A bit-level systolic array for digital contour smoothing
Petkov, Nikolai; Sloboda, Fridrich

Publication date: 1989

Citation for published version (APA):
A bit-level systolic array for digital contour smoothing

Nikolai PETKOV

Academy of Sciences of GDR, Central Institute of Cybernetics and Information Processes, International Basic Laboratory for Image Processing, Kurstr. 33, P.O. Box 1298, Berlin 1086, German Dem. Rep.

Fridrich SLOBODA

Institute of Technical Cybernetics, Slovak Academy of Sciences 842 37 Bratislava, Dubrovska 9, Czechoslovakia

Received November 1988

Abstract. Linear operators for digital contour smoothing are described. These operators are defined by circulant Toeplitz matrices and allow to smooth digital contours in the least-squares sense. They minimize the undersampling, digitizing and quantizing error and allow to calculate invariants, such as curvature, which are not possible to calculate without smoothing. A bit-level systolic array which is capable of realizing the proposed operator is described. This array is easy to implement in VLSI, because the array cells involved are very simple. Furthermore, the array is completely pipelined on the bit-level, so that it operates with a high clock frequency achieving very high throughputs.

Keywords. Digital contour smoothing, circulant Toeplitz matrices, bit-level systolic array, VLSI.

1. Introduction

In [19] linear operators of the form

$$(1/c)C,$$

where $C$ is an $n \times n$ circulant Toeplitz matrix and $c$ is a suitable constant, are investigated. These operators allow to smooth simply closed digital curves in the least-squares sense. They minimize the undersampling, digitizing and quantizing error and allow to calculate invariants, such as curvature, which are not possible to calculate without smoothing [20]. In order to improve the stability of calculated global invariants [2,6,7,12] different techniques have been suggested [22]. For real-time applications the one-dimensional systolic array TOPSS-28 has been developed [14]. The Hughes Research Laboratories Toeplitz systems solver TOPSS-28 allows to perform real-time calculation of splines and to reduce the data required for reference images. A simple algorithm and a pipelined architecture for B-spline interpolation is described in [18].

In this paper a bit-level systolic array for digital contour smoothing is described. Kung and Leiserson [9,10] have introduced the notion of systolic arrays, a special kind of highly regular, homogeneous, pipelined processor arrays. These have been shown to hold a great potential for VLSI implementation and to effectively employ the two major principles of parallel processing, namely multiprocessing and pipelining [11]. Meanwhile a lot of different systolic algorithms and arrays have been proposed for different computational problems. However, little attention
has been paid to the fact that a systolic array can benefit from its regular structure for VLSI design only if its processors (cells) are relatively small and simple. In fact, only systolic arrays which have very simple cells operating on the bit-level—we refer to such arrays as to bit-level arrays—have been implemented in single VLSI chips [3,8,13]. Having a bit-level systolic algorithm, the corresponding systolic array is extremely easy to design. In this case only one cell of few gates should be designed, simulated and tested. The whole array is generated by replicating this cell [4]. Since the bit-level cells are separated by delay elements, the very high clock frequency results in a high throughput of the array. We refer to such arrays as to completely pipelined arrays, as the pipelining is realized on the finest grain, most primitive level, the bit-level. In this paper a completely pipelined, bit-level systolic array for digital contour smoothing is presented and real-time applications for this array are summarized.

2. Digital contour smoothing operators

In the following a digital picture $\Sigma$ is a finite rectangular array whose elements are called points or picture elements. Each point $P$ of $\Sigma$ is defined then by a pair of Cartesian coordinates $(x, y)$, which we may take to be integer valued. A point $P = (x, y)$ in a digital picture $\Sigma$ has two types of neighbors:

(a) its four horizontal and vertical neighbors $(u, v)$ such that $|x - u| + |y - v| = 1$,  
(b) its diagonal neighbors $(u, v)$ such that $|x - u| = |y - v| = 1$.

We shall refer to the neighbors of type (a) as 4-neighbors of $P$, and to the neighbors of both type, as 8-neighbors [16,17]. A simply closed digital curve is a path $\gamma = P_0, P_1, \ldots, P_n$ such that [17]

(a) $P_i = P_j$ iff $i = j$, and  
(b) $P_i$ is a neighbor of $P_j$ iff $i = j + 1 (\mod n + 1)$.

Let $P_{k,n}(i), \ k = 0, 1, 2, \ldots, m < n$ be a set of orthogonal polynomials defined on the equidistant set of points $0, 1, 2, \ldots, n$ where $k$ denotes the degree of the given polynomial. Let us suppose that these polynomials have the form

$$P_{0,n}(i) = 1, \quad P_{1,n}(i) = 1 - \frac{2i}{n}, \quad P_{2,n}(i) = 1 - \frac{6i}{n - 1} + \frac{6i^2}{n(n-1)},$$

$$iP_{m,n}(i) = - \left( \frac{m+1}{2(2m+1)} \right) P_{m+1,n}(i) + \frac{n}{2} P_{m,n}(i) - \frac{m(n+m+1)}{2(2m+1)} P_{m-1,n}(i).$$

Let $f : R_1 \rightarrow R_1$ be defined on the set of points $0, 1, 2, \ldots, n$ and let the values of $f$ on these points be $f(i), \ i = 0, 1, 2, \ldots, n$. Let the approximation function have the form

$$\Phi(i) = P_m(i) = c_0 P_{0,n}(i) + c_1 P_{1,n}(i) + \cdots + c_m P_{m,n}(i)$$

and let this function approximate the function $f$ in the least-squares sense, i.e.,

$$\sum_{i=0}^{n} [f(i) - \Phi(i)]^2 = \text{min}!$$

According to the orthogonality of the polynomials, the coefficients $c_j$ are defined by [1]

$$c_j(P_{j,n}, P_{j,n}) = (f, P_{j,n}), \quad j = 0, 1, \ldots, m$$
where
\[
    c_j = \frac{\sum_{i=0}^{n} f(i) P_{j,n}(i)}{\sum_{i=0}^{n} P_{j,n}^2(i)} = \frac{(2j+1)n^{(j)}}{(j+n+1)^{(j+1)}} \sum_{i=0}^{n} f(i) P_{j,n}(i), \quad j = 0, 1, \ldots, m
\]  \hspace{1cm} (5)

and
\[
    n^{(j)} = n(n-1) \cdots (n-j+1),
\]
\[
    (j+n+1)^{(j+1)} = (j+n+1)(j+n) \cdots (n+1).
\]

Let us consider N + 1 function values f(i) defined on the equidistant set of points 0, 1, 2, \ldots, N, where N ≫ n. Let the function f be approximated on each subset consisting of n + 1 points by a polynomial P_{m}. Let m be odd and let n be even. Let us denote the running subset of n + 1 points by i − n/2, \ldots, i, \ldots, i + n/2. Then the smoothed value of f in the midpoint i is defined by the value of \Phi(i) = P_{m}(i).

Let
\[
    x = x(t), \quad y = y(t)
\]
be a simply closed curve in the 2-dimensional Euclidean space \mathbb{R}^2. Let this curve be approximated by a set of N points P_1 = (x_1, y_1), P_2 = (x_2, y_2), \ldots, P_N = (x_N, y_N) which are elements of a finite rectangular array N, and let these points represent a simply closed 4-connected digital curve for which
\[
    || P_i - P_{i-1} ||= |x_i - x_{i-1}| + |y_i - y_{i-1}| = 1.
\]

Let us denote by
\[
    X = \begin{bmatrix}
        x_1 & y_1 \\
        x_2 & y_2 \\
        \vdots & \vdots \\
        x_N & y_N
    \end{bmatrix}
\]
the array of all contour points. The least-squares smoothing of a simply closed digital curve is then defined by the linear operator \((1/c)C\) which is applied on X:
\[
    (1/c)CX
\]

where C is an N × N circulant Toeplitz matrix and c is the sum of all elements in a row of C, whereby the coefficients of the matrix C are defined by (5). For different values of m and n + 1 we obtain the following operators [19]:
- m = 1, n + 1 = 3:
\[
    \frac{1}{c} C = \frac{1}{3} \begin{bmatrix}
        1 & 1 & 1 \\
        1 & 1 & 1 \\
        1 & 1 & 1
    \end{bmatrix};
\]  \hspace{1cm} (6)

- m = 1, n + 1 = 5:
\[
    \frac{1}{c} C = \frac{1}{5} \begin{bmatrix}
        1 & 1 & 1 & 1 & 1 \\
        1 & 1 & 1 & 1 & 1 \\
        1 & 1 & 1 & 1 & 1 \\
        1 & 1 & 1 & 1 & 1 \\
        1 & 1 & 1 & 1 & 1
    \end{bmatrix}
\]  \hspace{1cm} (7)
A smoothed contour by operator (8) is shown in Fig. 1(a) and by operator (10) in Fig. 1(b).

A subset of linear operators defined by an \( N \times N \) circulant Toeplitz matrix \( C \) which smooth digital closed contours in the least-squares sense is suitable for digital contour approximation and these operators will be called feasible [19]. Let us denote

\[
(1/c)C = X' \quad \text{for} \quad X' = \text{least-squares smoothing}.
\]

(11)

where \( X' \) has elements \( x'_i \) and \( y'_i \). Then a linear operator \((1/c)C\), where \( C \) is an \( N \times N \) circulant Toeplitz matrix whose elements are defined by (5), is feasible if

\[
|x_i - x'_i| < \frac{1}{3}, \quad |y_i - y'_i| < \frac{1}{3}.
\]

(12)
According to this definition a feasible operator is defined by the constrained least-squares smoothing with box constraints (12). It has been shown that (6), (9), (10) are feasible operators [19]. These operators generate points which lie in a corridor defined by (12), in which also the original curve (see Fig. 2) approximated by the 4-connected closed digital curve lies.

The generalization of (1)–(5) for digital contour smoothing is based on the following equivalence:

Let \( r_i \) denote the radius vector defined by the point \( P_i = (x_i, y_i) \) and by \( O = (0, 0) \) the initial point. Let us denote

\[
\| P_i - P_0 \| = |x_i - x_0| + |y_i - y_0| = i,
\]

(13)

\[
\| r_i \| = |x_i - 0| + |y_i - 0|.
\]

(14)

Then \((1/c)CX\) is equivalent to the one-dimensional smoothing defined by (1)–(5) on the equidistant set of points defined by (13) and with function values \( f_i \) defined by (14), see Fig. 3.

The linear operators described above allow to smooth simply closed digital curves provided that they are defined by uniformly spaced points. They minimize the undersampling, digitizing...
and quantizing error and so allow to improve the stability of calculated local and global invariants, and enable to calculate invariants which are not possible to calculate without smoothing.

3. Completely pipelined bit-level systolic array

Let $X$ be an $N \times 2$ matrix which represents a simply closed 4-connected digital curve. The elements of this matrix are defined by the coordinates of $N$ contour points which are integer values; let this matrix have the form

$$X = \begin{bmatrix}
x_1 & y_1 \\
x_2 & y_2 \\
\vdots & \vdots \\
x_N & y_N
\end{bmatrix}.$$ 

Let us denote

$$x = (x_1, x_2, \ldots, x_N)^T, \quad y = (y_1, y_2, \ldots, y_N)^T.$$

In the following a bit-level systolic array will be described which performs the matrix-vector multiplication

$$u = Cx$$

where $C$ is defined by (10). The matrix-matrix multiplication $CX$ can be then performed by two matrix-vector multiplications $Cx$ and $Cy$. Let us denote the non-zero elements of $C$ by

$$a_{-3} = -2, \quad a_{-2} = 3, \quad a_{-1} = 6, \quad a_0 = 7, \quad a_1 = 6, \quad a_2 = 3, \quad a_3 = -2.$$ 

The matrix-vector multiplication can be represented by the circulant convolution

$$u_i = \sum_{j=-3}^{3} a_j x_{i-(j) \bmod N}, \quad i = 1, 2, \ldots, N$$

where $u_i$ is the $i$th component of $u$ and $x_0 = x_N$. A word-level systolic array for this transformation is shown in Fig. 4. The array cells in Fig. 4(a) are combinational circuits which are capable of executing the so-called inner product step in one clock period. Since the primitive operations of this algorithm are operations on the word level, the array is qualified as a word-level systolic array and the parallelism involved is called word-level parallelism. The small bars in Fig. 4(b) denote delay elements which are controlled by a common clock. The
The necessary input operations are shown in Fig. 4(c). \( z_i^n, i = 1, 2, \ldots, N \) denote the initial starting values of the components of the vector \( z \).

The circulant Toeplitz matrix (10) can be represented by the sum of two matrices

\[
C = C^{(1)} + C^{(2)}
\]

with the coefficients \( a_j^{(1)} \) and \( a_j^{(2)} \), respectively,

\[
a_j = a_j^{(1)} + a_j^{(2)}, \quad j = -3, -2, \ldots, 2, 3
\]

which are powers of 2, so that the corresponding multiplications can be carried out by shifting the \( x \)-data. The coefficients will be represented by the following powers of 2:

<table>
<thead>
<tr>
<th>( j )</th>
<th>(-3)</th>
<th>(-2)</th>
<th>(-1)</th>
<th>(0)</th>
<th>(1)</th>
<th>(2)</th>
<th>(3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_j )</td>
<td>(-2)</td>
<td>(3)</td>
<td>(6)</td>
<td>(7)</td>
<td>(6)</td>
<td>(3)</td>
<td>(-2)</td>
</tr>
<tr>
<td>( a_j^{(1)} )</td>
<td>0</td>
<td>(2^0)</td>
<td>(2^1)</td>
<td>(-2^0)</td>
<td>(2^1)</td>
<td>(2^0)</td>
<td>(-2^1)</td>
</tr>
<tr>
<td>( a_j^{(2)} )</td>
<td>(-2^1)</td>
<td>(2^1)</td>
<td>(2^2)</td>
<td>(2^3)</td>
<td>(2^2)</td>
<td>(2^1)</td>
<td>0</td>
</tr>
</tbody>
</table>

The coefficients are chosen in such a way that the neighboring coefficients are represented by the neighboring powers of 2. This corresponds to shifting by just one position in the neighboring calls of the corresponding array. According to this, the systolic array shown in Fig. 4 can be realized by two systolic arrays of the same kind which correspond to the matrices \( C^{(1)} \) and \( C^{(2)} \), respectively, and which are chained together as shown in Fig. 5.
2 × 3 + 1 delay elements are arranged in the \( x \)-connections between the two arrays in order to compensate the greater number of delay elements in the \( z \)-connections of the first array and to provide the proper input timing of the data entering the second array (data alignment). The resulting structure is shown on Fig. 6.

\[
\text{procedure bit-level cell("full adder")}
\]

\[
\text{(all variables : boolean )}
\]

\[
\begin{align*}
\text{begin} \\
& z := (z + x + c) \mod 2, \text{("sum")} \\
& c := xz \text{ or } xc \text{ or } zc, \text{("carry")} \\
& x := x \\
\text{end}
\end{align*}
\]

Fig. 6.

Fig. 7.
In the following the realization of the inner product step on the bit level will be described. Figure 7 shows the implementation of the first two array cells which themselves are realized as linear, vertical, bit-level arrays. The multiplication by $2^1$ in the second cell is realized by shifting the input $x$-data upwards by one bit position. The addition is carried out in a carry-ripple array of full adders. For 8-bit input data and according to the sum

$$a_j = 2^1, \quad j = -3 \ldots 3$$

the intermediate result will be represented by $(8 + \lfloor \log_2 21 \rfloor)$-bit numbers. Hence, one word-level cell, i.e. one column of the bit-level array, consists of $(8 + \lfloor \log_2 21 \rfloor) = 13$ bit-level cells (full adders). The clock period of this system is controlled by the carry-rippling delay from the LSB to the MSB cell, which is readily estimated to be $13t$ where $t$ is the carry delay of a single full adder. To achieve shorter clock period, the carry rippling was eliminated by coupling the full adders lying in one column via delay elements, forming so a vertical pipeline. The input data are skewed (see Fig. 8).

Notice, that due to the shifting of the $x$-data upwards, a retiming is necessary by removing one of the delay elements in each of the $z$-connections in order to provide data alignment of the $x$- and $z$-bits in a given cell. The clock period of this completely pipelined system is controlled by the delay $t$ of a single bit-level cell and it is 13 times shorter than the clock period of the non-pipelined system.

In the next the realization of a multiplication with a negative coefficient is described. In the third column of the array we have $a_0^{(1)} = -2^0$. The multiplication by $-1$ is realized by converting the input $x$-words into their 2's complements before their addition. This can be
Fig. 9. First four columns of the bit-level array.

Fig. 10.
done quite easily by inverting the corresponding x-inputs of the full adders (see Fig. 9, where the small circles denote inverters), and adding a 1 via the carry input of the LSB full adder.

The whole array is a regular matrix of full adders arranged in 12 columns (corresponding to the 12 coefficients) and 13 rows (corresponding to the bits of different significance). In Fig. 10 the profile of the x- and z-connections, the number of the delay elements which are arranged in these connections, and the positions at which inverters (shown as small circles) are arranged in the x-inputs of the full adders are shown. The number of delay elements in the carry interconnections is constant for the whole array: one delay element in each carry interconnection. These connections are not considered in Fig. 10. The number of delay elements in the x- and z-interconnections is derived from the corresponding number of delay elements in the original word-level array (Fig. 6) according to the following rule: a delay element is removed from the z-connection or x-connection between two cells of the bit-level array, if the x-data are shifted upwards or downwards between these cells, respectively.

The directions of the carry and z-connections are constant for the whole array and these connections are local. The direction of the x-connections is constant within one column, but changes from column to column. However, all x-connections are local due to the fact that the x-data are shifted maximal by one bit position between two neighboring columns. Inverters are arranged in the x-inputs of the full adders in the third, sixth and the seventh column. The general technique for the bit-level systolic arrays design for the circulant convolution is presented in [15].

The smoothing operator defined by (10) requires also a normalization by \( c = 21 \). This can be done in a straightforward way but it depends on the application in which moment it will be performed. In the next section it is shown that this will be performed as one division after the invariants have been calculated from the array \( CX \). Another approach is to use a fast approximation algorithm which takes advantage from the particular value of the denominator [15].

4. Applications

Let \( a_i \) be an arbitrary linear segment in the 2-dimensional Euclidean space \( R_2 \), see Fig. 11. The area of a closed polygon represented by the string \( a_1, a_2, \ldots, a_N \) can be calculated by [5]

\[
P = \left| P_X \right| \quad \text{where} \quad P_X = \sum_{i=1}^{N} a_{ix} (y_i + \frac{1}{2} a_{iy}).
\]

For the smoothed closed curve \((1/c)CX\) we have

\[
P_{(1/c)CX} = c^{-2} P_{CX}.
\]

For the first-order moment \( M_{1x}^X \) of the array \( X \) according to the x-axis it holds

\[
M_{1x}^X = \sum_{i=1}^{N} \frac{1}{2} a_{ix} \left( y_i^2 + a_{iy} \left( y_i + \frac{1}{2} a_{iy} \right) \right)
\]

and for the smoothed closed curve we obtain

\[
M_{1x}^{(1/c)CX} = c^{-1} M_{1x}^{CX}.
\]

According to this the normalization defined by division by \( c \) will be performed after the area and the first-order moments have been calculated from the array \( CX \). Similarly we can derive formulas also for the second-order moments [5]. The linear operators described above minimize the undersampling, digitizing and quantizing error and so they are able to improve the stability
of calculated local and global invariants such as the area, center of gravity defined by the first-order moments and invariants defined by the second-order moments [2,6,7,12], the length of a curve, and enable to calculate invariants such as curvature [20] which are not possible to calculate without smoothing. For length and curvature calculation of smoothed digital curves efficient string processing algorithms have been suggested [21]. In [22] another approach for moment invariants calculation is described.

5. Conclusion

Linear operators for digital contour smoothing are described. These operators are defined by circulant Toeplitz matrices and enable to smooth digital contours in the least-squares sense. A bit-level systolic array which is capable of realizing the proposed operator is presented. Since the array cells are very simple and only one cell type is used, the array is extremely easy to implement in VLSI. The array is completely pipelined on the bit-level, so that it operates at high clock frequency achieving very high throughputs and is devoted to real-time applications. Another example of a bit-level systolic array exploitation is described in [15].

References


