A NEW ARCHITECTURE FOR IN-MEMORY IMAGE CONVOLUTION

Vasily G. Moshnyaga, Kazuhiro Suzuki and Keikichi Tamaru
Department of Electronics & Communication, Kyoto University
Yoshida-Honmachi, Sakyo-ku, Kyoto 606-01, JAPAN

ABSTRACT

A new memory-based architecture for real-time image convolution with variable kernels is proposed. The architecture exploits the highest possible bandwidth inherent in memory and achieves the fine-grain parallelism of computations inside the memory. Unlike existing approaches, the architecture ensures convolution with very large kernels under the real time constraints of video applications. It does not require external memory banks or large I/O count and features single chip VLSI implementation.

1. INTRODUCTION

1.1. Motivation of this work

Discrete image convolution (or non-recursive FIR filtering) is one of the most critical steps in digital image processing. Having a 2-D image X of $(M \times N)$ pixels and a 2-D kernel, W of $(H \times K)$ coefficients with finite support, the task of convolution of X by W can be represented by the following expression:

$$y(i,j) = \sum_{h=0}^{H-1} \sum_{k=0}^{K-1} w(h,k) \times x(i-h, j-k)$$

where $0 \leq i \leq M-1$, $0 \leq j \leq N-1$, $x(i,j) \in X$ and $w(h,k) \in W$.

Algorithms involved in image recognition, correlation, enhancement, usually demand convolution by dynamic range kernels, with sizes varying from 2×2 and 3×3 (for the first order methods) up to 16×16 or 32×32 (for the second order methods). The larger the values of H and K, the closer the convolution filter will match the desired frequency response, the better enhancement and restoration will be provided[1]. Furthermore, in most applications the convolution is applied at the front end of image processing and hence has to be carried out at real-time requirements. For example, if an image has 720×576 pixels coded on 8 bits (standard TV Format), a kernel has 32×32 coefficients coded on 6 bits, and the input pixel rate of 27MHz, over 14 Giga operations have to be executed per second, with one operation being a 6×8 multiplication-accumulation (MAC). The only way to handle such an enormous computational rate is to implement the convolution algorithm in hardware.

1.2. Related Research

A number of hardware architectures have been proposed for image convolution over the last decade[2]-[7]. Despite differences, all of them have one feature in common: they all assume that image data is stored in a memory external to the processing array. Therefore efforts have been concentrated on the data flow optimization within processor array to perform computations as fast as possible under severely limited I/O bandwidth. The goal is usually approached by restricting the kernel size up to 10 by 10 format and reducing the coefficient variations to central symmetry[2] or power of two[8]; partitioning of large convolutions into cascaded 1-D filter sections[9],[10] to be executed in parallel. Although these methods speed-up the execution, they are bounded by the bandwidth limitations imposed by the memory-processor separation. As result none of the proposed architectures can deal with kernel format larger than 12×12.

This work differs from the previous research in that it executes convolution inside the memory. By placing the actual processing logic on the memory chip, and logically and physically matching it to memory array, we utilize the excessive internal memory bandwidth available within the memory structure itself. Several researchers have already reported the memory-logic integration benefits for general purpose application [11],[12]; vector quantization[13], motion estimation[14], DCT[15], multimedia[16], [17], etc. However, the problem of memory-based convolution has not been investigated yet.

1.3. Contribution

In this paper, we propose a new memory based architecture for the real-time convolution with variable kernel format. In our design, explicit provisions have been made to exploit the highest possible bandwidth inherent in memory and achieve the fine-grain parallelism of computations at the pixel level. Compared to existing architectures, our architecture ensures convolution with very large kernels (up to 32×32 coefficients) while satisfying the timing constraints of the real-time video applications. The architecture features small I/O count, regular layout and is suitable for single-chip VLSI implementation.

The next section presents the architecture. Section 3 analyzes the performance and implementation cost. Conclusions are drawn in section 4.

2. THE ARCHITECTURE

2.1. Overview

The proposed architecture consists of Functional Memory Array (FMA) and controller, as shown in Figure 1(a). The inputs R/W and $Conv$ determine working mode ($Read$, $Write$ or $Convolution$), in which the architecture either reads/writes image data to location A in the FMA or convolves the data stored in the array to kernel (W) of size $H \times K$. (We assume that the kernel parameters are established by the user through inputs H and K). The image pixels are supplied via bus I The bus O outputs the convolution results.

2.2. Functional Memory Array

The FMA is composed of three shift registers (Rw, $R1$, $R2$), decoders and an array of $(N \times M)$ Functional Memory cells, $FM(i,j)$, $0 \leq i \leq (N-1)$, $0 \leq j \leq (M-1)$, as shown in Figure 1(b). The register Rw receives the active coefficient $w_{h,k}$ and shifts it to the left every clock cycle, sending the coefficient's Least Significant Bit (LSB) into the array (line w). The registers $R1$,$R2$ control the convolution process in horizontal and vertical direction, respectively. (The size of register $R1$ is of N bits; $R2$ is of M bits). Every array cell, $FM(i,j)$, is connected to its four neighbors, input bus ($I(j)$), output bus ($O(j)$), the coefficient line

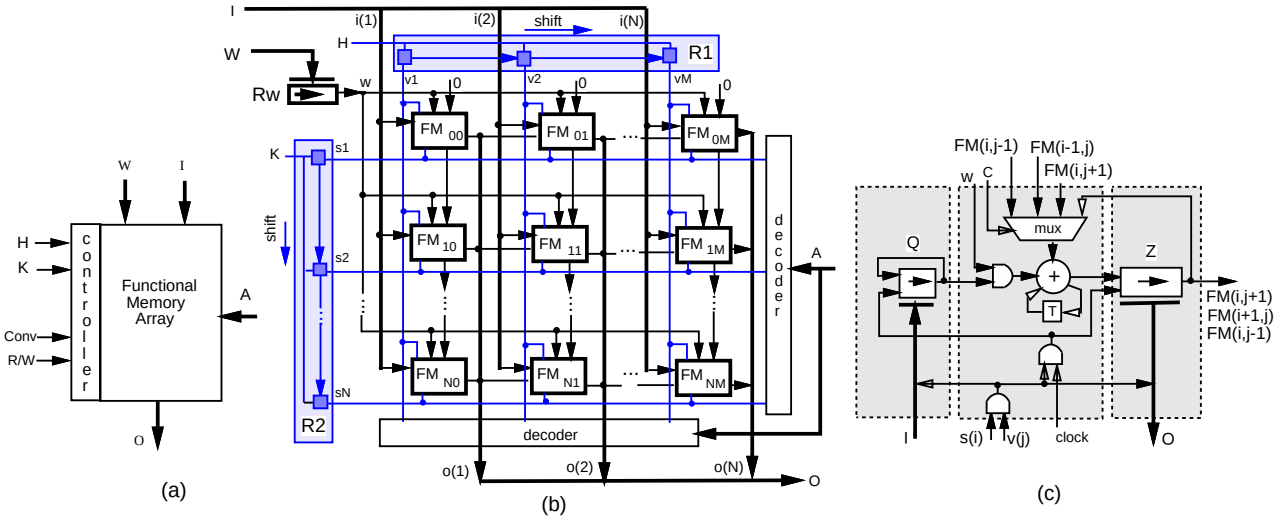

Figure 1. Proposed architecture: (a) An outline; (b) organization of the FMA; (c) structure of the cell $FM(i,j)$

(w), two select lines (s_i, v_j) and control lines. (For the simplicity, the control lines are not shown in the figure). During the read/write operation the select lines implement the addressing function, while in the convolution mode, they activate/disactivate logic elements within the cells.

Figure 1(c) shows internal organization of a cell $FM(i,j)$. As patterns depict, the cell consists of three parts: two memories built upon shift registers Q and Z, respectively, and a logic circuit placed between the memories and composed of 1-bit adder, 4x1 multiplexor (mux) and three gates. The register Q stores the image pixel, $x(i,j)$; the register Z stores the intermediate and final result, $y(i,j)$. In every clock cycle, t, the cell's logic computes the bit-product, $p^{[t]}(i,j)$, of the pixel bit $x^{[t]}(i,j)$, fed from register Q, and the coefficient bit, $w^{[t]}(h,k)$, fed from the input (w), and adds it to the corresponding bit of the sum $y^{[t]}(i,j)$ accumulated at the previous cycle in the neighbor cells $(FM(i,j-1), FM(i,j+1), FM(i-1,j))$ or its own Z register. The carry signal of the addition, is kept in the flip-flop, T, while the sum is written to the LSB of the register Z if and only if both select lines in the cell are high, i.e. $s(i)=1$ and $v(j)=1$. Shifting of the registers Q and Z by 1-bit to the left at the end of each cycle brings a new $(t+1)$ bit of $x(i-h, j-k)$ and frees the LSB of Z for a new bit of $y(i,j)$. Since this serial accumulation necessitates processing of all b_z bits of the register Z,for each coefficient bit, $w^{[t]}(h,k)$, the product of $x(i,j)$ by $w^{[t]}(h,k)$ will be computed after one iteration of b_z clock cycles. If the coefficient has b_z bits in size, the product, $x(i,j) \times w(h,k)$ will be accumulated in the register Z after b_w of such iterations (i.e. one computational phase).

2.3. Controller

Depending on the input signals, the controller can be either in the memory access state activating the I/O-busses and blocking the clock, or in the computation state when it supplies signals, C to select multiplexor's inputs in the FM cells and shifts the registers $R1, R2, Q$ and Z. Given the convolution parameters H, K, the controller selects the input $FM(i-1,j)$ at the initial phase $(n=0)$, and then in every $n \times (H-1)$ phase (n=1,2,..), during which $R2$ is shifting one bit down. In between, the controller selects the inputs $FM(i,j+1)$ or $FM(i,j+1)$ and shifts $R1$ to the right or to the left. In order to sustain for any kernel the correct placement of the convolution results in the memory array, the controller implements the right-first zig-zag shifting (Fig.2,a) when H is odd, and the left-first zig-zag shifting (Fig.2,b) when H is even. This means that the shift right of $R1$ and selection of the input $FM(i,j+1)$ are performed every odd sequence of $(K-1)$ phases, when H is odd and every even sequence of $(K-1)$ phases, when H is even. Correspondingly, the left shift in $R1$ and the $FM(i,j+1)$ input selection are performed in every even sequence of $(K-1)$ phases, if H is odd, and in every odd sequence of $(K-1)$ phases, if H is even. Thus, during computation the partial results are iteratively moved between the FMs cells: (K-1) times in a horizontal scan and then one step down in vertical scan.

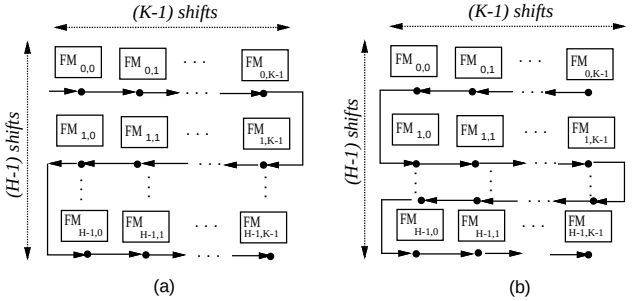

Figure 2. Data-transfer direction of odd H (a) and even H (b)

2.4. The system operation

We assume that before processing, all pixels of an image are written into the FMA in a way that $x(i,j)$ is stored in register Q of memory cell, $FM_{i,j}$, Also, we assume that the kernel parameters (H,K) are set on registers $R1, R2$ through the following masking of the register bits:

$$R1(i) = \begin{cases} 1 & \text{if } 0 \le i \le (M-K) \\ 0 & \text{otherwise} \end{cases}$$

$$R2(i) = \begin{cases} 1 & \text{if } 0 \le i \le (N-H) \\ 0 & \text{otherwise} \end{cases}$$

Figure 3(a) illustrates the pixel distribution in the FM array and initial state of the registers $R1, R2$ for a simple example of 4×4 image and 2×2 kernel (Fig.3,b). Let us assume for simplicity that all Z registers in the array initially null. (This however is not necessary in the reality).

The convolution begins with sending the coefficient, $w(1,0)$, to the register Rw and broadcasting its LSB value to all the FM cells.

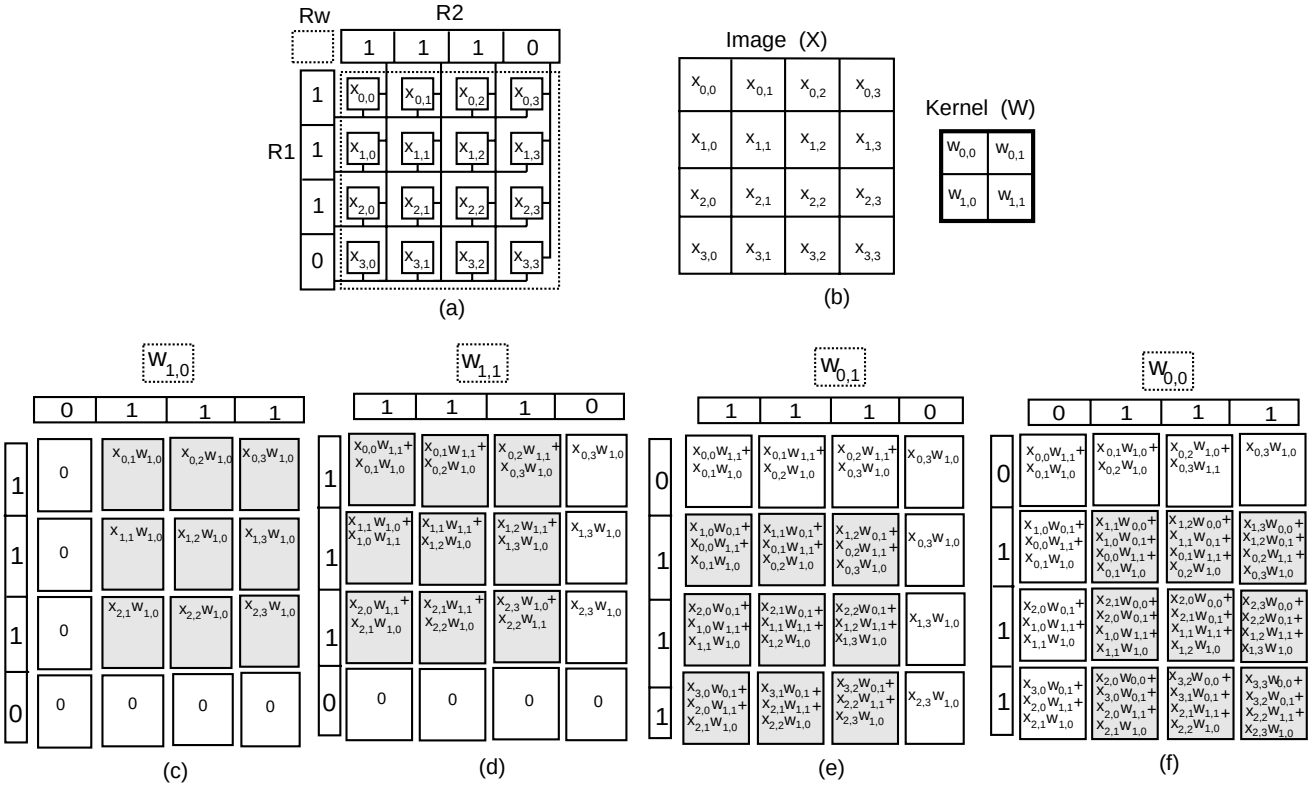

Figure 3. Illustration of the convolution process on example of 4×4 image and 2×2 kernel

During the first $b_z \times b_w$ consecutive cycles, each FM cell in the array multiplies its own pixel by the coefficient value and adds the product to the partial sum, taken from its top-adjacent cell. Since cells with zero select lines can not save the results in registers, the partial results are stored only in the $(M-K) \times (N-H)$ cells, shown by grey patterns. Figure 3(c) depicts the results saved in the cells after the first computation phase.

With each new coefficient, the register $R1$ is circularly shifted one bit to the left (because the kernel size H is even), and the computational process is repeated. Thus during the K phases, the $(M-K) \times (N-H)$ partial sums calculated in the first phase are iteratively moved along the rows $(K-1)$ times; each time accumulating the result computed at the FM they visit. Figure 3(d) illustrates the results accumulated in the FMA after the second phase. When a new element of the kernel row enters the array, we move the register $R2$ one bit down, as it shown in Figure 3(e), and then start shifting $R1$ in the opposite direction implementing the zig-zag scanning fashion. As Figure 3 shows, the accumulation process is dynamic; unlike those of other architectures. The sum computed in the FM(0,0) cell in the phase 1 travels the $(M-K) \times (N-H)$ FMs in the zig-zag direction each time accumulating a new partial product. Generally, after $(K \times H)$ phases, all coefficients of the kernel are processed and all the $(N \times M)$ convolution results (1) are available simultaneously in the Z registers of the FM cells. Figure 3(f) exemplifies final convolution results computed in the array.

3. EVALUATION

Since the proposed architecture convolves $(M-K) \times (N-H)$ image pixels by one coefficient in parallel, taking $b_w \times b_z$ clock cycles per coefficient, the latency for processing the $K \times H$ kernel is $L = b_z \times b_w \times H \times K$. Hence the architecture can estimate up to $f_{max} = 1/clock\_period \times L$ images per second.

Figure 4 shows the number of images which can be convolved per second as a function of the kernel row (column) size assuming that $H = K$, the clock cycle of 5 ns, b_w=6 and b_z=24 bit. As can be seen, the architecture can convolve 1356 images/sec by kernels of 32×32 coefficients, 21 images/sec by kernels of 32×32 or 5.29 images by kernels equal to the Videophone Standard picture size ($N = 288 \times 352$). The cross symbol characterizes the maximal kernel size (H_{max}=240) which can be processed under the conventional video rate requirements (24 images per second). Note that no existing architecture can provide feasible solutions for the quadratic kernels larger than H=16. In comparison, our architecture can convolve any image by such kernel in 184 μ sec.

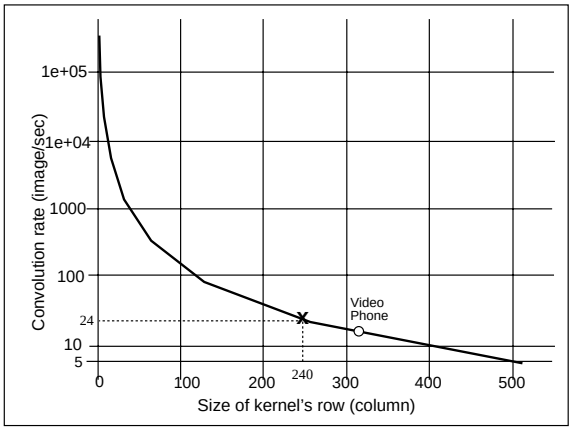

Figure 4. Image rate vs. kernel's row(column) size

Another advantage of our architecture is that it performs all the convolution operations in place with the original image. That is, the output and input images are stored in the same array, at the

same locations and there is no reformating or serialization of the input image. This saves a lot of time because we do not have to re-input the image or do any restructuring/ reformating of the data.

We evaluated the layout area requirements for convolving images of Video Telephone Format (288 × 352 pixels), TV Format CCIR Rec.601 (720 × 576 pixels) and HDTV Format (720 × 1280 pixels) under assumptions that each pixel takes 8 bits representation, a coefficient takes 6 bits and the maximal kernel size is 32 × 32. Table 1 shows the area estimation results for CMOS fabrication technology. As can be seen, the HDTV picture convolution by the 32 × 32 kernel can be implemented in $0.25\mu m$ CMOS technology in a single 20 × 20 mm^2 chip.

Table 1. Area requirements (mm^2)

Image Format	Number of FM cells	Technology(μm)		
		0.5	0.35	0.25
VideoPhone	101,376	169.4	86.4	42.3
Standard TV	414,720	692.9	353.5	173.2
HDTV	921,600	1539.7	785.5	384.9

4. CONCLUSIONS

We presented a new memory-based architecture for 2-D image convolution. Due to distribution of arithmetic/logic operations over the memory elements at the bit and word level, we were able to obtain a fine-grain computational parallelism and exploit the enormous bandwidth available within the memory array. The preliminary results show that the architecture ensures feasible solutions for the HDTV image convolution with not only parameterizable kernels but also with kernels of very large sizes (more than 1000 coefficients). Future research will be dedicated to detailed chip design.

REFERENCES

[1] W.Green, *Digital Image Processing: A system Appraoch*, 1989.

[2] J.Kim and W.Alexander, ''A Multipocessor Architecture for Two-Dimensional Digital FIlters'', *IEEE Trans. Computers*, C-36 (7), 1987, pp.876-883.

[3] B.Arambepola, et al., ''Cascadable One/Two-Dimensional Digital Convolver", *IEEE CICC'87*, pp.311-314.

[4] N.Demassieux, et al., ''A VLSi Architecture for Real Time Image Convolution with Large Symmetric Kernels", *IEEE ICASSP'88*, pp.1961-1964.

[5] H.Hwan, ''Systolic and Parallel Realization of 2-D Digital Filters", *IEEE ISCAS'90*, pp.2346-2348.

[6] M.Aboelaze, et al., ''Two-Dimensional Digital Filtering Using a Linear Processor Array", *IEEE ISCAS'91*, pp.2943-2944.

[7] C.Chou, et al., ''Modular Architectures for High Speed and Flexible Two-Dimensional Digital Filters", *IEEE ISCAS'90*, pp.2320-2323.

[8] J.Evans, et al., ''A High-Speed Programmable Digital FIR Filter'', *IEEE ICASSP'90*, pp.969-971.

[9] A.Venetsanopoulos, B.Mertzios, ''A Decomposition Theorem and Its Application to the Design and realization of two-dimensional Filters'', *IEEE Trans. ACSSP*, Vol.ASSP-33, (6), 1985, pp.1562-1575.

[10] T.Deng and T.Soma, ''Variable Digital Filter Using the Outher Product Expansion'', *IEE Proc. Vis.Image Sign.Proc.*, Vol.141,(2), 1994, pp.123-128.

[11] P.Kogge, et al., ''Combined DRAM and Logic Chip for Massively Parallel Systems'', *Proc. 16-th Conf. on Adv.Research in VLSI*, pp.4-13, 1995.

[12] D.Patterson, et al., ''A Case for Intelligent DRAM: IRAM'', *Proc. Hot Chips VIII*, pp.75-94, 1996.

[13] K.Kobayashi, et al., A Memory Based Parallel Processor for Vector Quantization'', *ESSCIRC'96*, pp.184-187, 1996.

[14] V.Moshnyaga, K.Tamaru, ''A Memory Efficient Array Architecture for Real Time Motion Estimation'', *Proc. IPPS'97*, pp.28-32, 1997.

[15] D.Elliott, et al., ''Computational Ram: A Memory-SIMD Hybrid Architecture and its Application to DSP'', *Proc. IEEE CICC*, pp.30.6.1-30.6.4, 1992.

[16] T.Kimura, et al., ''Design of 1.28-GB/s High bandwidth 2Mb SRAM for Integrated Memory Array Processor Application'', *J. of Solid-State Circuits*, Vol.30 (6), pp.637-643, 1995.

[17] T.Shimizu, et al., ''A Multimedia 32b RISC Microprocessor with 16Mb DRAM'', *1996 IEEE ISSCC*, pp.216-217, 1996.