Fast Evaluation of Polynomials and Inverses of Polynomials

Xavier MERRHEIM, Jean-Michel MULLER and Hong-Jin YEH

CNRS, Lab. LIP-IMAG
Ecole Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, FRANCE

This work is partially supported by the "PRC Architectures de Machines Nouvelles" of the French "Ministère de la Recherche et de l'Espace"

Abstract

We deal with the parallel and on-line (i.e. digit serial, most significant digit first) evaluation of polynomials and inverses of polynomials. We propose new algorithms and architectures for such evaluations. A 3-D implementation model is proposed.

1 Introduction

Fast and accurate evaluation of polynomials is a major goal in computer arithmetic, since any continuous function can be approximated by a polynomial as accurately as desired [1]. As a matter of fact, polynomial approximations have been widely used for software implementation of elementary functions [2, 3]. Due to the advances in VLSI technology, fast circuits evaluating polynomials can now be implemented. The most common approach for evaluating polynomials in hardware is based on the use of MA (Multiply-Add) cells [4]: Horner's scheme using a linear array of MA cells, "divide-and-conquer" using a tree structure of MA cells [5]. Thus, many researchers have focused on the design of efficient MA circuits [6, 7, 5, 8, 9] including digit-serial (or, bit-serial) ones [10, 11, 12].

In this paper, it is assumed that all the coefficients of a polynomial are known in advance, so they can be "prepared" (for instance, multiplied by a small integer) for the computation. We present algorithms for evaluating a polynomial $P(x)$ when the value of x is received either in parallel at the beginning of the computation, or serially, starting from the most significant digit (i.e. *on-line* [13]). Besides it is explained how the inverse of polynomials can be computed in parallel or in on-line mode using the same architecture.

Throughout this paper, we suppose that the numbers are represented in a fixed-point radix-2 redundant number system, with digits equal to -1, 0 or 1. Such a *signed-digit* number system makes it possible to perform additions in parallel, without carry propagations, within a time independent of the length of operands (i.e. their precision [14]). Without loss of generality, we suppose in this paper that the image of polynomials for $x \in [-1,1]$ is also in the interval $[-1,1]$. Extending our approach to higher radices signed-digit number systems or simply to carry-save representations will be straightforward (except for the case of on-line evaluation of inverses of polynomials), and is left to the reader.

2 Parallel evaluation of polynomials

Before giving our method for polynomials of degree n, let us study the case of degree-3 polynomials. Assume that we want to evaluate at point $x \in [-1,1]$ a polynomial $P(x) = a_3x^3 + a_2x^2 + a_1x + a_0$. Let us denote: $x = \sum_{i=1}^{\infty} d_i 2^{-i}$ ($d_i = -1,\ 0,\ 1$) and $x_k = \sum_{i=1}^{k} d_i 2^{-i}$. x_{k+1} is equal to $x_k + d_{k+1}2^{-k-1}$, therefore, since P is equal to its order-3 Taylor expansion:

$$\begin{aligned} P(x_{k+1}) &= P(x_k) + d_{k+1}2^{-k-1}P'(x_k) \\ &\quad +\frac{1}{2}d_{k+1}^2 2^{-2k-2}P''(x_k) \\ &\quad +\frac{1}{6}d_{k+1}^3 2^{-3k-3}P'''(x_k) \end{aligned}$$

The polynomials P' and P'' also satisfy similar relations:

$$\begin{aligned} P'(x_{k+1}) &= P'(x_k) + d_{k+1}2^{-k-1}P''(x_k) \\ &\quad +\frac{1}{2}d_{k+1}^2 2^{-2k-2}P'''(x_k) \\ P''(x_{k+1}) &= P''(x_k) + d_{k+1}2^{-k-1}P'''(x_k) \end{aligned}$$

Therefore, since $P'''(x) = 6a_3$, if we note $P_k = P(x_k)$, $P'_k = P'(x_k)$ and $P''_k = P''(x_k)$, we obtain

$$\left\{ \begin{aligned} P_{k+1} &= P_k + d_{k+1}2^{-k-1}P'_k \\ &\quad +d_{k+1}^2 2^{-2k-3}P''_k \\ &\quad +d_{k+1}^3 2^{-3k-3}a_3 \\ P'_{k+1} &= P'_k + d_{k+1}2^{-k-1}P''_k \\ &\quad +d_{k+1}^2 2^{-2k-2}(3a_3) \\ P''_{k+1} &= P''_k + d_{k+1}2^{-k-1}(6a_3) \end{aligned} \right. \qquad (1)$$

Relation (1) makes it possible to compute P_1, P_2, P_3, ... successively, starting from $P_0 = a_0$, $P'_0 = a_1$

and $P_0'' = 2a_2$. Since P_k goes to $P(x)$ as k goes to infinity, we have obtained an iterative method for evaluating P at point x. Assuming that $3a_3$ is stored, the computation of $(P_{k+1}, P'_{k+1}, P''_{k+1})$ from (P_k, P'_k, P''_k) only needs additions and shifts, and can be performed in a constant time (independent of the length of operands) if we use the carry-free-addition ability of signed-digit (or, carry-save) arithmetic. And yet, it is possible to simplify relation (1), in order to suppress the variable shifts. Let us denote

$$\begin{cases} \Phi_k^{(0)} = P_k \\ \Phi_k^{(1)} = 2^{-k} P'_k \\ \Phi_k^{(2)} = \frac{2^{-2k}}{(2)!} P''_k = 2^{-2k-1} P''_k \\ \Phi_k^{(3)} = \frac{2^{-3k}}{(3)!} P'''_k = 2^{-3k} a_3 \end{cases}$$

Then, we obtain

$$\begin{pmatrix} \Phi_{k+1}^{(0)} \\ \Phi_{k+1}^{(1)} \\ \Phi_{k+1}^{(2)} \\ \Phi_{k+1}^{(3)} \end{pmatrix} = \begin{pmatrix} 1 & \frac{d_{k+1}}{2} & \frac{d_{k+1}^2}{4} & \frac{d_{k+1}^3}{8} \\ 0 & \frac{1}{2} & \frac{d_{k+1}}{2} & \frac{3d_{k+1}^2}{8} \\ 0 & 0 & \frac{1}{4} & \frac{3d_{k+1}}{8} \\ 0 & 0 & 0 & \frac{1}{8} \end{pmatrix} \begin{pmatrix} \Phi_k^{(0)} \\ \Phi_k^{(1)} \\ \Phi_k^{(2)} \\ \Phi_k^{(3)} \end{pmatrix}$$

Therefore, the variable-sized shifts in (1) are suppressed.

Now, let us turn to the general case. Assume that P has degree n, and let us denote $P_k^{(i)}$ the value of the i^{th} derivative of P at point x_k. From Taylor's expansion of the derivatives of P, we deduce

$$P_{k+1}^{(i)} = \sum_{j=0}^{n-i} d_{k+1}^j 2^{-j(k+1)} \frac{1}{(j)!} P_k^{(i+j)} \qquad (2)$$

If we define $\Phi_k^{(i)} = 2^{-ik} \frac{1}{(i)!} P_k^{(i)}$ then

$$\Phi_{k+1}^{(i)} = \sum_{j=0}^{n-i} d_{k+1}^j 2^{-i-j} \frac{(i+j)!}{(i)!(j)!} \Phi_k^{(i+j)} \qquad (3)$$

Define the coefficients $C_{ij} = 2^{-i-j}\frac{(i+j)!}{(i)!(j)!}$. These values for $i + j \leq 7$ are shown in Table 1. It is noted that relation (3) is very convenient for evaluating polynomials of small degree, because the coefficients C_{ij} are sums of a few number of powers of 2. For instance, the coefficients appearing when evaluating a polynomial of degree 5 (i.e. the coefficients C_{ij} such that $i+j \leq 5$) are equal to $\frac{1}{32}$, $\frac{5}{32}$ or $\frac{5}{16}$, and a multiplication by 5 reduces to an addition, since $5 = 2^2+1$. For polynomials of degree greater than 7, Table 1 shows that our method cannot be efficiently used, since the coefficients C_{ij} become too complex. However, this problem partially vanishes using the architecture proposed in section 6. In the following sections, we will represent the parallel polynomial evaluator implementing (3) as shown in Figure 1.

i,j	0	1	2	3	4	5	6	7
0	1	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{64}$	$\frac{1}{128}$
1	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{3}{8}$	$\frac{1}{4}$	$\frac{5}{32}$	$\frac{3}{32}$	$\frac{7}{128}$	–
2	$\frac{1}{4}$	$\frac{3}{8}$	$\frac{3}{8}$	$\frac{5}{16}$	$\frac{15}{64}$	$\frac{21}{128}$	–	–
3	$\frac{1}{8}$	$\frac{1}{4}$	$\frac{5}{16}$	$\frac{5}{16}$	$\frac{35}{128}$	–	–	–
4	$\frac{1}{16}$	$\frac{5}{32}$	$\frac{15}{64}$	$\frac{35}{128}$	–	–	–	–
5	$\frac{1}{32}$	$\frac{3}{32}$	$\frac{21}{128}$	–	–	–	–	–
6	$\frac{1}{64}$	$\frac{7}{128}$	–	–	–	–	–	–

Table 1: The values of C_{ij} for $i + j \leq 7$

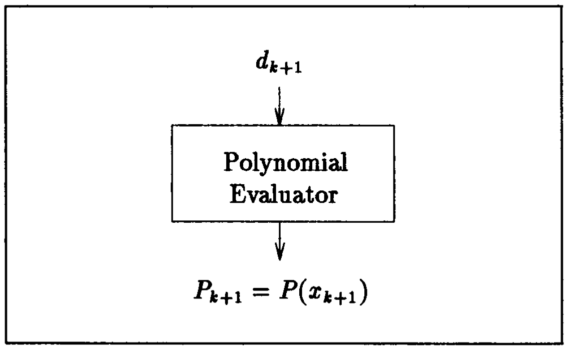

Figure 1: A parallel polynomial evaluator

3 Parallel evaluation of inverses of polynomials

Assume that x belongs to [-1,1]. Define a sequence (x_k) as:

$$\begin{cases} x_0 = 0 \\ x_{k+1} = x_k + d_{k+1} 2^{-k-1} \end{cases}$$

$$\text{with } d_{k+1} = \begin{cases} 1 & \text{if } x_k \leq x \\ -1 & \text{if } x_k > x \end{cases}$$

Then, x_k goes to x as k goes to infinity. The proof is obvious: it suffices to show the relation $|x_k - x| \leq 2^{-k}$ by induction.

We assume that P is (strictly) monotonous in $[-1, 1]$, without loss of generality, we can suppose that P is increasing. Thus, $x_k \leq x$, if and only if $P_k = P(x_k) \leq y$. Then, we are able to compute $x = P^{-1}(y)$ from y by:

$$\begin{cases} x_0 = 0 \\ x_{k+1} = x_k + d_{k+1} 2^{-k-1} \end{cases} \qquad (4)$$

$$\text{with } d_{k+1} = \begin{cases} 1 & \text{if } P_k \leq y \\ -1 & \text{if } P_k > y \end{cases}$$

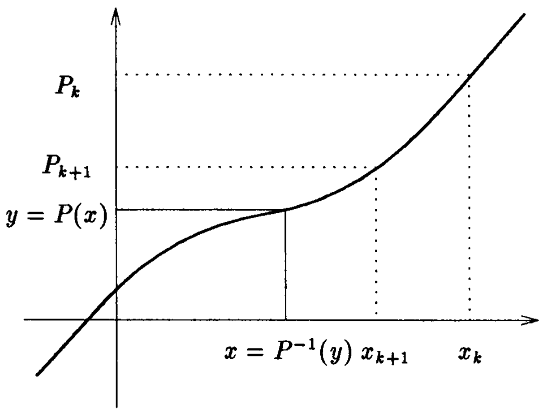

Figure 2: The graph of $y = P(x)$

The terms $P_k = P(x_k)$ are computed as in the previous section. As a matter of fact, the comparison between y and P_k is not convenient at all, since it may require the examination of an arbitrarily large number of digits of y and P_k. Moreover, in the last section, dedicated to the on-line evaluation of P^{-1}, this comparison will not be possible, since y will not be exactly known before the end of the computation.

We assume that $\hat{x}_k = \sum_{i=1}^{k} \hat{d}_i 2^i$ is an approximation of the value x and that $\hat{P}_k = P(\hat{x}_k)$ is computed from $\hat{x}_k$ using (3). Let us define the largest integer, s and the smallest integer, r such that $0 < 2^s \leq P'(x) \leq 2^r$, for all $x \in [-1, 1]$. Then,

$$|\hat{P}_k - y| \leq |\hat{x}_k - x| \cdot \max |P'(x)| \leq 2^{-k+r}$$

and, for $y \in [-1, 1] \cap P([-1, 1])$,

$$\begin{aligned} |\hat{x}_k - x| &\leq |\hat{P}_k - y| \cdot \max |(P^{-1})'(y)| \\ &\leq |\hat{P}_k - y| \cdot \frac{1}{\min |P'(x)|} \\ &\leq |\hat{P}_k - y| \cdot 2^{-s} \end{aligned}$$

To simplify the comparison between $\hat{P}_k$ and y, we can rewrite the above algorithm as follows:

$$\begin{cases} \hat{x}_0 = 0 \\ \hat{x}_{k+1} = \hat{x}_k + \hat{d}_{k+1} 2^{-k-1} \end{cases} \qquad (5)$$

$$\text{with } \hat{d}_{k+1} = \begin{cases} 1 & \text{if } \hat{P}_k < y \\ 0 & \text{if } |\hat{P}_k - y| \leq 2^{s-k-1} \\ -1 & \text{if } \hat{P}_k > y \end{cases}$$

Let us show by induction that, for any k:

$$|\hat{x}_{k+1} - x| \leq 2^{-k-1}$$

Initially, $|\hat{x}_0 - x| = |x| \leq 1$ and let us assume that $|\hat{x}_k - x| \leq 2^{-k}$.

- if $\hat{d}_{k+1} = 1$, $\hat{P}_k - y < 0$ implies that
$$-2^{-k} \leq \hat{x}_k - x \leq 0$$
Therefore, $-2^{-k} + 2^{-k-1} \leq \hat{x}_k + 2^{-k-1} - x \leq 2^{-k-1}$, thus, $|\hat{x}_{k+1} - x| \leq 2^{-k-1}$.
- if $\hat{d}_{k+1} = -1$, $\hat{P}_k - y > 0$ implies that
$$0 \leq \hat{x}_k - x \leq 2^{-k}$$
Therefore, $-2^{-k-1} \leq \hat{x}_k - 2^{-k-1} - x \leq 2^{-k} - 2^{-k-1}$, thus, $|\hat{x}_{k+1} - x| \leq 2^{-k-1}$.
- if $\hat{d}_{k+1} = 0$ then obviously $\hat{P}_{k+1} = \hat{P}_k$ from (2). Since $|\hat{P}_k - y| \leq 2^{s-k-1}$,
$$\begin{aligned} |\hat{x}_{k+1} - x| &\leq |\hat{P}_{k+1} - y| \cdot 2^{-s} \\ &\leq |\hat{P}_k - y| \cdot 2^{-s} \\ &\leq 2^{s-k-1} \cdot 2^{-s} = 2^{-k-1} \end{aligned}$$

Let K be the integer obtained by truncating

$$(\hat{P}_k - y) \cdot 2^{-s+k+1}$$

after the radix point. Then,

$$|K| \leq 2^{-k} \cdot 2^r \cdot 2^{-s+k+1} + 1 < 2^{r-s+2}$$

The comparison between $\hat{P}_k$ and y is reduced to the examination of $r - s + 3$ digits of K. Thus, the algorithm becomes:

$$\begin{cases} \hat{x}_0 = 0 \\ \hat{x}_{k+1} = \hat{x}_k + \hat{d}_{k+1} 2^{-k-1} \end{cases}$$

$$\text{with } \hat{d}_{k+1} = \begin{cases} 1 & \text{if } K \leq -1 \\ 0 & \text{if } K = 0 \\ -1 & \text{if } K \geq 1 \end{cases}$$

4 On-line evaluation of polynomials

Our algorithm makes it possible to evaluate polynomials in on-line mode, by using a ***digitization*** process. At step k, we suppose that we have computed an approximation P_k of the value y from x_k using (3). Let us define the smallest integer, r such that $2^r \geq \max |P'(x)|$, $x \in [-1, 1]$. Then,

$$|P_k - y| \leq |x_k - x| \cdot \max |P'(x)| \leq 2^{-k+r}$$

From P_k, we want to deduce a new digit (as a matter of fact, the digit of weight 2^{-k+r+2}) of the on-line result of the computation, $y = P(x)$. The algorithm

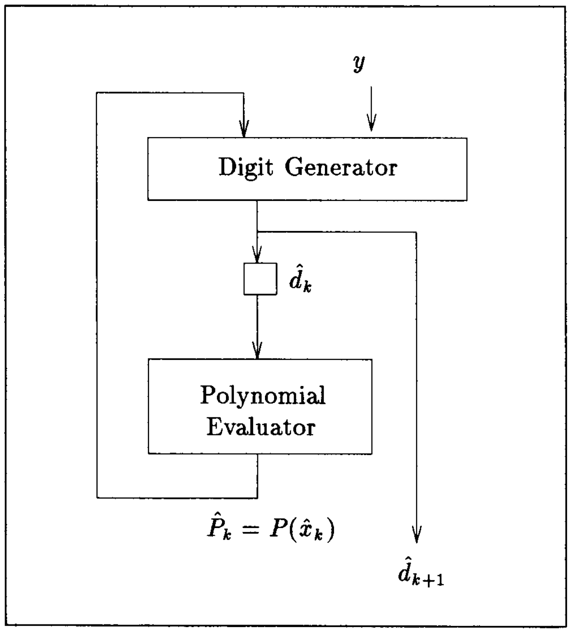

Figure 3: A parallel inverse evaluator

presented below will generate in on-line mode the successive digits of y in the radix-2 signed-digit number system.

Assume that we are at step $k+2$ of our algorithm. We have computed a value P_{k+2} satisfying $|P_{k+2}-y| \leq 2^{-k-2+r}$. From P_1, P_2, $\cdots$, P_{k+1}, we have deduced c_1, c_2, $\cdots$, c_{k-1-r}. The number Y_{k+2} is defined as the number obtained by truncating P_{k+2} after the $(k+2-r)$ position satisfying

$$\begin{aligned} |Y_{k+2}-y| &\leq |Y_{k+2}-P_{k+2}| + |P_{k+2}-y| \\ &\leq 2^{-k-2+r} + 2^{-k-2+r} \\ &\leq 2^{-k-1+r} \end{aligned} \tag{6}$$

Let us denote $\Psi = 0.c_1c_2\cdots c_{k-1-r}$. The interval I_{-1} of the numbers representable if we choose $c_{k-r} = -1$ is $[\Psi - 2^{-k+r+1}, \Psi]$, the interval I_0 of the numbers representable with $c_{k-r} = 0$ is $[\Psi - 2^{-k+r}, \Psi + 2^{-k+r}]$, and the interval I_1 of the numbers representable with $c_{k-r} = 1$ is $[\Psi, \Psi + 2^{-k+r+1}]$. From that we deduce:

- If $Y_{k+2} \leq \Psi - 2^{-k-1+r}$ then, from (6), $y \leq \Psi$ thus $y \in I_{-1}$: we choose $c_{k-r} = -1$.
- If $\Psi - 2^{-k-1+r} \leq Y_{k+2} \leq \Psi + 2^{-k-1+r}$ then, from (6), $\Psi - 2^{-k+r} \leq y \leq \Psi + 2^{-k+r}$ thus $y \in I_0$: we choose $c_{k-r} = 0$.
- If $Y_{k+2} \geq \Psi + 2^{-k-1+r}$ then, from (6), $y \geq \Psi$ thus $y \in I_1$: we choose $c_{k-r} = 1$.

This algorithm is easily implementable since it needs the examination of only 5 digits of Y_{k+2}. Let us call K the integer $2^{k+2-r}(\Phi - Y_{k+2})$. From (6) and the obvious relation $|\Psi - y| \leq 2^{-k+1+r}$,

$$\begin{aligned} |\Psi - Y_{k+2}| &\leq |\Psi - y| + |Y_{k+2} - y| \\ &\leq 2^{-k+1+r} + 2^{-k-1+r} = 5 \cdot 2^{-k-1+r}. \end{aligned}$$

Thus, $|K| \leq 10$. The algorithm becomes:

$$c_{k-r} = \begin{cases} -1 & \text{if } K \leq -3 \\ -1 \text{ or } 0 & \text{if } K = -2 \\ 0 & \text{if } -1 \leq K \leq 1 \\ 0 \text{ or } 1 & \text{if } K = 2 \\ 1 & \text{if } K \geq 3 \end{cases}$$

And, the on-line delay is $(r+2)$.

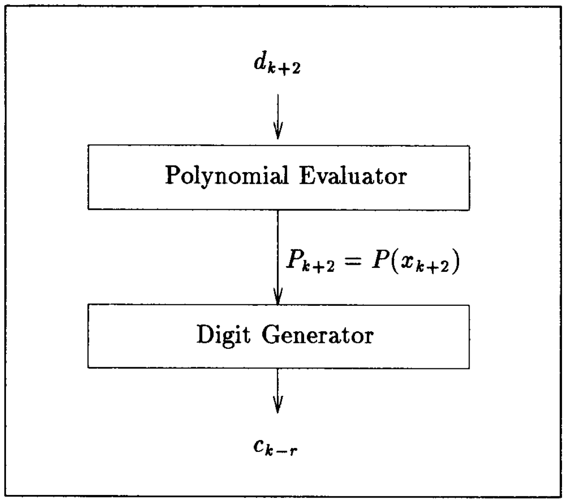

Figure 4: An on-line polynomial evaluator

5 On-line evaluation of inverses of polynomials

To explain the algorithm, we assume that P is an increasing function: $0 < 2^s \leq P'(x) \leq 2^r$, for all $x \in [-1,1]$. Let us denote $y_k = \sum_{i=1}^{k} c_i 2^{-i}$, $(c_i = -1, 0, 1)$. At step k, we assume that we have computed $\hat{P}_{k-\delta-1}$ from $\hat{x}_{k-\delta-1}$ using (3). The integer δ will be the on-line delay. Then, from y_k and $\hat{P}_{k-\delta-1}$, we want to deduce a new digit, $\hat{d}_{k-\delta}$ to obtain $\hat{x}_{k-\delta}$. Our purpose is to show that the algorithm defined by:

$$\hat{x}_{k-\delta} = \hat{x}_{k-\delta-1} + \hat{d}_{k-\delta} 2^{-k+\delta}$$

$$\text{with } \hat{d}_{k-\delta} = \begin{cases} 1 & \text{if } \hat{P}_{k-\delta-1} \le y_k - 2^{-k} \\ 0 & \text{if } |\hat{P}_{k-\delta-1} - y_k| \le 2^{-k+1} \\ -1 & \text{if } \hat{P}_{k-\delta-1} \ge y_k + 2^{-k} \end{cases}$$

where $\hat{x}_i = 0$, $i \le 0$, and $\delta = -s+2$ gives $|\hat{x}_{k-\delta} - x| \le 2^{-k+\delta}$. Initially, $|\hat{x}_0 - x| = |x| \le 1$, and let us assume that

$$|\hat{x}_{k-\delta-1} - x| \le 2^{-k+\delta+1} \qquad (7)$$

- if $\hat{d}_{k-\delta} = 1$, $\hat{P}_{k-\delta-1} - y_k \le -2^{-k}$ and $y_k - y \le 2^{-k}$. Thus, $\hat{P}_{k-\delta-1} - y \le 0$ implies that

$$\hat{x}_{k-\delta-1} - x \le 0$$

From (7), $-2^{-k+\delta+1} \le \hat{x}_{k-\delta-1} - x \le 0$, thus

$$\begin{aligned} -2^{-k+\delta} &\le \hat{x}_{k-\delta-1} + 2^{-k+\delta} - x \\ &= \hat{x}_{-k+\delta} - x \\ &\le 2^{-k+\delta} \end{aligned}$$

- if $\hat{d}_{k-\delta} = -1$, $y_k - \hat{P}_{k-\delta-1} \le -2^{-k}$ and $y - y_k \le 2^{-k}$. Thus, $y - \hat{P}_{k-\delta-1} \le 0$ implies that

$$x - \hat{x}_{k-\delta-1} \le 0$$

From (7),

$$0 \le \hat{x}_{k-\delta-1} - x \le 2^{-k+\delta+1}$$

thus

$$\begin{aligned} -2^{-k+\delta} &\le \hat{x}_{k-\delta-1} - 2^{-k+\delta} - x \\ &= \hat{x}_{-k+\delta} - x \\ &\le 2^{-k+\delta} \end{aligned}$$

- if $\hat{d}_{k-\delta} = 0$, $\hat{x}_{k-\delta} = \hat{x}_{k-\delta-1}$.
Since $|\hat{P}_{k-\delta-1} - y_k| \le 2^{-k+1}$,

$$\begin{aligned} |\hat{x}_{k-\delta} - x| &= |\hat{x}_{k-\delta-1} - x| \\ &\le |\hat{P}_{k-\delta-1} - y| \cdot 2^{-s} \\ &\le (|\hat{P}_{k-\delta-1} - y_k| + |y_k - y|) \cdot 2^{-s} \\ &\le (2^{-k+1} + 2^{-k}) \cdot 2^{-s} \\ &\le 3 \cdot 2^{-k-s} \\ &\le 2^{-k-s+2} = 2^{-k+\delta}. \end{aligned}$$

Let us call K the integer obtained by truncating $(\hat{P}_{k+s-3} - y_k) \cdot 2^k$ after the radix point. Since

$$\begin{aligned} |\hat{P}_{k+s-3} - y_k| &\le |\hat{P}_{k+s-3} - y| + |y - y_k| \\ &\le 2^{-k-s+3} \cdot 2^r + 2^{-k} \end{aligned}$$

we obtain

$$|K| \le 2^{r-s+3} + 1 \le 2^{r-s+4}$$

The comparison between $\hat{P}_{k-\delta-1}$ and y is reduced to the examination of $r - s + 4$ digits of K. Thus, the algorithm becomes

$$\hat{x}_{k-\delta} = \hat{x}_{k-\delta-1} + \hat{d}_{k-\delta} 2^{-k+\delta}$$

$$\text{with } \hat{d}_{k-\delta} = \begin{cases} 1 & \text{if } K \le -2 \\ 0 & \text{if } -1 \le K \le 1 \\ -1 & \text{if } K \ge 2 \end{cases}$$

And, the on-line delay is $(-s + 2)$.

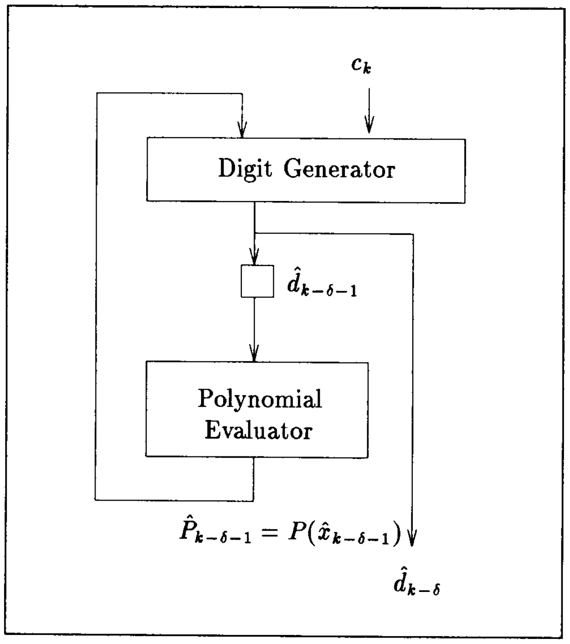

Figure 5: An on-line inverse evaluator

6 Implementation and complexity analysis

	parallel	on-line
the	on-line input	on-line input
polynomials	parallel output	on-line output
the inverse of	parallel input	on-line input
polynomials	on-line output	on-line output

Table 2: Summary of parallel/on-line evaluations

Now, we propose an architecture which computes the sequences $\Phi_k^{(i)}$ of equation (3). Simplifying the term $\frac{(i+j)!}{(i)!(j)!}$, this equation can be rewritten :

$$\Phi_{k+1}^{(i)} = \sum_{j=i}^{n} d_{k+1}^{j-i} 2^{-i} \frac{(j)!}{(i)!(j-i)!} \Phi_k^{(j)}$$

The multiplication by $\frac{(j)!}{(i)!(j-i)!}$ can be avoided using the properties of Pascal's triangle. Let us denote N the number of digits used for representing numbers. The algorithm for computing $\Phi_{k+1}^{(i)}$ becomes:

```
A[i] = $a_i$, $0 \le i \le n$;
C[i][j] = 0, $0 \le i \le n$ and $0 \le j \le i$;
for k = 0 to N - 1
begin
 for l = 0 to n in parallel
 begin
  $C[l][0] = A_{old}[n-l]$;
  $A_{new}[n-l] = 0$;
 end
 for m = 0 to N
 begin
  for all $0 \le i \le n$ and $0 \le j \le i$ in parallel
  begin in parallel
   $C_{new}[i][j] = \frac{1}{2}(C_{old}[i-1][j-1] + C_{old}[i-1][j])$;
   $A[i] = A[i] + d_{k+1}^{m-i} C_{old}[n][i]$;
  end in parallel
 end
end
```

Note that $C[i][j] = 0$ if $i < 0$, $j < 0$ or $j > i$. It is easy to show that at the beginning of the m^{th} iteration of step k,

$$C[n][i] = \frac{m!}{(m-i)!i!} 2^{-m} \Phi_k^{(m)}$$

and that at the end of step $(N-1)$,

$$A[i] = \Phi_N^{(i)}$$

This algorithm can be implemented with a circuit using $\frac{n(n+1)}{2}$ N-digit registers and $O(n^2)$ N-digit adders to compute a degree-n polynomial. For instance, an architecture for evaluating a degree-5 polynomial is described in Figure 6 and has a very regular form, as desired. The time needed to compute a degree-n polynomials is $O(nN)$. The circuit area is $O(n^2N)$.

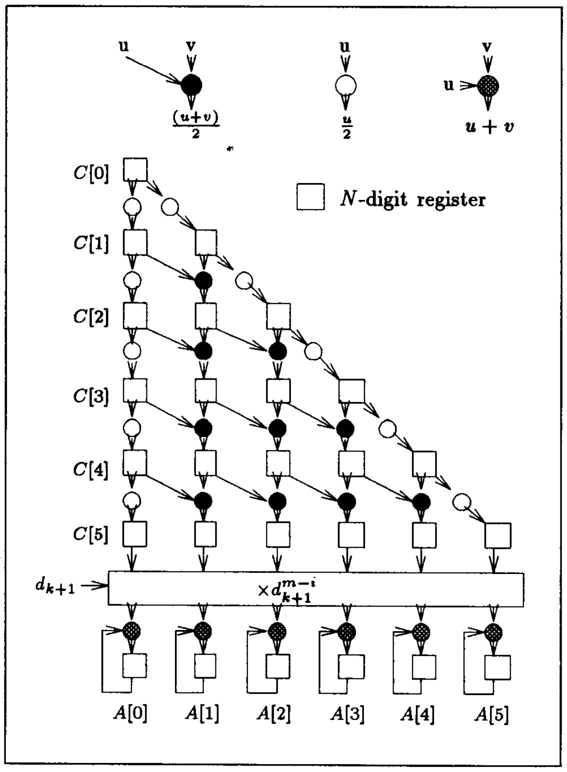

Figure 6: Architecture for the evaluation of a degree-5 polynomial

7 Conclusion

We have proposed new algorithms and architectures for evaluating small degree (say, lower than 10) polynomials both in parallel and in on-line mode. In particular, we have treated the polynomial itself as an operation, not a combination of MA cells. The proposed circuit implementation is very regular, and could be used to approximate most elementary functions (exponentials, logarithms, trigonometric functions). Additionally, the same implementation model can be used to compute the inverse of polynomials.

Acknowledgements

The authors would like to thank all the referees for reading carefully our draft and providing their helpful comments.

References

[1] E. Remes, "Sur un procédé convergent d'approximations successives pour déterminer les polynômes d'approximation," *C.R. Acad. Sci., Paris*, vol. 198, pp. 2063-2065, 1934. in French.

[2] C. Fike, *Computer Evaluation of Mathematical Functions*. Englewood Cliffs, N.J.: Prentice-Hall, 1968.

[3] W. Cody and W. Waite, *Software Manual for the Elementary Functions*. Englewod Cliffs, New Jersey: Prentice-Hall Inc., 1980.

[4] D. Knuth, *The Art of Computer Programming, Vol. 2, Seminumerical Algorithms.* Addison-Wesley, 2nd ed., 1981.

[5] N. Scott, *Computer Number Systems and Arithmetic.* Englewood Cliffs, New Jersey: Prentice Hall, Inc., 1985.

[6] M. Ercegovac, "A general hardware-oriented method for evaluation of functions and computations in a digital computer," *IEEE Trans. Comput.*, vol. c-26, no. 7, pp. 667–680, 1977.

[7] K. Hwang, *Computer Arithmetic Principles, Architecture and Design.* New York: Wiley, 1979.

[8] J. Duprat and J.-M. Muller, "Hardwired polynomial evaluation," *J. Parallel and Distributed Computing*, vol. 5, pp. 291–309, 1988.

[9] G. Corbaz, J. Duprat, B. Hochet, and J.-M. Muller, "Implementation of a VLSI polynomial evaluator for real-time applications," in *ASAP91: International Conference on Application-Specific Array Processors*, (Barcelona, Spain), Sept. 1991.

[10] R. Brackert(Jr.), M. Ercegovac, and A. N. Willson(Jr.), "Design of an on-line multiply-add module for recursive digital filters," in *Proc. 9th IEEE Symp. on Comput. Arith.*, pp. 34–41, IEEE Computer Society Press, 1989.

[11] R. Hartley and P. Corbett, "Digit-serial processing techniques," *IEEE Trans. Circuits and Systems*, vol. 37, no. 6, pp. 707–719, 1990.

[12] J. Bajard, J. Duprat, S. Kla, and J.-M. Muller, "Some operators for on-line radix-2 computations," Tech. Rep. 92-42, LIP-IMAG, Ecole Normale Superieure de Lyon, Oct. 1992. to appear in *J. of Parallel and Distributed Computing.*

[13] K. Trivedi and M. Ercegovac, "On-line algorithms for division and multiplication," *IEEE Trans. Comput.*, vol. c-26, no. 7, pp. 681–687, 1977.

[14] A. Avizienis, "Signed-digit number representations for fast parallel arithmetic," *IRE Trans. Electron. Comput.*, vol. EC-10, pp. 389–400, Sept. 1961.