Some results about on-line computation of functions.

Jean DUPRAT* Yvan HERREROS* Jean-Michel MULLER*, **

*Laboratoire LIP-IMAG, Ecole Normale supérieure de Lyon
46 Allée d'Italie
69364 Lyon Cedex 07
FRANCE
** CNRS, FRANCE

Abstract

We present some theoretical results about on-line algorithms : upper and lower bounds for on-line delays, and properties of functions computable in On Line mode. Then we present the notion of *sparse on line* arithmetic, suitable for manipulation of large numbers.

I. Introduction

Data transmission in digit-serial fashion results in layout improvement for a lot of circuits (signal processing circuits for instance [17]), and in good performances in communication and computation rates, since it enables a digit-level pipelining. Such a transmission may begin with the least significant bits (LSB) or with the most significant bits (MSB) : algorithms for LSB transmission seem more natural (when one performs an addition or a multiplication using the classical "pencil and paper" methods, the digits of the result are generated from the right to the left), but some operations, like division, cannot be performed in LSB mode.

Here, we shall study some aspects of the digit pipelined MSB mode, also called *On Line* mode in the literature ([11], [7], ...). In such a mode, since carries propagate from the least significant bits to the most significant ones, numbers must be in a *redundant* number system which enables computations without carries. Such systems have been studied for a long time : in [1], Avizienis introduces the *signed-digit* systems, where numbers are represented in radix r with digits taken in the digit set {-a, -a + 1, ..., 0, 1, 2, ..., +a} (a ≤ r-1) instead of the conventional digit set {0, 1, 2, ..., r-1}. He shows that if 2a ≥ r+1 (it is impossible if r=2), then a very simple algorithm enables fully parallel addition, without carry propagation. In radix 2 (with a = 1), it is possible to perform parallel additions, but with a slightly different algorithm. Signed-digit systems verifying 2a ≥ r are*redundant* because some numbers have different possible representations. These notations are not the only redundant number systems which enable parallel additions : it is possible to represent the integers in residue number systems or in carry-save form. However, here, we consider only signed-digit representations.

In 1977, Ercegovac and Trivedi present an algorithm for on-line division, based upon the non-restoring division recurrence, and an algorithm for on-line multiplication. Later on, a lot of on-line algorithms have been published (an overview is done in [6], recent algorithms may be found in [3], [16], [5] and [8]. On-line systems are characterized by the initial on-line *delay* (e.g. the number δ such that p digits of the result are deduced from $p+\delta$ digits of the input values, where δ may be either positive or negative) and the *period* (e.g. the time τ between two successive computations of digits). The result of an operation with on-line delay δ and period τ is obtained for p digits of precision in time $(p + \delta)\,\tau$. It is possible to perform successively all on-line operations in a digit-pipelined fashion : the global delay becomes the sum of the individual on-line delays. For instance, in the example of fig. 1 (we suppose that the periods are the same : the different operations are synchronized with the slowest one), the global delay is $2\delta_{mult} + \delta_{add} + \delta_{cos}$, therefore, p bits of the result are computed in time $(p + 2\delta_{mult} + \delta_{add} + \delta_{cos})\tau$.

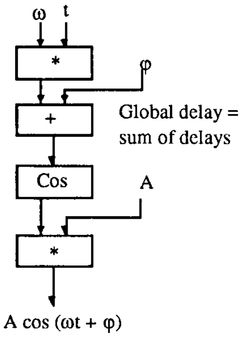

Fig. 1 Digit-level pipelining

A "good" algorithm needs a compromise between delay and period. In the first part of this paper, we are essentially concerned with on-line delays : we present lower bounds of on-line delays for functions with continuous derivatives (these lower bounds are interesting because they are reached by classical algorithms for arithmetic operations). For monotone functions of one variable, we give a higher bound of the lowest delay (it is only of theoretical interest, since the "algorithm" used by the proof has a huge period).

In the last part, we are concerned with very long precision arithmetic. Our aim is to develop methodologies, algorithms and architectures for integer and rational manipulations in computer algebra. Such a field needs very large integers. For instance, in [18], Villard solves exactly linear systems with integer coefficients, and gives the example of a 704x704 system with coefficients bounded by 100, whose rational solution is written with about 2500 radix-10-digit integers. If we suppose that we use fixed-size operators (the size of these operators may be large - a 1000-bit radix-2 on line multiplier is realistic, see [10] - but is necessarily bounded), a full on-line linear-time calculation of some functions, like multiplication or division, becomes impossible. Since we want to maintain at least partially the main advantage of on-line arithmetic, which is the ability of digit-level pipelining, we have to present a "degenerate" on-line mode, which will be called *sparse* on-line arithmetic.

II. Theoretical results.

II.a. General bounds for on-line delays.

In this part of the paper, we shall suppose that numbers are written in a radix-r fixed-point signed digit notation [1], and we shall work in the interval $]-1,1[$. The following definition is the same as that of ercegovac and Trivedi [7].

Definition 1. Let f be a function from $[-1,1]$ to $[-1,1]$. An algorithm for computing f has delay δ if it gives y_p from $x_1, x_2, \ldots, x_{p+\delta}, y_1, y_2, \ldots, y_{p-1}$, where $x = 0.x_1x_2x_3\ldots$ and $y = 0.y_1y_2y_3\ldots$ are represented in signed-digit form, and $y = f(x)$. A function f is computable in on-line mode if there exists an algorithm of finite (which does not depend upon the length of the operands) delay δ which computes f.

Now, we shall define on-line delays of *functions* , instead of delays of algorithms. One may define these on-line delays as follows :

Definition 2. The *absolute delay* $\delta_{abs}(f)$ of a function f is the lowest value of δ such that for any value of p, for any $x \in [-1, 1]$, the first p digits of a redundant representation of f(x) may be deduced from the first $p+\delta$ digits of any redundant representation of x.

But the redundancy of signed-digit notations leads to a problem : let us imagine an algorithm which gives, for a given value of p, the first p digits $y_1, y_2, \ldots, y_p$ of a representation of f(x) from the first $p+\delta$ digits of a representation of x. If we use this algorithm for computing the first $p+1$ digits $y'_1, y'_2, \ldots, y'_{p+1}$ of f(x) , we do not have necessarily $y_i = y'_i$ for any i. The following definition takes this problem in account.

Definition 3. The *practical delay* $\delta_{pra}(f)$ of a function f is the lowest value of δ such that y_p may be deduced from $x_1, x_2, \ldots, x_{p+\delta}, y_1, y_2, \ldots, y_{p-1}$, where $x = 0.x_1x_2x_3\ldots$ and $y = 0.y_1y_2y_3\ldots$ are written in redundant form, and $y = f(x)$. It is the lowest delay of an algorithm which computes f.

This definition seems more suitable for on-line studies than definition 2, but unfortunately, it leads to a delay which seems much more difficult to evaluate. Fortunately, theorem 2 shall prove that these two on-line delays are equal.

Theorem 1. A function computable in On Line mode is continuous in $]-1,1[$.

Proof (given in radix 2).
Let us suppose that f is computable in onLine mode with delay δ. Let us consider a number $x \in]-1,1[$, and $y = f(x)$. We shall show that f is continuous in the neighbourhood of x. First of all, we must observe that there exists a binary redundant representation of x, $0.x_1x_2x_3\ldots$, without an infinite sequence of $\overline{1}$ or 1 at its right (for instance, the non redundant binary representation of x).
For a given integer p, let us consider the numbers :

$$A = 0.x_1x_2x_3\ldots x_{p+\delta}\overline{1111111}\ldots$$
$$B = 0.x_1x_2x_3\ldots x_{p+\delta}11111111\ldots$$

Since f is computable in On Line mode with delay δ, there exist $y_1, y_2, \ldots, y_p$ such that for any $s \in [A,B]$, $y_1, y_2, \ldots, y_p$ are the first digits of a redundant representation of f(s). Therefore, for any $s \in [A,B]$, $| f(s) - f(x) | \leq 2^{-p+1}$. We deduce from that :

$\forall \varepsilon > 0 \ \exists \alpha$ (p is chosen such that $2^{-p+1} < \varepsilon$, $\alpha = \min\{|B-x|, |A-x|\}$) : $|x-s| < \alpha \Rightarrow |f(x)-f(s)| < \varepsilon$.
Thus f is a continuous function and Theorem 1 is proved.

Theorem 1 is not true for an operator whose operands are in conventional binary (nonredundant) form. Let us consider the following example (the number $0.a_1a_2a_3\ldots$ is in non redundant form) :

$$f(0.a_1a_2a_3\ldots) = 0.a_10a_20a_30\ldots$$

This function is not continuous, but is obviously computable MSB first.

Theorem 2.
Let f be computable in on-line mode. $\delta_{abs}(f)$ is equal to $\delta_{pra}(f)$. In the following, we shall denote this value $\delta(f)$.

Proof.
We have obviously : $\delta_{abs}(f) \leq \delta_{pra}(f)$. Let us show that $\delta_{abs}(f) \geq \delta_{pra}(f)$.
For the sake of simplicity, we prove this result in radix 2. Let us note $\delta = \delta_{abs}(f)$. For any x, we can deduce the first p digits of a redundant representation of $y = f(x)$ from the first $p+\delta$ digits of any redundant representation of x. Let us propose the following "algorithm" of delay δ :

We suppose that we have already computed $0.y_1y_2 \dots y_p$ from $0.x_1x_2 \dots x_{p+\delta}$. p+1 digits $0.y_1'y_2' \dots y_{p+1}'$ of a redundant representation of y are deduced from $0.x_1x_2 \dots x_{p+\delta+1}$. Let us chose :

$$y_{p+1} = \begin{cases} 1 \text{ if } 0.y_1y_2 \dots y_p < 0.y_1'y_2' \dots y_{p+1}' \\ 0 \text{ if } 0.y_1y_2 \dots y_p = 0.y_1'y_2' \dots y_{p+1}' \\ \bar{1} \text{ if } 0.y_1y_2 \dots y_p > 0.y_1'y_2' \dots y_{p+1}' \end{cases}$$

We are going to prove that $y_1, y_2, \dots, y_{p+1}$ are the first p+1 digits of a redundant representation of y. In order to do that, let us remark that :

i. $0.y_1'y_2' \dots y_{p+1}'$ and $0.y_1y_2 \dots y_p$ are integer multiples of 2^{-p-1}.

ii. $0.y_1'y_2' \dots y_{p+1}'\overline{1111} \dots \leq y \leq 0.y_1'y_2' \dots y_{p+1}'1111 \dots$
and $0.y_1y_2 \dots y_p\overline{1111} \dots \leq y \leq 0.y_1y_2 \dots y_p1111 \dots$

From *i* and *ii* we deduce easily that $0.y_1'y_2' \dots y_{p+1}'$ is equal to one of the following values (see Fig. 2) :

- $A = 0.y_1y_2 \dots y_p - 2^{-p} - 2^{-p-1}$
- $B = 0.y_1y_2 \dots y_p - 2^{-p}$
- $C = 0.y_1y_2 \dots y_p - 2^{-p-1}$
- $D = 0.y_1y_2 \dots y_p$
- $E = 0.y_1y_2 \dots y_p + 2^{-p-1}$
- $F = 0.y_1y_2 \dots y_p + 2^{-p}$
- $G = 0.y_1y_2 \dots y_p + 2^{-p} + 2^{-p-1}$

- If $0.y_1'y_2' \dots y_{p+1}'$ is equal to D, then the choice $y_{p+1} = 0$ is obviously convenient.

- If $0.y_1'y_2' \dots y_{p+1}'$ is equal to A or B or C, then $y \leq 0.y_1'y_2' \dots y_{p+1}'111 \dots = 0.y_1'y_2' \dots y_{p+1}' + 2^{-p-1}$, therefore $y \leq 0.y_1y_2 \dots y_p$, thus the choice $y_{p+1} = \bar{1}$ is convenient.

- If $0.y_1'y_2' \dots y_{p+1}'$ is equal to E or F or G, then $y \geq 0.y_1'y_2' \dots y_{p+1}'\overline{111} \dots = 0.y_1'y_2' \dots y_{p+1}' - 2^{-p-1}$, therefore $y \geq 0.y_1y_2 \dots y_p$, thus the choice $y_{p+1} = 1$ is convenient.

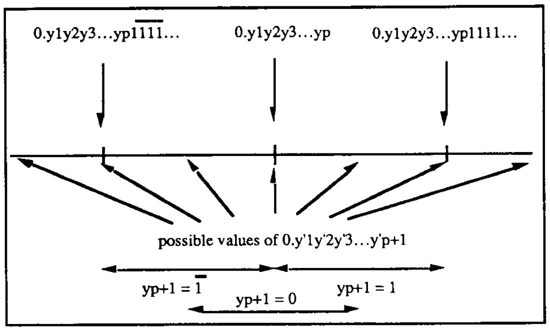

Fig. 2 Different possible configurations.

Theorem 3.
If a function f computable in on-line mode has a continuous derivative, then $\left\lceil \log_r \operatorname*{Max}_{[0,1]} \left|\frac{df}{dx}\right| \right\rceil \leq \delta(f)$.

Proof.
Let us suppose that we want to compute f with a delay $\delta < \left\lceil \log_r \operatorname*{Max}_{[0,1]} \left|\frac{df}{dx}\right| \right\rceil$, in radix r, with the digit set {-a, -a+1, ..., a-1, +a}. Since δ is an integer, we have : $\delta < \log_r (\max | f' |)$. Thus $\max |f'| > r^\delta$.
Since f ' is continuous there exists an interval I such than for any $x \in I$, $|f'(x)| > r^\delta$. Obviously, there exists $x_1, x_2, x_3, \dots, x_{p+\delta}$ such that α and β belong to I, with :

- $\alpha = 0.x_1x_2 \dots x_{p+\delta}0\overline{aaaaa} \dots = 0.x_1x_2x_3 \dots x_{p+\delta} - a\, r^{-\delta} \sum_{i=p+2}^{\infty} r^{-i}$
- $\beta = 0.x_1x_2 \dots x_{p+\delta}0aaaaa \dots = 0.x_1x_2x_3 \dots x_{p+\delta} + a\, r^{-\delta} \sum_{i=p+2}^{\infty} r^{-i}$

$f(\beta) - f(\alpha)$ is equal to $(\beta-\alpha)f'(s)$, where $s \in [\alpha,\beta]$, therefore :

$$|f(\beta) - f(\alpha)| > \left[\left(2ar^{-\delta}\right) \left(\sum_{i=p+2}^{\infty} r^{-i} \right) \right] r^\delta = 2a \sum_{i=p+2}^{\infty} r^{-i}.$$

Since, for any $y_1, y_2, \dots, y_p, y_{p+1}$:

$$0.y_1y_2 \dots y_{p+1}aaaaa \dots - 0.y_1y_2 \dots y_{p+1}\overline{aaaaa} \dots = 2a \sum_{i=p+2}^{\infty} r^{-i}.$$

we deduce that the information "$x_{p+\delta+1} = 0$" (e.g. "$x \in [\alpha,\beta]$") is not sufficient in order to compute the digits $y_1, y_2, \dots, y_p, y_{p+1}$ of f(x).

Theorem 4.
If f has a continuous derivative and is monotone, in radix 2 : $\delta(f) \leq \left\lceil \log_2 \operatorname*{Max}_{[0,1]} \left|\frac{df}{dx}\right| \right\rceil + 1$.

We conjecture that in radix r, $\delta(f) \leq \left\lceil \log_r \operatorname*{Max}_{[0,1]} \left|\frac{df}{dx}\right| \right\rceil + 1$.

Proof.
Let us note :

$$d = \left\lceil \log_2 \operatorname*{Max}_{[0,1]} \left|\frac{df}{dx}\right| \right\rceil + 1$$

Suppose that f is an increasing function (if f is decreasing the proof is similar). We build by induction a (probably very bad) algorithm of delay d which computes y = f(x). Suppose now that we have already calculated the first p-1 digits $0.y_1y_2y_3 \dots y_{p-1}$ of a redundant representation of f(x). The digits $0.x_1x_2x_3 \dots x_{p+d}$ of x are known. Let us note :

$$A = f^{-1}(0.y_1y_2y_3 \dots y_{p-1}0000000000 \dots)$$

and let us define

$$y_p = \begin{cases} 1 \text{ if } 0.x_1x_2 \ldots x_{p+d} > A + 2^{-p-d} \\ \overline{1} \text{ if } 0.x_1x_2 \ldots x_{p+d} < A - 2^{-p-d} \\ 0 \text{ if } A - 2^{-p-d} \leq 0.x_1x_2 \ldots x_{p+d} \leq A + 2^{-p-d} \end{cases}$$

- If $0.x_1x_2x_3\ldots x_{p+d} > A + 2^{-p-d}$ then $x = 0.x_1x_2\ldots x_{p+d}\ldots \geq 0.x_1x_2x_3\ldots x_{p+d}\overline{1111111}\ldots > A + 2^{-p-d} - 2^{-p-d} = A$, thus, since f is increasing, the choice $y_p = 1$ is convenient.

- If $0.x_1x_2x_3\ldots x_{p+d} < A - 2^{-p-d}$ then $x = 0.x_1x_2\ldots x_{p+d}\ldots \leq 0.x_1x_2x_3\ldots x_{p+d}111111\ldots < A - 2^{-p-d} + 2^{-p-d} = A$, thus, since f is increasing, the choice $y_p = \overline{1}$ is convenient.

- If $A - 2^{-p-d} \leq 0.x_1x_2x_3\ldots x_{p+d} \leq A + 2^{-p-d}$ then $A - 2^{-p-d+1} \leq x \leq A + 2^{-p-d+1}$, thus $f(A - 2^{-p-d+1}) \leq f(x) \leq f(A + 2^{-p-d+1})$.

Since f has a continuous derivative :

$$f(A - 2^{-p-d+1}) = f(A) - 2^{-p-d+1} f'(s),\ s \in [A - 2^{-p-d+1}, A].$$
$$f(A + 2^{-p-d+1}) = f(A) + 2^{-p-d+1} f'(t),\ t \in [A, A + 2^{-p-d+1}].$$

Therefore :

$$f(A) - 2^{-p-d+1} f'(s) \leq y \leq f(A) + 2^{-p-d+1} f'(t),$$
$$s \in [A - 2^{-p-d+1}, A],\ t \in [A, A + 2^{-p-d+1}].$$

Therefore :

$$f(A) - 2^{-p}\frac{f'(s)}{\mathrm{Max}_{[-1,1]}|f'|} \leq y \leq f(A) + 2^{-p}\frac{f'(t)}{\mathrm{Max}_{[-1,1]}|f'|}$$

Therefore :

$$f(A) - 2^{-p} \leq y \leq f(A) + 2^{-p}$$

Therefore :

$$0.y_1y_2y_3\ldots y_{p-1}0\overline{11111}\ldots \leq y \leq 0.y_1y_2y_3\ldots y_{p-1}011111\ldots$$

Thus the choice $y_p = 0$ is convenient.

It is worth noting that if f does not have a continuous derivative, even if f is continuous, it may be impossible to compute f in on-line mode. $f(x) = \sqrt{x}$ is an example, since the p^th digit of the square root 0.00...01... (p-1 zeroes) of x = 0.0000...01 (2p-1 zeroes) depends upon the 2p^th digit of x. However, if we use a floating point notation, $\sqrt{x}$ becomes computable. Another example is the following Peano's function :

$f(0.x_1x_2x_3x_4x_5\ldots) = 0.x_1x_3x_5x_7x_9\ldots$ where the numbers are written in binary non-redundant form.

That function is continuous, but has no derivative, and is obviously not computable in on-line mode since the knowledge of the result with accuracy 2^{-p} needs the knowledge of the input value with accuracy 2^{-2p}.

Now, we shall study briefly the case of functions of 2 variables. The following results may be easily generalized to functions of n variables.

Theorem 5.

Let f(x,y) be a function of 2 variables computable in on-line mode :

- $\delta_{abs}(f) = \delta_{pra}(f) = \delta(f)$
- If $\partial f/\partial x$ and $\partial f/\partial y$ exist and are continuous functions, then

$$\delta(f) \geq \left\lceil \log_r \mathrm{Max}\left(\left|\frac{\partial f}{\partial x}\right| + \left|\frac{\partial f}{\partial y}\right|\right)\right\rceil$$

Proof.

- $\delta_{abs}(f) = \delta_{pra}(f) = \delta(f)$: the proof is exactly the same as in theorem 2.
- Let us assume that $\partial f/\partial x$ and $\partial f/\partial y$ exist and are continuous functions. For the sake of simplicity, we suppose that the radix is 2 (in radix r, the proof is similar). We assume that we want to compute f in on-line mode with a delay equal to $\delta < \left\lceil \log_2 \mathrm{Max}\left(\left|\frac{\partial f}{\partial x}\right| + \left|\frac{\partial f}{\partial y}\right|\right)\right\rceil$. Since δ is an integer, we have :

$$\delta < \left(\log_2 \mathrm{Max}\left(\left|\frac{\partial f}{\partial x}\right| + \left|\frac{\partial f}{\partial y}\right|\right)\right)$$

Therefore $2^{\delta} < \mathrm{Max}\left(\left|\frac{\partial f}{\partial x}\right| + \left|\frac{\partial f}{\partial y}\right|\right)$

Since $\partial f/\partial x$ and $\partial f/\partial y$ are continuous, there exist a domain $J = [A,B] \times [C,D]$, $A<B$, $C<D$, such that for any $(x,y) \in J$, $(z,t) \in J$, $\left|\frac{\partial f}{\partial x}\right|(x,y) + \left|\frac{\partial f}{\partial y}\right|(z,t) > 2^{\delta}$, and a domain I, $J \supseteq I$, $I = [E,F] \times [G,H]$, $E<F$, $G<H$, such that the signs of $\partial f/\partial x$ and $\partial f/\partial y$ do not change in I. There exists $x_1, x_2, \ldots, x_{p+\delta}$; $y_1, y_2, \ldots, y_{p+\delta}$ such that $I \supseteq [\alpha,\beta] \times [\gamma,\tau]$, with :

- $\alpha = 0.x_1x_2x_3\ldots x_{p+\delta}0\overline{11111}\ldots$
- $\beta = 0.x_1x_2x_3\ldots x_{p+\delta}011111\ldots$
- $\gamma = 0.y_1y_2y_3\ldots y_{p+\delta}0\overline{11111}\ldots$
- $\tau = 0.y_1y_2y_3\ldots y_{p+\delta}011111\ldots$

Let us note $(\lambda,\mu) = (\gamma,\tau)$ if $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial y}$ have the same sign in I, and (τ,γ) if they have opposite signs.

$f(\beta,\mu) - f(\alpha,\lambda) = (\beta-\alpha).\frac{\partial f}{\partial x}(s_1,t_1) + (\mu-\lambda).\frac{\partial f}{\partial y}(s_2,t_2)$, where :

$$\alpha \leq s_1, s_2 \leq \beta \qquad \lambda \leq t_1, t_2 \leq \mu$$

Therefore : $| f(\beta,\tau) - f(\alpha,\gamma) | > 2^{-p-\delta}.2^{\delta} = 2^{-p}$. Thus the information "$x_{p+\delta+1} = y_{p+\delta+1} = 0$" is not sufficient in order to compute z_{p+1}.

Thus theorem 5 is proved.

II.b. Application to common functions.

Now we apply the preceding analysis to the most common mathematical functions. Lower bounds of on-line delays obtained with the help of theorems 3 and 5 are tabulated in Fig. 3.'Since we work in [-1,1], we have to consider two cases :

- Overflow is possible : the result obtained is not valid if $|f(x)|$ (or $|f(x,y)|$) is greater than 1, e.g. for addition.
- In order to avoid overflow, instead of computing f, we compute f/K, where K is a power of the radix greater than the maximal value of f in [-1,1]. It leads to "artificial" lower on-line delays, since, when f (x) (or f(x,y)) lies in [-1,1], K "artificial" zeros are put in the beginning of the result.

An interesting remark is that for addition and multiplication in radix r>2, these on-line delays are achieved (we shall consider later the problem of radix 2 addition). Hence the theoretical bounds presented here are realistic. The following theorem shows that the delay of radix-2 addition is 2.

Theorem 6. In radix 2, $\delta(+) = 2$.

Proof. Since methods of delay 2 are known (see [10] for instance), we have only to prove that $\delta(f) > 1$. In order to do that, suppose that we are able to perform on-line additions with delay 1.

Let x and y be two elements of]-1,1[. Assume that we have already computed, from p+1 digits ($0.x_1x_2...x_{p+1}$ and $0.y_1y_2...y_{p+1}$) of x and y, the first p digits of a representation of $z = x+y$, say $0.z_1z_2...z_p$. Let us define :

- $\alpha = 0.x_1x_2x_3...x_{p+2}\overline{11111}...$
- $\beta = 0.x_1x_2x_3...x_{p+2}11111...$
- $\gamma = 0.y_1y_2y_3...y_{p+2}\overline{11111}...$
- $\tau = 0.y_1y_2y_3...y_{p+2}11111...$

x may be any element of $[\alpha,\beta]$, and y may be any element of $[\gamma,\tau]$, therefore z may be any element of $I_1 = [\alpha+\beta,\gamma+\tau]$. Let us suppose that the knowledge of the new digits x_{p+2} and y_{p+2} is sufficient in order to compute z_{p+1}. The numbers which may be written with $0.z_1z_2...z_pz_{p+1}$ as p+1 first digits are those of the interval $I_2 = [0.z_1...z_{p+1}\overline{1111}...,\ 0.z_1..z_{p+1}111...]$, therefore, I_2 must contain I_1. Since the length of I_1 and I_2 are equal (to 2^{-p}), I_2 must be equal to I_1, thus $0.z_1...z_{p+1}$ must be equal to $\frac{1}{2}(\alpha+\beta+\gamma+\tau) = 0.x_1x_2x_3...x_{p+2} + 0.y_1y_2y_3...y_{p+2}$. It is impossible if, for instance, $x_{p+2} = 0$ and $y_{p+2} = 1$, since in such a case $2^{p+2}(0.x_1x_2x_3...x_{p+2} + 0.y_1y_2y_3...y_{p+2})$ is an odd integer while $2^{p+2}(0.z_1z_2...z_pz_{p+1})$ is necessarily even (its binary representation is $z_1z_2...z_pz_{p+1}0$).

Now, let us consider the particular case of division (and reciprocation). We shall suppose that numbers treated here are mantissas of redundant floating-point numbers. Such a floating-point number, with mantissa m can be :

- Normalized if $\frac{1}{r} \leq |m| < 1$
- Quasi-normalized if $\frac{1}{r^2} \leq |m| < 1$
- Pseudo-normalized if $\frac{1}{r^p} \leq |m| < 1$, with $p \geq 3$.

At least, normalized and quasi-normalized cases have to be considered. Practically, it is very difficult to ensure that a redundant floating-point number is normalized (it is straightforward in non-redundant notation), but the examination of the signs of its two most significant digits allows to check if it is quasi-normalized (and quasi-normalization of a number may be performed easily in on-line mode). Therefore, if the input data is non-redundant, we have to consider the normalized case, else we have to consider the quasi-normalized case. In [16], Lin and Sips present a reciprocal algorithm whose delay is, for radix 2 numbers, 3 and 4 to 5 for normalized and quasi-normalized inputs respectively; and for radix-r numbers, r≥4, 1 and 2 to 3 for normalized and quasi-normalized inputs. We deduce from the result of Lin and Sips, and from the table of Fig. 3, that for radix-r numbers (r≥4), the minimal delay of reciprocation is 1 for normalized numbers, and 2 or 3 for quasi-normalized numbers. This implies that the delay of division for radix-r numbers (r≥4), is 2 for normalized numbers, and 3 or 4 for quasi-normalized numbers.

f(x) or f(x,y)	interval	$\max_I \lvert f'\rvert$	$\lceil \log_2 \max_I \lvert f'\rvert \rceil$	$\lceil \log_r \max_I \lvert f'\rvert \rceil$
sin x	[-1 , 1]	1	0	0
cos x	[-1 , 1]	sin 1	0	0
e^x	[-1 , 1]	e	2	2
$e^x/4$ (radix 2)	[-1 , 1]	e/4	0	0
$e^x/r)$ (r > 2)	[-1 , 1]	e/r	0	0
ln (x)	[1/e , 1]	e	2	1 if r > 2
x + y	[-1 , 1]	2	1	1
(x+y)/r	[-1 , 1]	2/r	0	0
xy	[-1 , 1]	2	1	1
1/rx	[1/r , 1]	r	1	1
$1/(r^2x)$	$[1/r^2 , 1]$	r^2	2	2
$1/(r^px)$	$[1/r^p , 1]$	r^p	p	p
x/ry	[1/r , 1]	1 + r	2	2
$x/(r^2y)$	$[1/r^2 , 1]$	$1 + r^2$	3	3
$x/(r^py)$	$[1/r^p , 1]$	$1 + r^p$	p+1	p+1
$\sqrt{x}$	$[r^{-p} , 1]$	$0.5\, r^{p/2}$	$\lceil p/2 - 1 \rceil$	$\lceil p/2 - \log_r 2 \rceil$

Fig. 3 Lower bounds of on-line delays for usual functions . (for a function of 2 variables, $|f'|$ represents $|\partial f/\partial x| + |\partial f/\partial y|$).

III. Sparse on-line arithmetic.

In the preceding part, we have considered only theoretical dependency relations. Now, in order to study practical implementations of functions in high precision arithmetic, we have to consider two kinds of functions : those computable on-the-fly with fixed sizes operators, and the other functions. Let us suppose that we want to perform arbitrarily long precision arithmetic, and that we have fixed-size operators. Some calculations, like addition, remain obviously possible in linear time on-line mode. However, a lot of computations, like multiplication or division, become impossible in linear time. Our purpose is to conserve, for these computations, the great advantage of on-line arithmetic, e.g. , the digit-level pipelining. In order to do that, we define the concept of *sparse on-line* arithmetic.

III.a. Functions not computable in linear time with fixed-size operators.

Let us consider the problem of multiplying two large n-digit integers. Since we have only a bounded-size multiplier, the time complexity of multiplication is the same as with purely sequential processes, e.g. the same as in multiprecision software. In 1969, Cook and Aandera showed in [2] that this complexity is lower-bounded by $O\left(\frac{n \log n}{(\log \log n)^2}\right)$. Therefore, our multiplication cannot be performed in linear time. Since the same lower bound is true for division, the result is the same. Practically, only some piecewise linear functions like addition, maximum, average value, opposite, absolute value, etc. appear to be computable in linear time.

III.b. Use of fixed-size operators : sparse on-line arithmetic.

We take as time unit value the delay between the introduction of two consecutive digits in the operator. At each time unit, our operators give a "digit" value. Since some operations cannot be performed at this frequency, some of these digit values will have no sense and will not appear in the final representation of the result. Such digits with no sense will have a special representation, and will be called *holes.*

Since our operands have no constant sizes, we have to introduce an other kind of non numeric digit, called *end of number (eon)* . A number ends with an *eon* symbol, and sparse numbers contain "true" digits and holes.

Let us consider again the problem of multiplying large numbers. There are some ways to perform large multiplications using small multipliers. If our operands are 2p-digit radix r numbers :

$$A = 0.a_1a_2...a_{2p} = A_1 + r^{-p}A_2,$$
$$\text{with } A_1 = 0.a_1...a_p \text{ and } A_2 = 0.a_{p+1}...a_{2p}$$

$$B = 0.b_1b_2...b_{2p} = B_1 + r^{-p}B_2,$$
$$\text{with } B_1 = 0.b_1...b_p \text{ and } B_2 = 0.b_{p+1}...b_{2p}$$

The product P = AB may be expressed as :

$$P = A_1B_1 + (A_1B_2 + A_2B_1)\, r^{-p} + A_2B_2r^{-2p} \qquad (1)$$

With such a decomposition, a 2p-digit product leads to 4 p-digit products. There are better decompositions (the decomposition of Karatsuba and Ofman [15] leads to 3 p-digit products, Cook and Aandera [2] study similar decompositions), but for the sake of simplicity we assume that we use decomposition (1). Let us assume that we have a pxp-digit on line multiplier. Let us number (1) the digits of weigth r^0 to r^{-p+1} of A_1B_1, (2) the digits of weigth r^{-p} to r^{-2p+1} of A_1B_1, (3) the digits of weigth r^{-p} to r^{-2p+1} of A_1B_2, and so on, as depicted in Fig. 4.

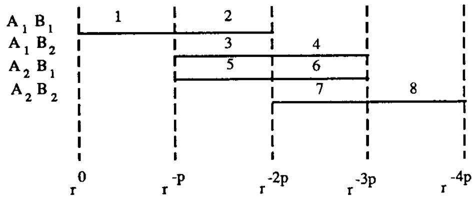

Fig. 4 Diagram of the decomposition.

The multiplier starts with the product A_1B_1. The digits of (1) are given in classical on-line mode. After that, during the computation of (2), (3) and (4), which are stored, we output holes. During the computation of (5), we can output in on-line mode the digits of weigths r^{-p} to r^{-2p+1}, and so on, as depicted in Fig.5.

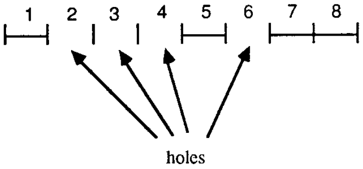

Fig. 5 holes of the result.

If, instead of a pxp-digit multiplier we have a (p/2)x(p/2) digit multiplier, each of the products A_1B_1, A_1B_2, A_2B_1 and A_2B_2 is decomposed as AB was decomposed previously, and we obtain holes into the parts 1, 5, 7 and 8 of Fig. 5.

Conclusion.

We have given here some complexity results which allow to determine exactly or to bound the on line delay of most common arithmetic and elementary functions. It allows to show that a lot of classical on-line operators presented in the literature are optimal in delay (but not necessarily in period). We have proposed a way to conserve, for large numbers manipulations, the main advantage of on-line arithmetic, which is the ability of digit-level pipelining, by presenting *sparse on-line* arithmetic. Now, we have to develop algorithms and operators especially designed for sparse mode.

References.

[1] A. Avizienis, *Signed-digit number representations for fast parallel arithmetic* , IRE Transactions on electronic computers, 10, pp. 389-400, 1961.

[2] S.A. Cook and S.O. Aandera, *On the minimum computation time of functions* , Trans. of the AMS, 142, 1969, pp. 291-314.

[3] M.D. Ercegovac et T. Lang, *A division algorithm with prediction of quotient digits* , 7th Symposium on computer arithmetic, Urbana, Illinois, June 1985.

[4] M.D. Ercegovac et T. Lang, *On-the-fly conversion of redundant into conventional representations* , IEEE Trans. on Computers, Vol. C-36, No 7, July 1987.

[5] M.D. Ercegovac et T. Lang, *On-line scheme for computing rotation factors* , 8th Symposium on computer arithmetic, Como, Italy, May 1987, IEEE Publ. No 87CH2419-0.

[6] M.D. Ercegovac, *On-line arithmetic : an overview* , SPIE Vol. 495, Real time signal processing VII, pp 86-93, 1984.

[7] M.D. Ercegovac et K.S. Trivedi, *On line algorithms for division and multiplication* , IEEE Trans. on Computers, Vol. C-26 No 7, pp 681-687, July 1977.

[8] M.D. Ercegovac et P.K.G. Tu, *A radix-4 on-line division algorithm* , 8th Symposium on computer arithmetic, Como, Italy, May 1987, IEEE Publ. No 87CH2419-0.

[9] A.L. Grnarov et M.D. Ercegovac, *On the performance of on-line arithmetic* , Proc. 1980 Intern. Conference on parallel processing, IEEE Publ. No 80CH1569-3, pp 55-62, Aug.1980.

[10] A. Guyot, Y. Herreros and J.M. Muller, *JANUS, an On-line Multiplier/divider for manipulating large numbers* , 9^{th} Symposium on Computer Arithmetic, Santa Monica, Sept. 1989.

[11] M.J. Irwin, *An arithmetic unit for Online computation*, PhD thesis, tech. report UIUCDCS-R-77-873, Dept. of Computer science, university of Illinois, Champaign-urbana, IL 61801, May 1977.

[12] M.J. Irwin, *A pipelined processing unit for on-line division* , Proc. 5th symposium on Computer architecture, IEEE Publ. No 78CH1284-9C, pp 24-30, April 1978.

[13] M.J. Irwin et R.M. Owens, *On-line algorithms for the design of pipeline architectures, 6th symposium on Computer Architecture* , Philadelphia, PA, April 1979.

[14] M.J. Irwin et R.M. Owens, *Digit-pipelined arithmetic as illustrated by the paste-up system : a tutorial* , IEEE Computer, pp 61-73, April 1987.

[15] A. Karatsuba and Y. Ofman, *Multiplication of large numbers on an automata* , Dokl. Akad. Nauk. SSSR, 145, 1962, pp. 293-294 (in russian).

[16] H. Lin and H.J. Sips, *A novel floating-point on-line division algorithm* , 8th Symposium on computer arithmetic, Como, Italy, May 1987, IEEE Publ. No 87CH2419-0.

[17] G. Privat, *Architectures spécialisées de circuits VLSI pour le traitement du signal* , Ph.D. dissertation, Ecole Nationale Supérieure des Télécommunications, France, 1986.

[18] G. Villard, *Calcul formel et parallélisme, résolution de systèmes linéaires* , Ph. D. dissertation, TIM3-INPG, Grenoble, France, Dec. 1988.