A Computing Architecture of Adjustable Convolution System for Image Processing

PO-NING CHEN*, YUNG-SHENG CHEN* and WEN-HSING HSU*,**

*Institute of Electrical Engineering,
National Tsing Hua University
**Institute of Information Science,
Academia Sinica

This paper describes a design of adjustable convolutional hardware in image processing. As performing an $n \times n$ convolution traditionally needs $n^2$ processing elements (PEs), therefore, the larger the mask size is, the more the number of PEs and the cost are. In order to achieve a reasonable performance/cost ratio, we design a new architecture using $n$ or less PEs to perform an $n \times n$ convolution. This hardware architecture together with a few delay-line buffers is designed in pipeline-and-parallel fashion. Presently, a $3 \times 3$ convolutional prototype consisting of only two PEs has been constructed on a single board and can approximately operate in real-time.

1. INTRODUCTION

Preprocessings play an important role in the vast image processing and pattern recognition systems. Researchers have always tried to find out some effective and efficient methods of preprocessings. For instance, the significant intensity changes in a grey-scale image, which is so called "edge", have to be extracted by edge detection for further line or curve extractions [1, 2]. Sometimes it may require to remove the noise of high frequency by a moving average filter or a smoothing operator [3, 4]. All of these operations can be easily implemented by the well-known convolutional (or masking) operator.

Traditionally, the convolutional algorithms are implemented in software. The chief disadvantage is that the processing will take an excessive amount of time depending on the sophistication of the algorithm and the performance of the host machine. So, it is not an efficient way except for the hardwiring in developing real-time image processing system.

† Received November, 1986

63
An obvious and intuitive way of speeding up the convolutional process is to use backend-hardware. However if the hardware is implemented in direct form, (such as a $3 \times 3$ convolution using nine multiplicative units and one nine-inputs additive unit), it could be so expensive that it is not appropriate for some users whose research works are on personal computers. Nowadays, many Image (or Digital) Signal Processors have become chip products which thanks to the advanced VLSI technique. But the cost is still too high for the researchers using a personal computer. For example, to use four ISP (Image Signal Processor) chips each containing four PEs [5], we can perform a $4 \times 4$ convolution for the $256 \times 256$ gray-scale image at the speed of 10.9 msec. But this architecture needs 16 complex PEs to complete its work, which means paying a high price for a high performance.

In order to achieve a reasonable performance/cost ratio, we have designed a generalized $n \times n$ convolutional hardware, using only $k$ PEs, where $k$ is adjustable from 1 to $n$. To avoid the degradation due to reduction of PEs, it seems that we need more powerful PEs whose access time must be within 100 nsec; and undoubtedly such performance is easily achieved by the current VLSI technique. Hence we design a new hardware architecture, using only small number of PEs and some variable-length buffers to match the real-time performance (i.e., to match the video scanning speed.)

We must emphasize that many buffers are used in this architecture. But it will not increase the cost of the hardware because the memory device is quite cheap now and will be getting cheaper.

This paper first describes the entire hardware prototype, which accomplishes a $3 \times 3$ convolutional operator at a maximum rate of 10 million pixels per second. Then it discusses the generalized architecture of the convolutional algorithm and evaluates its performance. In order to introduce the system clearly, we detail the working procedure step by step in section 2.2.3.

2. SYSTEM DESCRIPTION OF $3 \times 3$ CONVOLUTION USING TWO PEs

Before describing the convolutional hardware, the formula of
performing an $n \times n$ convolutional operator should be reviewed and expressed as below:

$$F(i, j) = \sum_{k=-1}^{n} \sum_{l=-1}^{n} h(k, l) f(i-k, j-l)$$

(1)

where matrices $f$ and $F$ contain the raw and convolved image data, respectively, and matrix $h$ contains the mask (or weight) coefficients.

Figure 1 shows a simplified block diagram for the entire system. The major reason of using only two PEs in this system is to reduce the cost. This architecture is the simplest case among all generalized architectures which is fast enough to match the video scanning rate as the executing time of the PEs is under 50 nsec (it is possible for current technique). At present, the system has been constructed on one printed circuit board using TTL device (see Fig. 8.)

![Block Diagram](image-url)

Fig. 1. The block diagram of $3 \times 3$ convolutional hardware with two processing elements. Where ADD/R consists of an adder and a 19-bit register.
2.1 Convolution mask buffer

The hardware system consists of two primary parts: one is convolution mask buffer and the other is processing block. The convolution mask buffer is used to store the mask coefficients \( h(k, l) \) which shown in formula (1). In order to make our system more flexible, the mask coefficients can be defined by software using "OUT" operator, so that the user can define the mask values of various convolutional functions. As Fig. 1 shows, the convolution mask buffer is designed in circulate fashion, so that the mask values can be used repeatedly under the control of system pipeline clocks.

2.2 Processing block

The processing block divides into three parts: named process-ins elements (PEs), delay-line buffers and parallel-in-serial-out units (PISOs). The performance of this system is limited by the processing speed of PEs. The PEs in this system are TDC 1008 chips, which can perform multiplication and addition within 100 ns. The TDC 1008 chip also consists of a 19-bit temporary register and can perform in two's complement or sign-magnitude mode.

2.2.1 Delay-line buffer

There are three delay-line buffers in the processing block of Fig. 1. The first buffer stores one-line data of the image picture in the host computer. And the second buffer, whose head links to the tail of the first buffer, also stores one-line data. In the same way, the second and the third buffers are linked tail-to-head. Hence, three delay-line buffers always store three consecutive lines of the image data.

2.2.2 Parallel-in-serial-out unit

The parallel-in-serial-out unit of Fig. 1 is viewed as the fetch pipes. The PISO 1 parallelsly accesses three image data at the same position of the three consecutive lines, then serially transmits them to PE 1 and PE 2. Because the \( 3 \times 3 \) convolution for pixels of odd and even address is done by PE 1 and PE 2 respectively, PE 2 needs the previous data having been used by PE 1 at every two steps. This is the reason why we add PISO 2 in the PISO unit.
Then the 2-by-1 switch box will route the proper data to PE 2. The simplified work procedure is that after the PISO 2 has been filled with the previous contents of PISO 1 the switch box changes to the state of lower-arbitration and PISO 2 sends the data to PE 2 in serial.

The output of PISO 1 links to the input of itself. The major reason for the queueing design of PISO 1 is that, like PISO 2, the PE 1 needs the previous data of the PISO 1 at every two steps.

2.2.3 The processing procedure

In this section, we will show the processing procedure step by step as follows:

Step 0. As in Fig. 2(a), the image data have already been loaded into the frame memory. And the mask values have also been defined in the convolution mask buffer by the user.

Step 1. Image data is fetched simultaneously from three delay buffers into the three stages of PISO 1 as shown in Fig. 2(b). After three clocks, the image data of PISO 1 is transmitted synchronously to PISO 2 and the $x$-inputs of multipliers of PE 1 and PE 2. At the same time, the data of PISO 1 is re-transmitted into PISO 1, which is queueing-designed. At the end of this step, the accumulator of PE 1 contains the sum of the first six products of the convolution for odd address pixels, as shown in Fig. 2(c). At this time, PE 2 has completed the convolutional operation and the value contained in the register of PE 2 is transmitted to the image frame-memory.

Step 2. Re-access the image data into PISO 1. And the clock controller sends three pipeline clocks to the PISO unit and the convolution mask buffer simultaneously. As we have mentioned in section 2.2.2, PE 2 uses the previous data of PISO 1, which now is stored in PISO 2. Hence, the switch-box must be lower-arbitrative now. The result is shown in Fig. 2(d). Also, PE 1 transmits the content of its register to the image frame memory at the end of this step.
Fig. 2. The processing procedures. (a) Step 0, (b) Step 1, (c) Step 2, (d) Step 3, and (e) Step 4.
Fig. 2. (continued)
Step 3. Without accessing new image data into PISO 1, the system repeats sending three pipeline clocks, and the PE must use the same image data as step 2. Hence the switch box changes back to upper-arbitrative state and the queueing-designed PISO 1 have already stored the previous data used in step 2. The result of the system is shown in Fig. 2(e). The output lines of PEs are disabled at this step.

Step 4. Repeat step 1 to 3 until all the pixels of the image are computed.

This architecture can be further extended to $n \times n$ convolution for $k$ PEs. Figure 3 shows the architecture of the $5 \times 5$ convolution using two PEs. In section 3 we will discuss the generalized architecture in detail.

2.3 The real-time architecture

In Fig. 4, we show the real-time architecture of the image-data buffer unit for the $3 \times 3$ convolution using two PEs. Here we use the concept of double-buffering (or called swing buffering), that is,
when one buffer is processing, the other is scanning at the same time.

The major reason for using multi-buffering is that the access operation will not happen in every step. As mentioned in step 3 of section 2.2.3, the access operation is disabled every two steps and this does not fit the frequently accessing fashion of scanning video. In order to absorb the difference of the two systems, we use multi-buffering techniques to match real-time processing.

The architecture of one-line buffer is shown below.
Each block of the buffers stores one line of the image data, and each buffer set contains four blocks. The image data has to be accessed in an overlapped fashion as shown in the above figure. We ignore the design of its accessed control because it is not important here.

There are totally 8-by-4 line buffers in this system. The working procedure is shown step by step as below.

```
1  1 2 3 4  
2  2 3 4 5  
3  3 4 5 6  
4  5 6   
   .. 9 8 7   5 6   
6  
7  
8  
```

**Step 1.** Sets 1, 2, 3 are under processing; sets 4, 5, 6 are under scanning; and sets 7, 8 are back for the next step.

```
4  5 6 7 8  
5  8 7 8 9  
6  7 8 9 10 
   .. 13 12 11   8 10   
7  
3  
1  1 2 3 4  
2  2 3 4 5  
```

**Step 2.** Now, the sets are rearranged. Sets 4, 5, 6 are under processing; sets 7, 8, 3 are under scanning; and sets 1, 2 are back for the next step.
Step 3. Also, like step 2, the new arrangement is shown above. The steps are illustrated in Table 1.

<table>
<thead>
<tr>
<th>Processing set numbers</th>
<th>1</th>
<th>4</th>
<th>7</th>
<th>1</th>
<th>4</th>
<th>7</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td>5</td>
<td>8</td>
<td>2</td>
<td>5</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>6</td>
<td>3</td>
<td>6</td>
<td>3</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>Scanning set numbers</td>
<td>1</td>
<td>4</td>
<td>7</td>
<td>1</td>
<td>4</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>5</td>
<td>8</td>
<td>2</td>
<td>5</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>6</td>
<td>3</td>
<td>6</td>
<td>3</td>
<td>6</td>
<td>3</td>
</tr>
</tbody>
</table>

It is shown that sets 1, 4, 7 and sets 2, 5, 8 are triple-buffering in Table 1 and sets 3, 6 are double-buffering. Because the switching network is easy to implement, we also ignore it here.

The simultaneous processing and scanning is possible under the multi-buffering techniques. We shall point out that, using three PEs instead of two PEs may simplify the architecture or release a few buffers. However, cutting down one PE and adding some buffers may reduce the cost from $O(3)$ to $O(2)$. Hence, it is worthwhile to use this architecture if the cost of the memories is cheaper than that of PEs.

3. GENERALIZED $m \times n$ CONVOLUTIONAL SYSTEM

This section introduces the generalized convolutional architecture and its control sequences. By using variable-length buffers, we can do any $m \times n$ convolution with $n$ or less PEs, which means reduction of cost.
3.1 Control sequence and architecture

The control sequence and architecture of $m \times n$ convolution are illustrated in Fig. 5. The $S$ (witch) box is set if it is upper-arbitrated, and reset if it is lower-arbitrated, as shown in Fig. 6. $A$ (cess) control-bit in "set" condition means that it is accessing new image data, just like step 1 in section 2.2.3. The $O$ (output) control-bit controls which output of PEs must be enabled at each step. The detailed control sequences are described as follows.

![Diagram of control sequence and architecture](image)

**Fig. 5.** The generalized architecture of $m \times n$ convolver with $k$ PEs.

![Switch box](image)

**Fig. 6.** A switch box which is a 2-by-1 arbiter.

When the processing begins, the system clock controller repeatedly sends $\hat{S}$, $\hat{A}$, and $\hat{O}$ until the computation completes. Control sequences $\hat{S}$, $\hat{A}$, and $\hat{O}$ are defined below:
The $S$ control vector: \( \hat{S} = \begin{bmatrix} S_1 \\ S_2 \\ \vdots \\ S(k-1) \end{bmatrix} \) \hfill (2)

where \( S_i = [1 \ldots \cdot 1 0 \ldots \cdot 1 1] \), \( k-i \leq i \leq n-k+i-1 \).

The $O$ control vector: \( \hat{O} = \begin{bmatrix} O_1 \\ O_2 \\ \vdots \\ O_k \end{bmatrix} \) \hfill (3)

where \( O_i = [0 \ldots \cdot 0 1 0 \ldots \cdot 0 0] \), \( k-i \leq i \leq n-k+i-1 \).

The $A$ control sequence: \( A = (1 \ldots \cdot 1 0 0 \ldots \cdot 0) \). \hfill (4)

For instance, the control sequences of the prototype (where $n$ and $k$ are equal to 3 and 2, respectively), are:

\( \hat{S} = [1 0 1] \); the first "1" means that the switch box is upper-arbitrative at step 1, the switch boxes are lower-arbitrative at step 3, respectively.

\( \hat{O} = [0 1 0] ; \) the "1" in the first line denotes that the output PE 1 is enabled at step 2, the "1" in the second line denotes that the output PE 2 is enabled at step 1.

\( A = (1 1 0) ; \) the same as above, it means that the access operation is enabled at step 1 and step 2, and disabled at step 3.

Note that where each step consists of $n$ pipeline clocks, the access and output clocks are combined with the first and the last pipeline clocks at each step.

For the sake of simplicity, if using $n$ PEs to do an $n \times n$ convolution, it is not necessary to count $A$(cess) control sequences; that is, the access operation performs at each step. And also, it is not necessary to use $n$ PISOs and $(n-1)$ switch-boxes in the $n \times n$ convolver. The most important thing is that this architecture does not need multi-buffering delays in real-time processing. We show the simplified architecture and control sequence in Fig. 7.
Fig. 7. The simplified architecture of $n \times n$ convolver with $k$ PEs.

We see that there is a trade-off between multi-buffering and multi-PEs. But for the present VLSI techniques, the former will be more preferable because of its cheapness.

3.2 Performance evaluation

This section discusses the performance of the generalized convolutional architecture. Assuming that the pipeline clock period is $T$ nsec and there are only $k$ PEs, then the computing time delay ($T_d$) for $N \times N$ image scale is

$$T_d = \frac{(N \times N \times m \times n \times T)}{k}.$$  \hspace{1cm} (5)

Obviously, for a $256 \times 256$ grey-scale image and the $3 \times 3$ convolution with two PEs (i.e., $N = 256$, $m = n = 3$, $k = 2$), the processing time will be 14.7 msec, if $T$ is equal to 50 nsec. Consequently, if we use more powerful PEs and faster clock rate, this architecture can easily achieve the real-time performance.

As mentioned in section 1, this system is bounded by PE's computing speed. So, if we could increase the computing speed of PE, its performance will be further improved, and a high performance/cost ratio can be achieved.

4. PHYSICAL IMPLEMENTATION

The prototype, where photograph is shown in Fig. 8, now has been constructed on a single board using TTL device. The performing results for a given image, which is shown in Fig. 9, are displayed from Fig. 10 to Fig. 12. They are named smoothing, edge detection and Laplacian process, respectively.
Fig. 8. The $3 \times 3$ convolution prototype is constructed on a single board and connected with the personal computer.

Fig. 9. The image directly captured from camera.
Fig. 10. The result of smoothing performed by the convolution prototype.

Fig. 11. The result of edge detection performed by the convolution prototype.

Fig. 12. The result of Laplacian process performed by the convolution prototype.
In Table 2, we show comparison of the performance of the prototype and the software using 8 MHz clock rate written in 8086 assembly language on NEC 9801 F2 personal computer.

Table 2. The processing speed of $3 \times 3$ convolution by 8086 assembly language and that of the prototype

<table>
<thead>
<tr>
<th></th>
<th>Smoothing</th>
<th>Edge detection</th>
<th>Laplacian</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software</td>
<td>3 sec</td>
<td>12 sec</td>
<td>12 sec</td>
</tr>
<tr>
<td>Hardware</td>
<td>69 msec</td>
<td>69 msec</td>
<td>69 msec</td>
</tr>
</tbody>
</table>

5. CONCLUSIONS

The system presented in this paper is an inexpensive hardware by which the convolutional algorithm can be applied to video data at a better performance/cost ratio. The hardware also provides a way to improve the fidelity of datum-bounded image processing. Furthermore, the PE-reducing architecture also provides many choices to achieve desired performance at both the reasonable cost and complexity.

Presently, a prototype of performing the $3 \times 3$ convolution has been constructed on one printed-circuit board using TTL device and is working with a 16-bit personal computer. Preliminary studies strongly indicate the possibility of both reducing the hardware cost and achieving real-time performance.

REFERENCES