Pipeline Architectures for Computing 2-D Image Moments

Carlos Coelho
cfspc@algos.inesc.pt

Nuno Roma
nuno.roma@inesc.pt

Leonel Sousa
las@inesc.pt

Dept. of Electrical Engineering
Instituto Superior Técnico/INESC
Rua Alves Redol, 9 - 1000-029 Lisboa,
PORTUGAL

Abstract

A digital image, or a segment of an image, may be represented by the moments of its intensity function. Image moments are used in image analysis for object modeling and matching. However, a large number of MAC arithmetic operations are required to compute high order moments; real-time processing is not achieved with nowadays general purpose computers. This paper proposes a new class of pipeline architectures for generating a single moment or a set of moments of an arbitrary order in real time. The architectures adopt wavefront processing techniques and floating-point arithmetic units. They only use a power core for generating all the required powers to compute the 2-D moments and are easily scalable to achieve the desired speedup.

1 Introduction

Two-dimensional object representation and recognition is an important topic in the computer vision area. Moments of the intensity function of a set of pixels are commonly used for representing an object or an image. Eq. 1 defines the moments $\mathcal{M}_{m,n}$ of an array of pixels with values $f(x,y)$ and size NM.

$$\mathcal{M}_{m,n} = \sum_{x=1}^{N}\sum_{y=1}^{M} x^m y^n f(x,y) \quad , \tag{1}$$

Low order moments are commonly used to find the location and orientation of an object. The zero through second order moments provide information about the area, center of gravity and about the approximation of an object to an ellipse [1]. High order moments are combined to form moment invariants to certain shape transformations, such as translation, rotation and scaling [2]. Moment invariants have been used as features for pattern recognition.

For computing eq. 1, NM additions and $(m+n)NM$ multiplications have to be performed. A moment invariant of order ψ results from the combination of the set of moments $\mathcal{M}_{m,n}$, with $\psi \leq m+n$.

In the last years, several algorithms and architectures have been proposed to improve the speed of moments computation. Some of them are only valid for low-order moments and/or for binary images [3, 4, 5]. Other proposals sugest architectures without practical interest due to its complexity [6]. This paper proposes a class of pipeline architectures for computing 2-D gray level image moments with any order in real-time. Floating point arithmetic has to be used to compute moments of different orders, given that the required dynamic range increases very rapidly with their order. The proposed architectures adopt the wavefront processing techniques to reduce the processing time. In synchronous processing arrays, such as systolic arrays, the "worst" processing period has to be respected. This may lead to inefficient systems, due to the variable processing time usually associated to the floating point arithmetic units. The proposed architectures are modular, which means that the required number of Processing Elements (PEs) depends only on the technology solutions and on the available time to process an image.

This paper is organized as follows. An efficient *power core* for generating the x and y powers is proposed in section 2. In section 3, a Dependence Graph (DG) corresponding to the computation of a single 2-D image moment is projected on a 1-D processor space, which is the basis for deriving the wavefront architectures presented in the remaining sections. Section 4 describes the proposed class of wavefront architectures for computing a single 2-D moment of an image received in the raster format. This class includes a *serial pipeline architecture*, with only 1 multiplier and 1 adder, and *parallel pipeline architectures* that improve the speed of the processing β times, by using β multipliers and β adders ($1 < \beta \leq M$). In section 5, a parallel pipeline architecture for computing a set of moments is proposed. Section 6 concludes the presentation.

2 Power Core

For computing the powers x^m and y^n with a basic sequential architecture, m and n clock cycles are required, respectively. Such architecture leads to a great amount of computation time for high order moments. A faster architecture is obtained by applying the following decomposition:

$$m = \sum_{i=0}^{k-1} b_i^m 2^i \quad ; \quad n = \sum_{i=0}^{k-1} b_i^n 2^i \tag{2}$$

$$k = \lceil \max\{\log_2 m, \log_2 n\} \rceil \tag{3}$$

The expressions for the powers x^m and y^n have the form:

$$x^m = x^{b_0^m.2^0 + b_1^m.2^1 + \ldots + b_{k-1}^m.2^{k-1}} = \prod_{i=0}^{k-1} x^{b_i^m.2^i} \tag{4}$$

$$y^n = y^{b_0^n.2^0 + b_1^n.2^1 + \ldots + b_{k-1}^n.2^{k-1}} = \prod_{i=0}^{k-1} y^{b_i^n.2^i} \tag{5}$$

Thus, it is possible to conclude that, in order to compute the value x^m, it is only necessary to use $2k$ multipliers: k of them for computing the x^{2^i} terms and the remaining k to calculate the product of the $x^{b_i^m.2^i}$ terms.

Figure 1 represents one possible hardware structure with k PEs to compute eq. 4. If the input values were composed by the sequence $\{1, 2, 3, \ldots, N\}$, the sequence $\{1^m, 2^m, 3^m, \ldots, N^m\}$ would be obtained at the output. The overall module is operated in a synchronous way, with the D symbol representing a processing delay. With this structure, it is possible to obtain one output value in each clock period, with a latency of k clock periods.

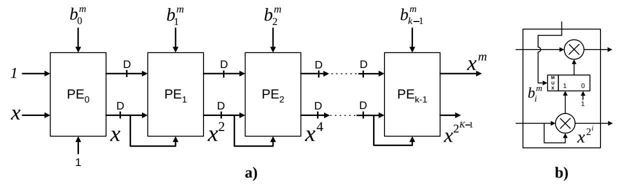

Figure 1: Power-Core module; a) Architecture; b) PE.

3 Deriving Pipeline Architectures

The computation of $\mathcal{M}_{m,n}$, given by eq. 1, corresponds to the product of a $N \times M$ matrix F (image matrix) by a row vector X with the x^m values, and by a column vector Y with the y^n values:

$$\begin{aligned} \mathcal{M}_{m,n} &= XFY = XG = HY \quad (6)\\ X &= \begin{bmatrix} 1^m & 2^m & \cdots & N^m \end{bmatrix} \\ Y &= \begin{bmatrix} 1^n & 2^n & \cdots & M^n \end{bmatrix} \\ F(x,y) &= \{f(x,y) : x = 1 \ldots N, y = 1 \ldots M\} \end{aligned}$$

Hence, the computation of eq. 1 corresponds to a vector-matrix product followed by a vector dot product, and can be represented by the planar DG of figure 2a.

If this DG was directly used as the structure of an array architecture, N by $(M+1)$ PEs would be required. For real images, with a great number of pixels, it is convenient to reduce the array dimension. This can be achieved by projecting the DG on a linear structure with a valid

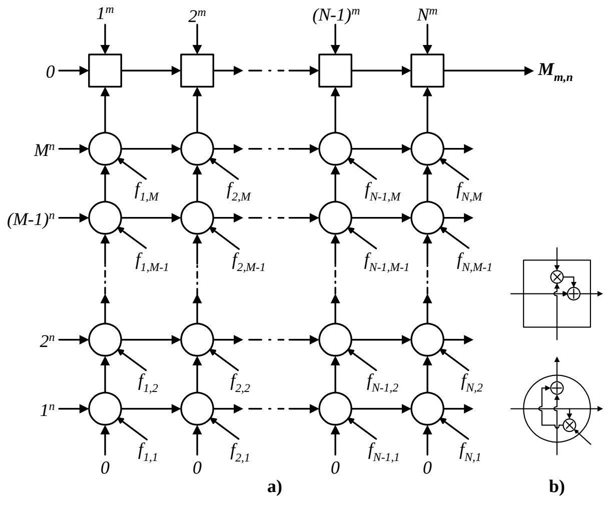

Figure 2: Dependence Graph; a) Structure; b) PEs.

pair of *projection* and *schedule* vectors $(\vec{d}, \vec{s})$. The systolic structure presented in figure 3 is obtained by defining $\vec{d} = \begin{bmatrix} 1 & 0 \end{bmatrix}^T$ and $\vec{s} = \begin{bmatrix} 1 & 1 \end{bmatrix}^T$, while maintaining the same type of PEs.

Although the dimension of the processor has been reduced, this structure still requires a large amount of hardware, namely for high definition images (e.g. N=M=1024). Furthermore, the rows of the image have to be provided in parallel, which requires the usage of storage devices since images are usually provided in the raster format.

One possible way to solve these problems is to map several PEs of the linear array on a single PE, using the partitioning technique [7]. The Signal Flow Graph (SFG) can be regularly partitioned into blocks, each consisting of a cluster of PEs, as will be further explained in section 4.2. In the limit, the set of all PEs represented by circles is mapped on a single PE. This will lead to a serial pipeline architecture, as described in section 4.1 (see figure 4a), which represents the most economic alternative in what concerns hardware resources. In fact, for the PE represented by a circle, only 1 multiplier is used. Partial results are accumulated in the output PE, as depicted in figure 4b. Configurable devices, such as Field Programmable Gate Arrays (FPGAs) [8], have enough capacity to implement such a simple processor. This architecture also has

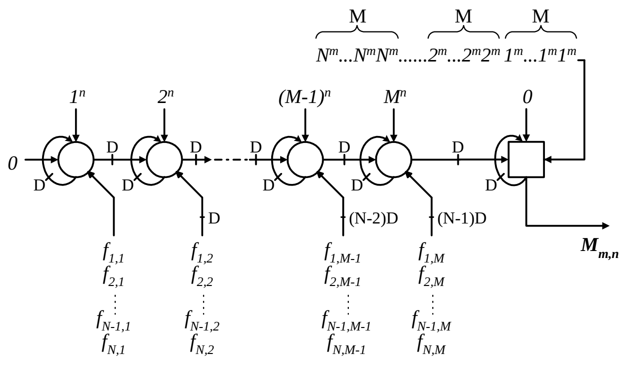

Figure 3: Result of projecting the DG.

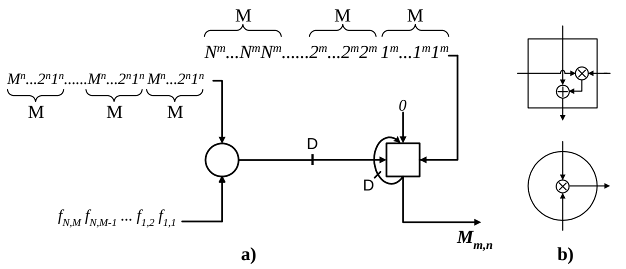
M
M
M
$N^m...N^mN^m......2^m...2^m2^m\ 1^m...1^m1^m$
$M^n...2^n1^n......M^n...2^n1^n\ M^n...2^n1^n$
M
M
M
0
D
D
$f_{N,M}\ f_{N,M-1} \cdots f_{1,2}\ f_{1,1}$
$M_{m,n}$
a)
b)

Figure 4: Result of mapping the linear array; a) structure; b) PEs.

the advantage of accepting the image data in a raster format. Nevertheless, it has the inconvenience of increasing the total processing time to values of the order of $N \times M$.

The structures described so far are based on a synchronous control scheme. However, as will be referred in section 4.1, it is possible to map these structures in wavefront array architectures by replacing the delay elements by data interchange devices, such as FIFO memories.

4 Pipeline Architectures for Computing a Single Moment

In this section, two main types of wavefront architectures are proposed for computing a single 2-D image moment: a serial pipeline architecture and a class of faster and modular parallel pipeline architectures.

As referred in the introductory section, the usage of floating-point arithmetic units leads to the choice of wavefront array architectures. This option considers the usage of a local clock inside each PE and wavefront processing techniques to control the operation of the overall processor. These architectures present some advantages when images are provided in a raster mode, since the data stream from the sensor usually does not have a constant sampling frequency, due to the horizontal and vertical retrace times. For a synchronous implementation, an input buffer memory (FIFO) would have to be used in order to provide the pixel data at a constant rate. In contrast, with a wavefront architecture, data is processed as soon as it is available at the input of the circuit.

4.1 Serial Pipeline Architecture

A block diagram of the serial wavefront architecture for the computation of a single moment is shown in figure 5. It only requires one power core block to generate the sequences of x^m and y^n and one PE to compute $\mathcal{M}_{m,n}$. Since the described architecture performs the processing of the image in a line by line fashion, the multiplication of the term x^m can be done independently:

$$\mathcal{M}_{m,n} = \sum_{x=1}^{N}\sum_{y=1}^{M} x^m \left(y^n f(x,y)\right) \qquad (7)$$

Before the processing of each line is started (when $Count\ y = 0$), the calculation of x^m is performed and its value is stored in a register. Then, during the processing of each line, this value is used to calculate the term $x^m\left(y^n f(x,y)\right)$. Therefore, it was possible to decrease the total number of multipliers with the cost of a single register.

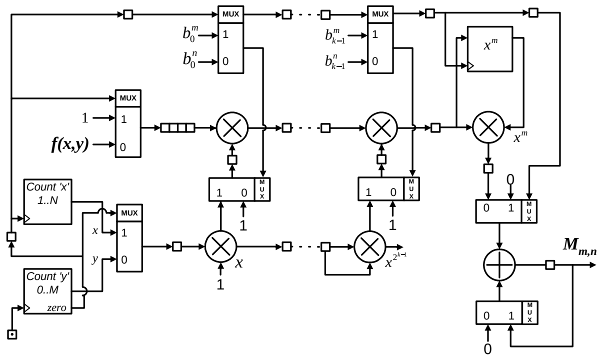

Figure 5: Block diagram of the serial processor.

As it can be seen in figure 5, the serial pipeline processor uses a total of $2k+1$ multipliers and 1 adder. Disregarding the time spent in data communication, the maximum processing time for calculating the $\mathcal{M}_{m,n}$ moment of an $N \times M$ image is:

$$T = [N(M+1) + t_{pow} + 2] \times t_{op}, \qquad (8)$$

where t_{pow} is the latency of the power-core circuit given by:

$$t_{pow} = 1 + \log_2\left(\lceil \max\{m,n\}\rceil\right) + 1 \qquad (9)$$

and t_{op} is the maximum time spent in performing a single floating-point multiply or addition. Hence, the total latency is:

$$T = [N \times (M+1) + \log_2\left(\lceil \max\{m,n\}\rceil\right) + 4] \times t_{op} \qquad (10)$$

4.2 Parallel Pipeline Architectures

In order to accomplish the desired computation time, the serial architecture may require very fast arithmetic units. The number of PEs can be increased to β, so that β pixels are processed in parallel. With the scheme represented in figure 6a, the β PEs process β adjacent columns of pixels and each PE processes $\mu = M/\beta$ columns, where β and μ are integers. If the image currently being acquired arrives in raster mode, which is a current standard, we would need an input buffer almost as large as the image itself. This is a serious drawback because the image may be quite large and because it increases the latency of the whole calculation.

The latency and the amount of storage can be reduced by using the interleaved pixel-processing scheme illustrated in figure 6b. Since each PE performs operations on

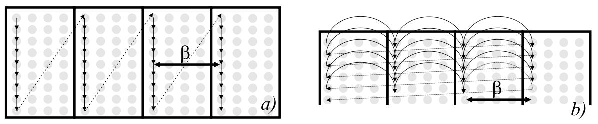

Figure 6: Processing sequence for PE_0 and β PEs: a) sequential column processing; b) interleaved pixel processing.

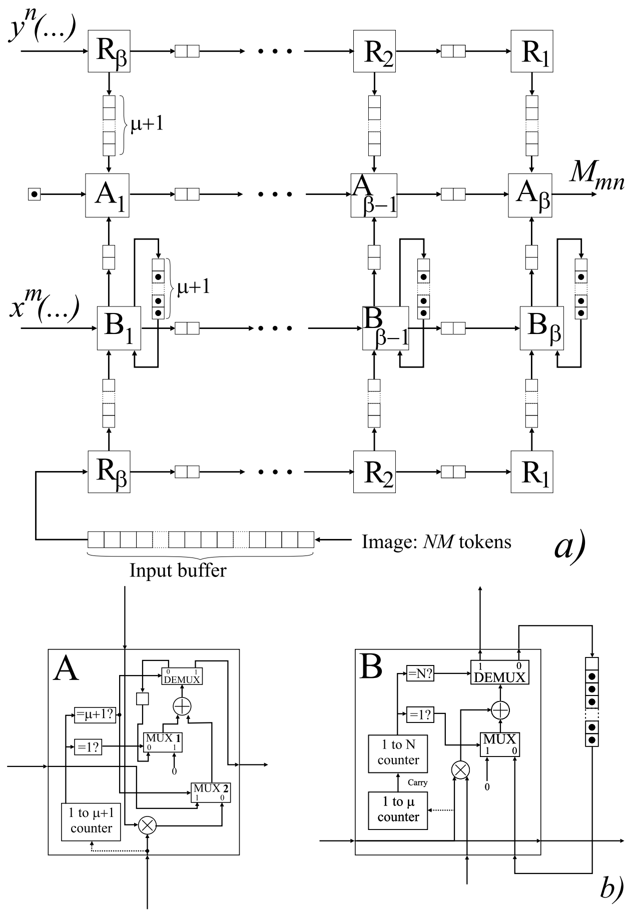

Figure 7: Wavefront array architecture for computing $\mathcal{M}_{m,n}$;a) Structure; b) PEs.

μ pixels of the same line, x^m changes more slowly than y^n. The number of multiplications can be reduced if the dot product HY is performed at the end (see eq. 6).

Figure 7a presents a wavefront array architecture with β PEs for computing the single moment $\mathcal{M}_{m,n}$. It adopts the interleaved pixel-processing scheme. In this figure, $y^n(\ldots)$ represents the n^{th} power of all integers from 1 to M and $x^m(\ldots)$ represents the m^{th} power of all integers from 1 to N, each repeated μ times. The inner structure of PEs A and B is depicted in figure 7b.

The R units are used for data distribution. Pixel and y^n values must be delivered to the correct PE by using an adequate routing scheme. R_i can be a simple routing element that receives tokens from a data source and moves a token to the respective processor queue every i tokens it receives, with all other tokens passed to the next routing element R_{i-1}. This next routing element will remove a token every $i-1$ tokens it receives and so on.

PE B_j performs the vector dot product of the columns of pixels $F_{j+k\beta}$ with X (see eq. 6), with k varying from 0 to $\mu-1$. It maintains μ accumulator registers that contain $H_{j+k\beta}$ values after processing the N image lines. While the pixels of the first line are being processed, the output of the multiplexer in figure 7b will be zero. The μ tokens generated at the output of the floating point adder, that take the values $f(1, j+k\beta)$, are routed to the μ accumulator registers through the DEMUX. The μ values of $H_{j+k\beta}$ are updated for the next image lines, by resetting the selection input of the MUX and DEMUX. During the last line, the output of the floating point adder will be routed to the PE A_j, by setting the selection input of the DEMUX represented in figure 7b.

PE A_j performs two different tasks. Firstly, it calculates the partial vector dot product P_j (eq. 11). In this phase, it consumes the μ tokens produced by B_j ($H_{j+k\beta}$ values), and μ tokens from the y^n distribution system $[(j+k\beta)^n]$, with k varying from 0 to $\mu-1$. While the first token is being processed MUX1's selection input is set, which corresponds to reset the accumulator register. In the other phase, that begins after μ tokens have been processed, PE A_j receives a token from PE A_{j-1} that contains the sum of all P_i, with i in the range from $[1\ldots(j-1)]$. This token is used as one of the inputs of the floating point adder by setting the selection input of MUX2. At this time, the other input of the adder has the value currently in the accumulator register, *i.e.* P_j. The output of this adder, that represents the sum of all P_i, with i in the range from $[1\ldots j]$, is routed to PE A_{j+1} through the DEMUX. PE A_β outputs $\mathcal{M}_{m,n}$, according to eq. 12.

$$P_j = \sum_{k=0}^{\mu-1} (j+k\beta)^n H_{j+k\beta} \ \ ; 1 \leq j \leq \beta \qquad (11)$$

$$\mathcal{M}_{m,n} = \sum_{x=1}^{N}\sum_{y=1}^{M} x^m y^n f(x,y) = \sum_{j=1}^{\beta} P_j \qquad (12)$$

Each PE of type B requires a floating-point adder, a floating-point multiplier and $\mu+1$ accumulator registers. Each PE of type A requires a floating-point adder, a floating point-multiplier and a single accumulator register. In each PE, the multiply and accumulate operation is pipelined. The image buffer and the buffers that connect the lower R units should be designed considering the pixel frequency, t_{op} and β.

Disregarding the power core, this architecture requires 2β adders and 2β multipliers. These quantities can be reduced to β adders and β multipliers by noticing that the PEs of type A are only used in the final stage of the calculation. The operations they perform can also be accomplished by the PEs B at the cost of a minor loss of parallelism and additional control logic.

The modified PE B_j operates in three phases, as represented in figure 8. It starts by processing N lines for determining $H_{j+k\beta}$ (phase 1). Then it computes P_j consuming the μ tokens corresponding to $H_{j+k\beta}$ and $(j+k\beta)^n$ values (phase 2). Finally, each PE B_j will add its P_j to the accumulated value of the previous P_i's, received from PE B_{j-1} through the buffers that where used to transmit x^m (phase 3). Then, it will send off a token with the result to PE B_{i+1}. PE B_β will produce a token with the value of $\mathcal{M}_{m,n}$.

The architecture presented in figure 9 uses the modified PEs of type B. The type A PEs were removed from the

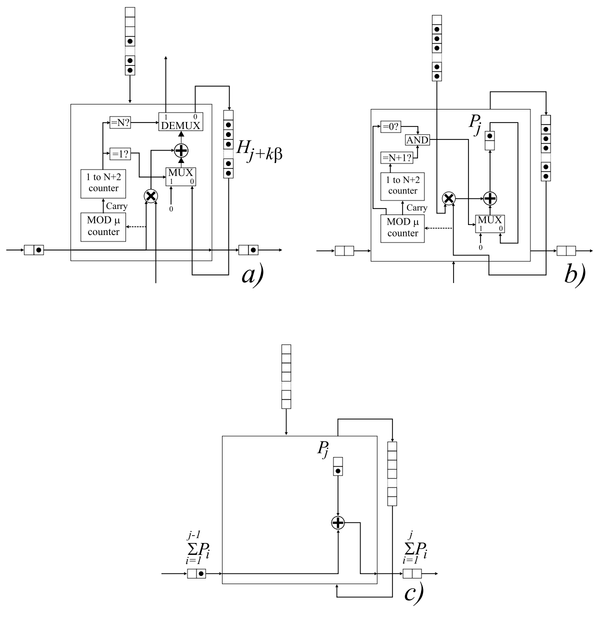
Figure 8: Modified B_j PE: a) phase 1; b) phase 2; c) phase 3.

architecture presented in figure 7, which leads to a considerable reduction of the required hardware.

Since a new x^n value is needed only after μ pixels have been processed, and the y^n values are only needed after M image lines have been processed, there is no need for two power cores. The y^n values can be generated while the PEs are computing H, by interleaving the calculation of y^n with the calculation of x^m. This can be implemented with a single power core, by using the structure presented in figure 10 (W block represents the power core). For the purpose of determining the hardware requirements for this architecture, it was considered that the power core was the one described in section 2. However, this power core can be replaced by a slower and less hardware consuming unit.

For each set of μ tokens processed by the power core, one comes from the x counter and is routed to PE B_1 while the remaining $\mu-1$ come from the y counter and are routed

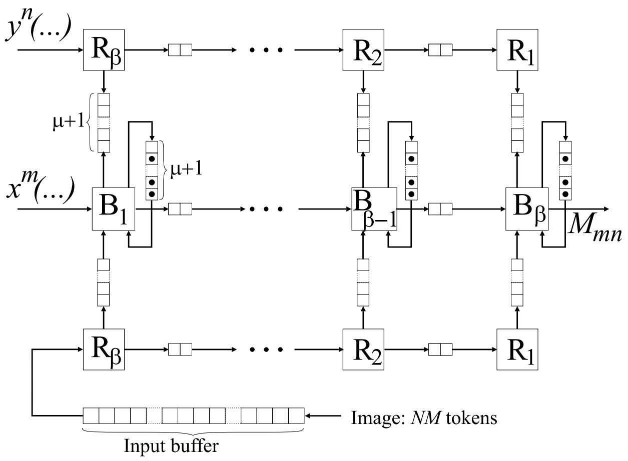
Figure 9: Enhanced architecture for computing $\mathcal{M}_{m,n}$.

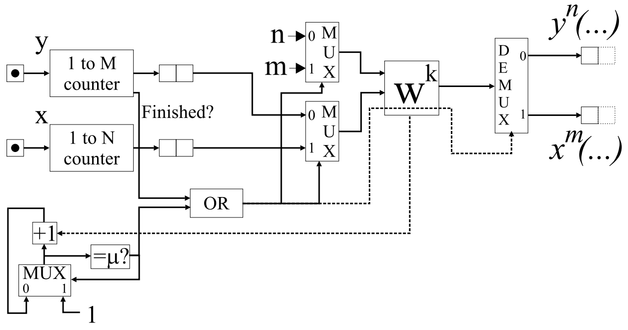

Figure 10: Modified power core.

to the y^n distribution system, marked as $y^n(\ldots)$ in figures 7 and 9. After the y counter reaches M, all the tokens that pass through the power block come from the x counter and are routed to PE B_1 ($x^m(\ldots)$). The x^m values simply propagate through the B PEs. PE B_1 only consumes one token after processing the μ pixels corresponding to an image line.

The complete circuit generates a moment in:

$$T = [t_{pow} + N\mu + \mu + 1 + (\beta - 1)]\, t_{op}, \qquad (13)$$

where t_{pow} is the latency of the power core already provided in section 2. The circuit requires $2k$ floating point multipliers for the power core and β additional floating point multipliers and adders.

5 Pipeline Architectures for Computing a Set of Moments

The PE of type B can be modified to produce a set of moments $\mathcal{M}_{m,0}$, through $\mathcal{M}_{m,n}$. The $n+1$ values are generated reusing the same $H_{j+k\beta}$ (eqs. 14 and 15). This entails an additional delay of $(n\mu + n)\, t_{op}$ corresponding to $n\mu$ multiplication and $(n\mu + n)$ addition operations.

In such a structure, after each PE has calculated the corresponding $H_{j+k\beta}$ set of values, it passes on to a second stage where each $P_j^{(g)}$ value (with $0 \le g \le n$) is generated at a cost of μ multiplications and additions. The H tokens are reinserted in the accumulator buffer after they have been multiplied by $y = j + k\beta$, according to eq. 14, eq. 15 and figure 11a.

$$\mathcal{M}_{m,g} = \sum_{j=1}^{\beta} \sum_{k=0}^{\mu-1} (j + k\beta)^g\, H_{j+k\beta} = \sum_{j=1}^{\beta} P_j^{(g)} \qquad (14)$$

$$P_j^{(g+1)} = \sum_{k=0}^{\mu-1} (j + k\beta) \left[(j + k\beta)^g\, H_{j+k\beta} \right] \qquad (15)$$

After each $P_j^{(g)}$ value is generated at PE B_j, it is added to the output of PE B_{j-1}, in a similar manner to what was done in the computation of a single moment. Then, the modified PE B_j determines the new $P_j^{(g+1)}$ value, adds it to the output of the previous PE B_{j-1}, repeating this process $(n+1)$ times.

This change eliminates the need for a power generator circuit for the y^n calculation, since the first n powers of y are now implicitly generated within each PE. However there is still the need to provide each processor with the adequate values of y. This can be done by the distribution system previously used for y^n or in a setup phase. The later option seems more interesting since it reduces the hardware and communication requirements.

In figure 11 we present the structure for the modified PE of type B. Since phase one has not been changed, only the second and third phases of its activity cycle are represented. In this figure, it is represented the y data coming from the y^n distribution system, because it greatly simplifies the representation. In fact, the setup phase that initializes the counters and the control logic in each PE can be also responsible for filling the y buffer with the μ required tokens.

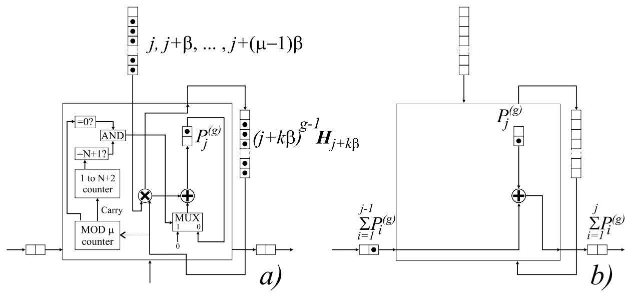
j, j+β, ... , j+(μ−1)β
=0?
AND
=N+1?
1 to N+2 counter
Carry
MOD μ counter
MUX
$P_j^{(g)}$
$(j+k\beta)^{g-1} H_{j+k\beta}$
a)
$\sum_{i=1}^{j-1} P_i^{(g)}$
$\sum_{i=1}^{j} P_i^{(g)}$
b)

Figure 11: PE B for computing a set of moments: a) phase 2; b) phase 3.

The computation time for computing the $n+1$ moments with the modified circuit is:

$$T = [t_{pow} + N\mu + (n+1)\,\mu + (n+1) + (\beta - 1)]\, t_{op}. \quad (16)$$

Apart from additional control logic in the B PE, this circuit requires no more hardware than the previous one.

6 Discussion and Conclusions

This paper proposes a new class of wavefront architectures for computing 2-D gray level image moments of any order, $\mathcal{M}_{m,n}$. These architectures require a single power core with $2k$ multipliers ($k = \lceil \max\{\log_2 m, \log_2 n\} \rceil$), independently of the image size. This core uses a regular and repetitive architectural structure, well adapted to its implementation on a VLSI circuit. Serial and parallel architectures were proposed to compute a single moment. Tables 1 and 2 give the hardware requirements and the computation time of both types of architectures. The amount of FIFO memory required to support the data flow between two PEs is denoted by ϕ in table 1.

For the serial architecture, just $(2k+1)$ multipliers and one adder are required, but the arithmetic units have to be fast enough to accommodate one floating point operation in a sampling interval (t_{op} on table 2). Only $(4 \times k)$ FIFO memories with size ϕ are required in this wavefront architecture. In the parallel architecture, β $(< M)$ PEs can be used to speed up the computation. The power core only has

Architect.	Nr. of multipliers	Nr. of adders	Memory
Serial	$2k+1$	1	$(4k)\phi$
Parallel	$2k+\beta$	β	$2\beta\mu + (4k+4\beta)\phi$
Mom. Set	$2k+\beta$	β	$2\beta\mu + (4k+4\beta)\phi$

Table 1: Hardware required by the proposed architectures.

Architect.	Normalized computation time (T/t_{op})
Serial	$t_{pow} + (M+1)N + 2$
Parallel	$t_{pow} + M(N+1)/\beta + \beta$
Mom. Set	$t_{pow} + M\,(N+n+1)\,/\beta + n + \beta$

Table 2: Computation time of the proposed architectures.

to calculate $M+N$ values, instead of $M \times N$ values, required by the serial architecture. However, approximately $2 \times M$ more memory elements are required. With the introduction of β multipliers and adders, the computation speed has been increased β times. Alternatively, slower arithmetic units may be used to obtain the same performance. Finally, a parallel architecture was proposed for computing a set of moments $\mathcal{M}_{m,x}$, with $0 \leq x \leq n$. Comparing it with the previous parallel architecture, the required hardware is the same and the computation time is increased by approximately (nM/β).

References

[1] G. Agin,"Handbook of Industrial Robotics", chap. *Vision Systems*, *Addison-Wesley*, 1985.

[2] R. Haralick and L. Shapiro,"Computer and Robot Vision", *Addison-Wesley*, vol. II, chap. 18. 1993.

[3] R. Andersson, "Real-Time Gray-Scale Video Processing Using a Moment-Generating Chip", *IEEE Journal of robotics and Automation*, vol. 1, no. 2, june 1985, pp. 79-85.

[4] K. Chen, "Efficient Parallel Algorithms for the Computation of Two-Dimensional Image Moments", *Pattern Recognition*, vol. 23, no. 1/2, 1990, pp. 109-119.

[5] X. Jiang and H. Bunke,"Simple and Fast Computation of Moments", *Pattern Recognition*, vol. 24, no. 8, 1991, pp. 801-806.

[6] H. D. Cheng, C. Y. Wu and D. L. Hung,"VLSI for Moment Computation and its Application to Breast Cancer Detection", *Pattern Recognition*, vol. 31, no. 9, 1998, pp. 1391-1406.

[7] P. Pirsch, "Architectures for Digital Signal Processing", *John Wiley & Sons*, chap.5, 1998.

[8] "XC4000E and XC4000X series Field Programmable Gate Array", *Xilinx Inc.*, June 1997.