Section 3. A comparison of the direct implementation with the fast method of discrete convolution is presented in Section 4. Section 5 gives the details of the parallel implementation, and Section 6, the hardware features. Finally, possible future extensions are suggested and some concluding remarks made in Section 7.

2. Discrete convolution

The 2-dimensional convolution operation is represented by

$$g(x,y)=\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} h(\alpha,\beta)\cdot f(x-\alpha,y-\beta)\,\mathrm{d}\alpha\,\mathrm{d}\beta \tag{1}$$

where f is the input signal, h the weighting function and g the convolved output.

The discrete version of this is the summation:

$$g(k,l)=\sum_{i=-M/2}^{M/2}\sum_{j=-M/2}^{M/2} m(i,j)\cdot f(k-i,l-j),\quad k,l=0\ldots N-1 \tag{2}$$

where f is the input image, m the mask, and g the output image. The input and output images are of size $N\times N$ (indexed $0\ldots N-1$) and the mask is of size $M\times M$ (indexed $0\ldots M-1$). Typical values of N and M are, 512 and 11. The images invariably have 8 bits per pixel. To provide motivation for the fast method, some details of the actual computations involved in convolution are necessary. These are given below:

The convolved signal is computed by superimposing the mask (an array containing the weighting coefficients, $m(i,j)$s in eqn (2)), the centre of the mask being aligned with the image point $f(k,l)$, and multiplying the values of the corresponding points in the image and the mask. The mask is usually a square array, with dimensions invariably being odd. The sum of the partial products, thus computed, is the output $g(k,l)$. The mask is moved over the image, in both the row and column directions, for computing the output at each pixel. It can be verified that for an image of dimension $N\times N$, and a mask of dimension $M\times M$, the direct implementation of the convolution entails $N^2\times M^2$ multiplications. For example, if $N=512$ and $M=5$, then the number of multiplications is $512\times 512\times 5\times 5=6\,553\,600$.

3. Results in the literature

Literature abounds in results on the implementation of convolution on various computers. They, however, amount merely to the division of a given image into sub-images, and a processor is assigned to each sub-image. A majority of the publications deals with systolic architecture for convolution. For instance, Kung and Song[1] present an algorithm on a machine comprising bit-serial basic cells interconnected to form a 2-D systolic array. Kung *et al*[2] use a systolic array, with pipelined arithmetic units as a building block, for 1-dimensional convolution. The systolic array (for multidimensional convolution) also uses a second level pipelining by allowing the processing elements themselves to be pipelined to an arbitrary degree.

Lee and Aggarwal[3] present a parallel convolution scheme, using a 1-dimensional systolic structure as a basic unit, for a mesh-connected array processor consisting of the same number of simple processing elements as the number of pixels in the image. The number of computation steps is equal to the number of coefficients of the convolution window. The computation is carried out along the so-called Hamiltonian path ending at the centre of the window, the length of which is equal to the number of window coefficients. The authors claim that the architecture and the control strategy make the scheme suitable for VLSI implementation.

Maresca and Li[4] present a generalized convolution algorithm for mesh-connected computers through a snake-sweeping mechanism. Chang *et al*[5] discuss the use of a pyramid computer of $O(n^2)$ processors for convolution in $O(\log n + k^2)$ time. Ranka and Sahni[6] also present an algorithm for a mesh-connected array, but with data movement less than that of the other algorithms and without broadcasting data (window) values. Giordano *et al*[7] suggest a semi-systolic architecture (comprising programmable VLSI components) for a high-speed pyramidal convolver which allows parallel computation to be carried out at different resolutions. Fang and Ni[8] deal with parallel algorithms for convolution on existing architectures.

4. Proposed fast convolution

The finite nature of the digital signals can be exploited to reduce the number of multiplications to a considerable extent. To this end, note that the image gray level is limited to a value in the integer range $0 \ldots 255$.

During the entire convolution operation each pixel value is multiplied by a particular mask value exactly once. That means, for one mask value, there are $N \times N$ multiplications. However, for the total $N \times N$ multiplications there are only 256 distinct products, because the image gray level is restricted to a value in the range $0 \ldots 255$. In other words, there are many redundant products, calculated during the course of the convolution. This redundancy is exploited to minimize the total number of multiplications.

4.1. *Product Look-Up Table (LUT)*

Let us now consider the convolution operation with a slight difference in the order in which the mask values are multiplied with the pixel values. Since each pixel value is multiplied exactly once with each mask value, superimposition of the mask, as obtained from the convolution definition, over the image may be dispensed with in the following manner. The operation can be performed by multiplying the image with one mask value at a time, to get an $N \times N$ array of partial product. In other words, the convolution is performed in $M \times M$ steps, each corresponding to getting the partial product for a particular mask value. These partial products are finally added, with appropriate offset indexing, to get the convolved image.

It is not necessary to perform multiplication at every pixel directly. A look-up table can be set to have all the possible products for the particular mask value, indexed by the pixel value. For a 256-level image the LUT will have 256 entries. Now, multiplication is simplified to a table look-up. The product is obtained by reading the input image and using the value as index in the LUT to get the product. With these simplified operations one 'PASS' is defined as:

1. Set the LUT for the particular mask value.
2. Read the input buffer.
3. Index the LUT with the value read.
4. Add the product read from the LUT to the output buffer.
5. Repeat steps 2, 3 and 4 until all points in the image are read.

One pass results in an output array of partial products corresponding to a particular mask value. And, one pass is independent of another if a different output buffer is maintained for each pass.

Then the partial products, calculated for every mask value, are added 'appropriately' to get the correct convolved output, as described below.

4.2. *Addition of partial products with offsets*

The summation of the partial products should be done by maintaining the neighborhood order with which the mask values are multiplied with the pixels. This can be illustrated with the help of Fig. 1. The partial product arrays corresponding to the two mask values, $m(-1,-1)$ and $m(-1,0)$, with the entire image are shown below.

According to direct evaluation, $p1(0,0)$ should be added to $p2(0,1)$ as they correspond to the horizontally adjacent values in the mask. Moreover, this sum should go to the output location $(1,1)$, because it corresponds to the image location over which the centre of the mask is placed. It is evident that addition of the partial products has to be done with suitable offsets computed as a function of the mask value indices and the image size.

Instead of obtaining the partial products separately, and then adding them to get the final result, the operation of getting the sum, as and when the partial products are calculated, can be performed using the offsets. For this, the output buffer, having the same size as that of the input image, is initialized to zero. The output buffer is modified during every 'pass'

$$f \times m(-1,-1),\ m(-1,-1) = -1 \qquad\qquad f \times m(-1,0),\ m(-1,0) = -2$$

$$p1 = \begin{array}{|c|c|c|c|c|c|}\hline -1 & -1 & -2 & -2 & -2 & -2 \\ \hline -1 & 0 & -2 & -2 & -2 & -2 \\ \hline -4 & -4 & -3 & -3 & -3 & -2 \\ \hline -4 & -5 & -3 & -3 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 \\ \hline \end{array} \qquad p2 = \begin{array}{|c|c|c|c|c|c|}\hline -2 & -2 & -4 & -4 & -4 & -4 \\ \hline -2 & 0 & -2 & -4 & -4 & -4 \\ \hline -8 & -8 & -6 & -6 & -6 & -4 \\ \hline -8 & -10 & -6 & -6 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 \\ \hline \end{array}$$

FIG. 1. Partial product computation.

in the following manner. Read the input image, pass it through the LUT to get the product, and add the partial product to the contents of the output buffer location offset by the indices of the mask value location, with respect to its centre.

For mask value $m(-1, -1)$, the row- and column-offsets are 1. The output from the look-up table is added to the output buffer contents at $(1, 1)$. This can be done by using the address (indices) generated for reading the input as the reference, and calculating the output address (indices) by adding the row- and column-offsets. The addition is done using Modulo N arithmetic to take care of the image size limits. During the operation of one pass, the offsets remain the same. So, for every pass, the look-up table has to be modified to hold the correct products, and the offsets have to be calculated for the particular mask value location. Since the partial products are added with suitable offsets to the same output buffer, there is no requirement of additional storage for the partial products.

4.3. *Generation of row- and column-offsets*

As mentioned earlier, the row- and column-offsets are obtained as a function of the mask indices and the image size. The expressions for the row- and column-offsets are as given below:

$$\text{Image\_size} = N \times N \quad \text{Mask\_size} = M \times M$$

Column-offset

```
for $I := 0$ to Mask_size/2 do
    column-offset [$I$]: = (Mask_size/2) − $I$;

for $I := $ (Mask_size/2) + 1 to (Mask_size − 1)
    column-offset [$I$]: = Image_size − (Mask_size − $I$);
```

Row-offset

```
for $I := 0$ to Mask_size/2 do
    row-offset [$I$]: = (Mask_size/2) − $I$;

for $I := $ (Mask_size/2) + 1 to (Mask_size − 1)
    row-offset [$I$]: = Image_size − (Mask_size − $I$);
```

In this method, the border pixels are wrapped around in both X and Y directions. For example, let $N = 128$ and $M = 5$. Consider the mask value $m(-2, -2)$. Row- and column-offsets are 2 and 2, respectively.

Input address			*Output address*
0, 0	Row:	$(0 + 2)$ mod 128	2, 2
	Column:	$(0 + 2)$ mod 128	
127, 10	Row:	$(127 + 2)$ mod 128	1, 12
	Column:	$(10 + 2)$ mod 128	
127, 127	Row:	$(127 + 2)$ mod 128	1, 1
	Column:	$(127 + 2)$ mod 128	

5. Parallel implementation

As described in Section 2.2, the convolution of the image of size $N \times N$, with a mask of size $M \times M$, is complete after $M \times M$ passes.

It can be seen that the outcome of one pass does not depend on the outcome of the other, at any stage. That is, if multiple passes, corresponding to different mask elements, are carried out in parallel, and the respective outputs are written to different output buffers, then the final summation can be performed as the last operation in the sequence.

Consider one unit, containing an input buffer, a LUT, an output buffer and the timing logic, that performs one pass. Using such units the convolution can be completed using full or partial parallel implementation.

If only one such unit is used to do the operation, in M^2 sequential passes, then the resultant structure is a serial fast convolver. When we have M^2 units performing M^2 passes in parallel, each unit doing a pass corresponding to a mask value, the convolution is completed in one parallel pass. This leads to a full parallel structure for the implementation of the fast convolution method (M^2-parallel scheme). However, K units, where K is a factor of M^2, can be made to proceed in parallel (K-parallel scheme). But each unit will perform a sequence of M^2/K passes, the resulting partial products being accumulated in the K output buffers. Then the partial products, calculated for every mask value, obtained using M^2- or K-parallel scheme, are added appropriately to get the correct convolved output.

5.1. *Speed-up factor*

It is easily seen that the number of multiplications is reduced to a total of $M \times M \times 256$. That is, the look-up table is initialized for each mask value, and in the process 256 multiplications are performed. For the entire operation the look-up table is set $M \times M$ times, as the number of passes equals the number of mask elements. So, the number of multiplications for convolving an image of size $N \times N$ with a mask of size $M \times M$ is independent of the size of the image. Moreover, the whole process is now transformed into simple memory accesses with appropriate addressing.

The execution times for different schemes are explained by the following analysis.

Let the image be of size $N \times N$ and 8 bits/pixel, and mask be of size $M \times M$. Let t_m be the time taken for one multiplication. Define one memory access as the sequence: reading input buffer, reading the LUT, and reading and writing the output buffer. Let t_a be the time taken for one such memory access.

Direct method

Number of multiplications: $N^2 \times M^2$

Number of memory accesses: $N^2 \times M^2$

Time taken to complete convolution: $N^2 \times M^2 \times (t_m + t_a)$

(neglecting the time taken to add the partial products).

Table I
Comparative analysis of the different schemes

Scheme	Time	Speed-up factor
Direct	$N^2 * M^2 * (t_m + t_a)$	1
Serial	$N^2 * M^2 * (t_a + 256 * t_m / N^2)$	$\frac{t_m + t_a}{t_a}$ (*N* large)
M^2-parallel	$256 * M^2 * t_m + N^2 * t_a$	$M^2 \frac{t_m + t_a}{t_a}$ (*N* large)
K-parallel	$N^2 * M^2 * ((256 * t_m / N^2) + (t_a / K))$	$K \frac{t_m + t_a}{t_a}$ (*N* large)

Similarly, the time and speed-up factors are computed for the other schemes (Table I).

The software implementation (serial scheme) of the LUT-based convolution was compared with the direct method of convolution. The speed-up is about 2.2. The small value is due to the fact that $t_a > t_m$, where t_a includes the time taken for instruction decoding and address calculation. So the improvement in the execution time is only marginal. But, in the direct hardware implementation t_a can be reduced to such an extent that $t_a < t_m$, and very high speed-up factors can be obtained.

If all the M^2 passes are performed in parallel, then a speed-up of approximately M^2 times the value for serial scheme can be obtained. This will make some of the low-level tasks performable in near real-time. However, it is possible to have *K* passes proceeding in parallel, and if *K* is a factor of M^2, the convolution can be completed with a speed-up of approximately *K* times that of the serial scheme.

6. Hardware features

The hardware prototype of the fast convolver has been fabricated. Only one unit has been built, so the convolution can be done only by sequential passes. The board is designed to be compatible with the I/O extension slot of the PC/AT[9]. (The board was fabricated using off-the-shelf components.) Essentially, it consists of fast memory chips, adders, counters, latches, monostable multivibrators and other logic chips. A single chip processor, Intel MCS-48[10], coordinates the internal operations and the communication with the host machine (PC/AT). The design is straightaway extendable to parallel operation by duplicating the hardware boards.

If the same board is used for parallel implementation, with *K* passes in parallel, then there will be *K* input and output buffers. Each board will accumulate results corresponding to (M^2/K) passes. The sum can be obtained as a last step by reading the contents of the *K* output buffers and adding them.

6.1. *Hardware description*

Refer to the block diagram (Fig. 2) of the convolver. The input buffer is 256 × 256, 8-bit wide. This amounts to a memory of size 65536 bytes. Two 32 K × 8 chips are used to form the 64 K-byte input buffer. The output buffer is 256 × 256, 16-bit wide. Four 32 K × 8 chips are used for realizing the output buffer. The input and output buffers are addressed using 16 lines. The LUT, implemented on an 8 K × 8 chip, has 256 locations, each 16-bit wide. In this case, 8 address lines are required to address the LUT. 16-Bit addition is performed using four 74LS283 4-bit adders[11]. The clock rate is 1.25 MHz, and hence the clock period is 800 ns. Address is generated using four 74LS161 4-bit preset counters. As described in offset address generation section, the output buffer is addressed using another 16-bit preset counter, which is driven by the same clock that drives the input address generator. The terminal count of the input address generator marks the end of one pass (Fig. 4).

6.2. *Timing considerations*

Addressing of the buffers and the LUT is done by the counters. Intermediate signals for latches and read signals are generated using monostable multivibrators.

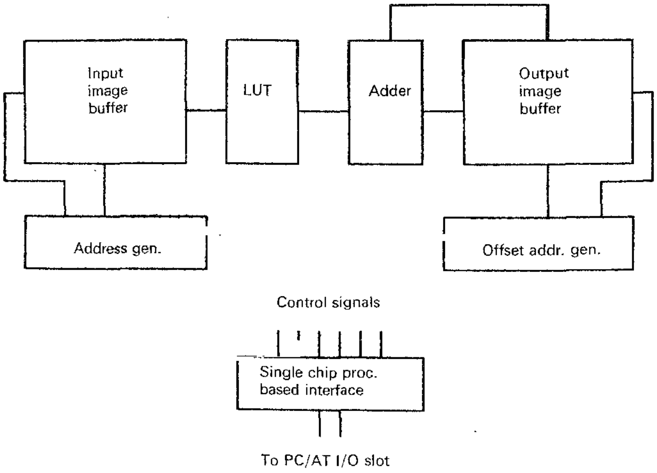

FIG. 2. Functional block diagram of the fast convolver.

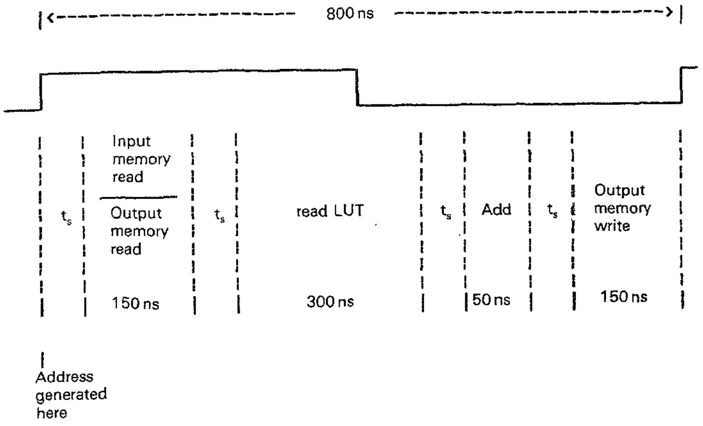

FIG. 3. Timing details of the fast convolver.

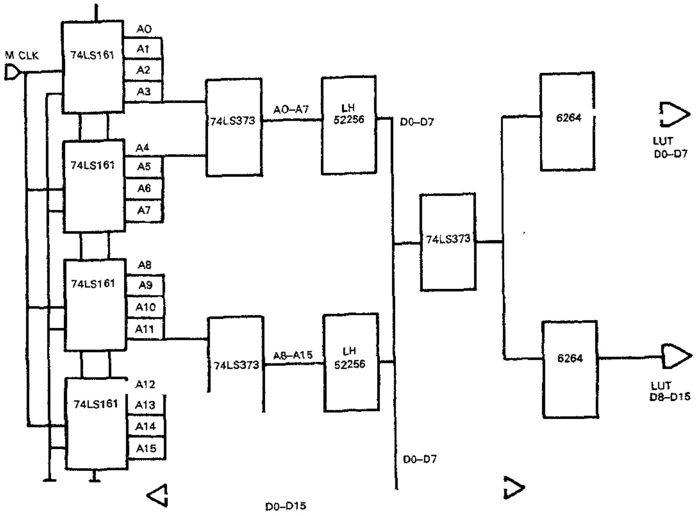

FIG. 4(a). Input section.

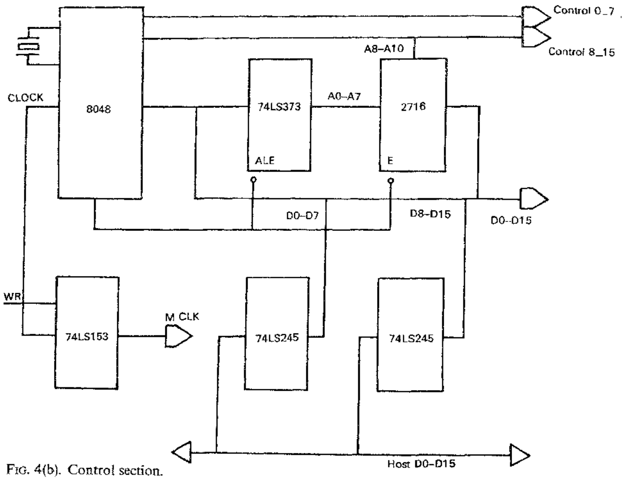

FIG. 4(b). Control section.

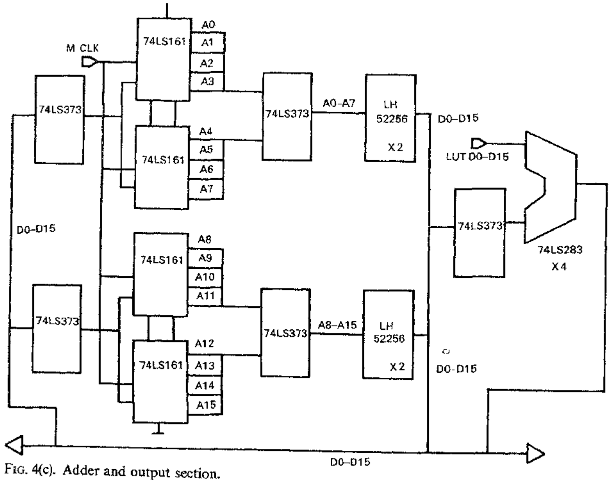

FIG. 4(c). Adder and output section.

FIG. 4(a)–(c). Hardware schematic diagram.

In one clock period the following operations have to be completed:

1. Read input buffer; read output buffer.
2. Use data read from the input buffer to address the LUT and read the LUT.
3. Add the value read from the LUT to the one read from the output buffer.
4. Write the output buffer.

The major time factors are the time taken for steps 1, 2, 4 and 5. The access time for the input and output buffers is 150 ns, and that of the LUT 300 ns. The adder takes 50 ns to complete 16-bit addition. In step 1, both the input and the output buffer are read simultaneously. So the total time taken for the above operations is $150+300+50+150 = 650$ ns. One pass is completed after 65536 clocks. The time taken (Fig. 3) to complete one pass of the convolution is derived to be 65536×800 ns.

For example, to complete the convolution of a 256×256 image with a 3×3 mask, the time taken will be $52.4 \times 9 = 471.6$ ms. That is approximately 1/2 s. If all the nine passes are performed in parallel, the time taken will be the same as that for one pass (54.2 ms). This does not include the time taken for the initial loading of the image and the setting up of the LUT. By using faster memory, typically 80 ns, the processing time per pass can be reduced to about 20 ms.

Along with the hardware development, the interface software has also been developed. The user is given a set of calls to download data, set operation codes, and get return codes. There are opcodes for the following operations at the board level: (i) Load input buffer from the host, (ii) start pass, (iii) load LUT, (iv) read output buffer, (v) load row- and column-offsets, and (vi) end of operation.

The opcodes for these operations are sent to the single-chip processor, MCS-48, controlling the board activities. MCS-48 decodes the opcodes and starts the appropriate action. After the completion of the operation a return code is sent to the host. Coordination between the board and the host is achieved through mutual interrupts.

7. Conclusions and further work

The principle of operation of the fast convolver board, and its parallel implementation, emphasizing the speed-up factor, are described. The speed advantage suggests that the parallel version will facilitate near real-time performance of low-level computer vision tasks. After the completion of prototype testing, parallel implementation will be taken up. High-level language support for the fast convolver will be provided so that the user need not use the low-level calls to use the fast convolver.

It is possible to extend the design of the convolver board to perform morphological operations, such as erosion and dilation. This can be achieved since the morphological operators are similar in nature to the convolution operation. In other words, slight modifications in the adder stage will make the board a multifunctional unit.

Acknowledgement

We wish to thank Ms Vidya and Ms Beena for assistance in building the hardware and preliminary testing and the reviewers for valuable suggestions.

References

1. KUNG, H. T. AND SONG, S. W.	A systolic 2-D convolution chip, *Multicomputers and image processing: Algorithms and programs*, (eds) K. Preston and L. Uhr, 1982, pp 373–384, Academic Press.
2. KUNG, H. T., RUANE, L. M. AND YEN, D. W. L.	Two-level pipelined systolic array for multi-dimensional convolution, *Image Vision Computing*, 1983, **1**, 29–41.
3. LEE, S. Y. AND AGGARWAL, J. K.	Parallel 2-D convolution on a mesh-connected array processor, *IEEE Trans.*, 1987, **PAMI-9**, 590–594.
4. MARESCA, M. AND LI, H.	Morphological operations on mesh-connected architectures: A generalized convolution algorithm, *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 1986, pp 299–304.
5. CHANG, J. H., IBARRA, O. H., PONG, T. C. AND SOHN, S. M.	Two-dimensional convolution on a pyramid computer, *IEEE Trans.*, 1988, **PAMI-10**, 590–593.
6. RANKA, S. AND SAHNI, S.	Convolution on SIMD mesh-connected multi-computers, *Proc. Inter. Conf. on Parallel Processing*, Vol. 3, 1988, pp 211–217.
7. GIORDANO, A., MARESCA, M., SANDINI, G., VERNAZZA, T. AND FERRARI, D.	A systolic convolver for parallel multi-resolution edge detection, *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 1986, pp 457–461.
8. FANG, Z. AND NI, L. M.	Parallel algorithms for 2-D convolution, *Inter. Conf. on Parallel Processing*, 1986, pp 262–269.
9.	*PC-AT Reference manual*, Intel, 1989.
10.	*Microcontroller handbook*, Intel, 1984, pp 15: 1–16: 32.
11.	*Signetics TTL data manual*, 1984, pp 4: 261, 4: 435.